A Survey of Methods for Generating a Test Sequence for Conformance Testing of Finite State Machine

Zainab Eid
King Abdul Aziz University
Jeddah, Saudi Arabia
zainabeid@hotmail.com

Hanan Nadeem
King Abdul Aziz University
Jeddah, Saudi Arabia
hanannadeem@hotmail.com

Reem Al Gamdi
King Abdul Aziz University
Jeddah, Saudi Arabia
Reem3820@hotmail.com

Abstract
Finite state machines testing problems has been studied due to its applications to conformance testing of communication protocol; we will focus on the primary problems of conformance testing (or fault detection problem) based on knowing where we are during the experiment, using status message, reset and distinguishing sequence. Given a specification machine we have its state transition and output functions in the form of a transition diagram or "black box" which we can only observe its input/output behaviour, and we will design a test to determine whether implementation conforms to the specification by generating and using test sequence called checking sequence. We discuss three methods for generating checking sequence: Status Message and Reset, Distinguishing Sequence Method and Using Separating Sequences Instead of Status Massages. Finally, we will review a small comparison between those methods about how we can choose between them.

1. Introduction
An FSM (finite state machine) is a machine of a finite number of states produces outputs on state transitions after receiving inputs. FSM are widely used to model systems in diverse areas including sequential circuits, particular types of programs (represented by lexical analysis, pattern matching etc.), and more recently communication protocols. An FSM can be described by a quintuple M = (Q,Σ,δ,q0,F) where:
- Q is a set of finite internal states,
- Σ is a set of finite input alphabet
- δ:Q×Σ→Q is a total functions called the transition function which represent that the machine M is moving from state q to the state δ(q,a) when receiving an input a.
- q0 in Q is the start state,
- F ∈ Q; elements of F are called accept or final states.[11]

An FSM can be also represented by a state transition diagram a directed graph whose vertices correspond to the states of the machine and whose edges correspond to the state transitions and labelled with the input and output of that transition as shown in Fig. 1.

Testing FSM problem has been studied in different areas and at various times and the area seemed to have mostly died down until a few years ago when the testing problem was resurrected and is now being studied anew due to its applications to conformance testing of communication protocol. It is very wide fields of testing hardware and software includes with an extensive literature which we cannot hope to include here we will focus on the basic problems of conformance testing (or fault detection problem).

There are two types of FSM (finite state machines): Mealy and Moore machines. The theory of both types is very similar, we mainly consider Mealy machines because its model finite state systems more properly. And also we have two types of testing problems:
1.1. In the first one we have a machine M with a complete description and we have the transition diagram of it but we lack some information about in which state it is

Problem 1: determining the final state after the test.

Problem 2: state identification problem to identify the initial state and the test sequence that solve this problem is called distinguishing sequence.

Problem 3: state verification problem which verify that the machine in the specified state and the test sequence that solve this problem is called unique input/output UIO sequence.

1.2. In the second type of testing the machine M that is being tested is either a black box i.e. we don’t know its state diagram (the transition and output function) and only we can recognize its I/O behavior or we have and know the machine transition diagram, and we have to test whether the implementation conforms to the specification which called the conformance testing or fault detection problem and the test sequence that solve this problem is called checking sequence.

In our paper, we will discuss 3 methods for constructing checking sequence for the conformance testing problem. In section II, we will discuss the concept of conformance testing problem, in section III we describe the Status Message and Reset message. In section IV the Distinguishing Sequence Method is described, and in section V Using Separating Sequences Instead of Status Massages including(W method, Wp method, UIO method, Distinguishing Sequence Method). In section VI we will describe the Using Identifying Sequences Instead of Distinguishing Sequences method.

2. Conformance Testing

Given an FSM $M_S$ which acts as a specification and for which we know its transition diagram, and another finite state machine $M_I$ which we can only observe its behavior. We want to test whether $M_I$ correctly implements or conforms to $M_S$. Conformance is formally defined as equivalence or isomorphism where $M_I$ conforms to its specification $M_S$ if and only if their initial states are equivalent, i.e. they will produce the same output for every input sequence, and in general two states $s_i$ and $s_j$ are equivalent (iff) if and only if for every input the machine generate the same output[12]. To prove this equivalence of $M_I$ and $M_S$, we have to look for a set of input sequences to apply it to the machine $M_I$ to prove that it is equal to its specification and that input sequence is called checking sequence. We are interested to find if $M_I$ fails to implement $M_S$ or not so the problem of conformance testing is called fault detection. The method of testing and detection the fault in the $M_I$ is as follows:

1. Generate from the machine $M_S$ a set of input sequences.
2. Applying each input sequence to $M_S$, produce the required output sequences.
3. Each pair of the input sequence and prospective output is a test and the set of all tests is a test suite.
4. Apply each of the input sequence to the machine $M_I$ and observe the output.
5. Compare actual output with the expected one, and if they differ, then the fault is detected [13].

This procedure of testing as it presented so far, can only be used to show the presence of bugs, but never to show their absence[13]. Generating a checking sequence (set of sequences that concatenated act as a unique checking sequence), is differ for their cost to produce test sequences, for the total size of the test suite (i.e. the total length of the checking sequence), and for their fault detection capability. In fact, test suites should be rather short to be applicable in practice, and it should cover the implementation as much as possible and detect as many faults as possible, we will use different methods to achieve these two opposite goals in which the main difference among those methods are in the assumptions they make about the machines $M_S$ and $M_I$. Some methods are very efficient in producing a checking sequence but can be restricted under a strong assumptions while Other methods produce exponentially long sequences, but perform the test under more general assumptions, we can make a decision of choosing between methods depending on the facts we have or assumed about the two machines $M_S$ and $M_I$, and these assumptions are of high importance because testing machine without any assumption is impossible and we have to introduce some assumptions about the machines we want to verify and these assumptions is as follows:

1. $M_S$ is reduced or minimal: because equivalent machines have the same I/O performance, and it is impossible to distinguish them by observing the outputs; we can minimize the machine $M_S$ and obtain an equivalent reduced machine and In that minimal machine there are no equivalent states. For every states $s$ and $t$, it should exist a sequence $x$, called separating sequence which distinguish $s$ from $t$ because the set of outputs generated from applying $x$ to $s$ are differing.
2. $M_I$ is completely specified: both of state transition function $\delta$ and output function $\lambda$ are defined for every state in $Q$ and every input in $\Sigma$. 

3. **M₅ is strongly connected:** which mean every state is reachable from every other via one or more state transitions.

4. **M₄ does not change during testing,** in case it has same inputs and outputs as M₅, which mean M₄ can accept and respond to all input symbols in $\Sigma$.

5. **Initial state:** the machines M₂ is in its initial state before we conduct a conformance test experiment. If M₂ is not in its initial state, we can apply a homing sequence.

6. **Same number of states:** M₂ and M₅ has the same number of states. Depending on this assumption the possible faults in M₂ could be one of two kinds: output faults, occurs because transition in the implementation produces wrong output, and transfer faults, in case of implementation goes to the wrong state. Fig. 2 shows two faulty implementations of the specification machine M₂ given in Fig. 1.

7. **reset message:** for all $s \in \Sigma$, $\delta(s,\text{reset}) = s_{1}$ and the output function $\lambda(s,\text{reset}) = -$ which mean that the machine has a particular input reset that from any state $s$ causes a transitions ends in the initial state $s_{1}$ with no output produced.

8. **status message:** if we label the states $s_{1}, s_{2},..., s_{n}$, we assume that the $\text{status}$ outputs the index $i$ when applied to state $s_{i}$, and the machines do not change state. Formally for all states $s \in S$, $\lambda(s,\text{status}) = i \in \delta(s,\text{status}) = s_{i}$ so the status message generates an output that uniquely identifies their current state.

9. **set message:** when a particular set of inputs set($s_{i}$) message is received in the initial system state, the machines move to state $s_{i}$ without producing any output. Formally for all $t \in \Sigma$, $\delta(\text{reset},\text{set}(t)) = t$ and $\lambda(s,\text{set}(t)) = -$ [13][14].

The first four properties listed above are requirements and without them a conformance test of the type to be discussed is not possible, the rest of the assumptions are convenient but not essential. Conformance test has many methods that generate the checking sequence using those assumptions and some of these methods based on knowing where we are during the experiment using the present or absent of set, reset, and status messages which we will discuss in our paper in next sections.

![Figure 2](image-url)
polynomial time algorithm for constructing a checking sequence [1].

4. distinguishing sequence methods

Distinguishing sequence can be used as unreliable status message because it gives a different output for each state; it is like a status message, except that it moves the machine to another state when applied. This method has two phases to check whether the implementation machine $M_i$ is equivalent to the specification $M_S$. Now it’s the time for establishing every state transition in phase2: we want to check the transition from state $s_i$ to $s_j$ with input/output $a/o$ when the machine is currently in state $t_k$. We would first take the machine from $t_k$ to $s_i$, apply input $a$, observe the output $o$, and verify the ending state $s_j$. We cannot simply use $\tau(t_k, s_i)$ to take the machine to state $s_i$, since faults may alter the ending state. Instead, we apply the following input sequence: $\tau(t_k, s_i) - I \times \tau(t_i - I, s_j)$ [1]. First transfer sequence is assumed to take the machine to state $s_i - I$, which is verified by its response to $x$, and as has been verified by (1), $\tau(t_i - I, s_j)$ definitely takes the machine to state $s_i$. We then start testing the transition using input $a$ and verify the ending state by $x$. Therefore, the following sequence tests for a transition from $s_i$ to $s_j$: $\tau(t_i, s_i - 1) \times \tau(t_i - 1, s_j) a x \ldots(2)$

\[
\begin{array}{cccccc}
 & x & t(1, s_2) & x & t(2, s_3) & x & t(3, s_1) & x \\
Checking sequence & ab & ab & ab & ab \\
output & 01 & 11 & 00 & 01
\end{array}
\]

machine $M_S$; It first builds an input sequence that visits each state using transfer sequences instead of reset and then applies its distinguishing sequence to test whether $M_i$ is similar to $M_S$. It then builds an input sequence to test each transition to guarantee that $M_i$ conforms to $M_S$.

**Definition (Transfer Sequence):** A transfer sequence $\tau(s_i, s_j)$ is a sequence that takes the machine from state $s_i$ to $s_j$; And since the machine is strongly connected, so that sequence is always exists for any two states by the assumption, **Phase 1:** by applying the distinguishing sequence $x$ starting from state $s_i$ and let $t_i$ be the final state which denoted by $t_i = \delta(s_i, x)$ and $\tau(t_i, s_i + 1)$ is the transfer sequence from $t_i$ to $s_i + 1$. The input sequence that checks the response of the machine to the distinguishing sequence in each state starting from the initial state $s_i$ is: $x \tau(t_i, s_i) x \tau(t_2, s_3) x \ldots \tau(m, s_i) x$ (1) and we can represent it as follows:

![Diagram of state transition](attachment:image)

Starting in state $s_i$, $x$ takes the machine to state $t_j$ and then $\tau(t_j, s_j)$ transfers it to state $s_2$ for its response to $x$. In the end the machine responds to $x \tau(t_n, s_j)$. If it operates correctly, it will be in state $s_i$, and this is verified by its response to the final $x$. During the test, we should recognize n different responses to the distinguishing sequence $x$ from n different states, and this verifies that the implementation machine $M_i$ is similar to the specification machine $M_S$. Now it’s the time for establishing every state transition in phase2: we want to check the transition from state $s_i$ to $s_j$ with input/output $a/o$ when the machine is currently in state $t_k$. We would first take the machine from $t_k$ to $s_i$, apply input $a$, observe the output $o$, and verify the ending state $s_j$. We cannot simply use $\tau(t_k, s_i)$ to take the machine to state $s_i$, since faults may alter the ending state. Instead, we apply the following input sequence: $\tau(t_k, s_i) - I \times \tau(t_i - I, s_j)$ [1]. First transfer sequence is assumed to take the machine to state $s_i - I$, which is verified by its response to $x$, and as has been verified by (1), $\tau(t_i - I, s_j)$ definitely takes the machine to state $s_i$. We then start testing the transition using input $a$ and verify the ending state by $x$. Therefore, the following sequence tests for a transition from $s_i$ to $s_j$: $\tau(t_i, s_i - 1) \times \tau(t_i - 1, s_j) a x \ldots(2)$

After performing this sequence, the machine will be in state $t_j$. We repeat the same process for each state transition and obtain a checking sequence. Note that the checking sequence length is a polynomial in the size of the machine $M_S$ and the length of the distinguishing sequence $x$ [1]. Example: For a machine in Fig.1, the distinguishing sequence is $x = ab$ and the responses from state $s_1$, $s_2$, and $s_3$ are $01$, $11$, and $00$ respectively. When we apply the distinguishing sequence in states $s_1$, $s_2$, and $s_3$ it will take the machine to $t_1 = s_2$, $t_2 = s_3$ and $t_3 = s_1$. And the transfer sequences can represented by $\tau(t_1, s_2) = \tau(t_2, s_3) = \tau(t_3, s_1) = e$. The sequence (1) becomes:

<table>
<thead>
<tr>
<th>Checking sequence</th>
<th>ab</th>
<th>ab</th>
<th>ab</th>
<th>ab</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
</tbody>
</table>

We can see that this input ends in state $t_2 = s_2$. The input sequences (2) can be concatenated to obtain:

<table>
<thead>
<tr>
<th>trans. to test</th>
<th>b0</th>
<th>a/1</th>
<th>a/0</th>
<th>a/0</th>
<th>b/1</th>
<th>b/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1-S3</td>
<td>S2-S2</td>
<td>S3-S3</td>
<td>S1-S1</td>
<td>S2-S3</td>
<td>S1-S2</td>
<td>S1-S2</td>
</tr>
</tbody>
</table>
5. Using Separating Sequences Instead of Status Massages

Unfortunately, a status message is seldom available. In the next section, we present how not to rely on a status message to determine the current state during the test. We assume now that the machines have no status message (but they still have a reset message), and we wish to test whether $M_S$ is equivalent to $M_I$ only observing the external behaviour. In the following, we present some methods that can be unified as proposed by Lee and Yannakakis [2]. All these methods share the same technique and used to identify the state by replacing the use of the status message with many kinds of sequences called in generally the separating sequences [2] used to identify the state which they applied to. Remember that by assuming $M_S$ is minimal, it for sure does not contain two equivalent states, in another word, for every pair $s_i$, $s_j$ there must exist an input $x$ that we call separating sequence, and that input will distinguish them because it will produce different outputs, i.e. $\lambda(s_i,x) \neq \lambda(s_j,x)$.

5.1. W Method:

This method use specific separating sequences that is called characterizing set and another set called transition cover set which visit each transition in the machine.

Definition 2. transition cover set of $M_S$ (or $P$ set for short) is a set of input sequences such that for each state $s \in S$ and each input $a \in I$ there exist an input sequence $x \in P$ starting from $s_1$ the initial state and ending by applying $a$ to state $s$. Formally $\forall s \in S$ and $\forall a \in I \exists$ sequence $y \in P : x = y.a$ and $\delta(s_1,y) = s$. [3][14]. A $P$ set forces the machine to perform every transition and then stop. A $P$ set can be built by using a standard breadth-first visit of the transition diagram of the machine $M_S$. Note that a $P$ set is closed under the operation of selecting a prefix: if $x$ belongs to $P$, then any prefix of $x$ is in $P$ too. One way of constructing $P[3]$ is to build first a testing tree $T$ of $M_S$ as explained in Algorithm 1 and then get the input generated from the partial paths of $T$. Where a partial path is a sequence of consecutive branches, starting from the root and ending in a terminal or non-terminal node. Because the branches in $T$ are labeled by an input symbol, so the sequence generated from a partial path $q$ is the input symbols on $q$. In addition to that, the empty input is a part of any $P$ set. Note that

<table>
<thead>
<tr>
<th>input sequence</th>
<th>$t(t_1,s_3)bx$</th>
<th>$t(t_1,s_3)ax$</th>
<th>$t(t_2,s_3)ax$</th>
<th>$t(t_3,s_2)ax$</th>
<th>$t(t_3,s_2)bx$</th>
<th>$t(t_3,s_1)bx$</th>
</tr>
</thead>
<tbody>
<tr>
<td>end state</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>output</td>
<td>1001</td>
<td>111</td>
<td>000</td>
<td>001</td>
<td>100</td>
<td>111</td>
</tr>
</tbody>
</table>

The checking sequence length is 27.

Instead of using a unique present distinguishing sequence for all the states, we can use an adaptive distinguishing sequence; (ADS) which is a decision tree specifies how to choose the next input adaptively depending on the spotted output to identify the initial state[14].

<table>
<thead>
<tr>
<th>input sequence</th>
<th>$t(t_1,s_3)bx$</th>
<th>$t(t_1,s_3)ax$</th>
<th>$t(t_1,s_3)bx$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Length and Cost of ADV: it has $O(n^2)$ length with transfer sequence cannot be longer than $n$. The sequence $(1)$ is long $O(n^2)$. Because there are $pn$ transitions, and every sequence $(2)$ has the length $O(n^2)$, and the cost is again $O(pn^2)$ to find the complete checking sequence. The advantage of this method is that it does not need a reset message.
Algorithm 1 terminates because the number of states is finite.

**Algorithm 1 (Building a test tree)**

1. Denote the root of the tree with the initial state \( s_1 \). And this is the level 1
2. Because we already built the tree up to level \( k \), so we have to continue with the \( k + 1 \) level.
   a) With each node \( t \) at level \( k \) from left to right
   b) If the node \( t \) is equal to another one at level \( j \), with \( j \leq k \), then \( t \) is terminated and considered a leaf of \( T \)
   c) Otherwise, for every input \( x \), if the machine moves from state \( s_i \) to \( s_j \), we attach a branch with label \( x \) and a node with label \( s_j \)

**Example:** A test tree for \( M_3 \) of Fig. 1 is shown in Fig. 4. From this test tree we obtain \( P = \{\varepsilon, a, b, ba, bb, bba, bbb\} \).

![Figure 4. A test tree for MS of Figure 1](image)

To test every transition of \( M_t \) the \( W \) method uses a \( P \) set and uses another set instead of the status message, called characterizing set or \( W \) set, which verify that we have the one expected end state of each transition.

Definition 3. Characterizing set is a set \( W \) of input sequences such that for every pair of distinct states \( s \) and \( t \) in \( S \), there exists an input sequence \( x \) in \( W \) such that \( \lambda(s, x) \neq \lambda(t, x) \) [14]. The characterizing set is briefly called \( W \) set or sometimes separating set. The input sequences in the \( W \) set are also called separating sequences. A \( W \) set exists for every machine that is minimal (Assumption 1). The choice of a \( W \) set is not unique, and the fewer are the elements in the \( W \) set, the longer are the separating sequences in the \( W \) set. An algorithm for building a \( W \) set as follows: Partition the set of states \( S \) into blocks \( B_i \) with \( i = 1, \ldots, r \). Initially \( W \) is \( \emptyset \), \( B_1 = S \) and \( r = 1 \). Until every \( B_i \) is a singleton, take two distinct states \( s \) and \( t \) in a \( B_i \) (that includes at least two states) and build their separating sequence \( x \). Add \( x \) to \( W \) and partition the states \( s_k \) in every \( B_j \) into smaller blocks \( B_j \) based on their different output \( \lambda(s_k, x) \). Repeat the process until each \( B_i \) becomes a singleton and \( r \) becomes \( n \). For every pair of states \( s_i \) and \( s_j \), the resulting \( W \) set includes an input sequence \( x \) that separates \( s_i \) from \( s_j \). Note that there are no more than \( n - 1 \) partition, and therefore \( W \) set has no more than \( n - 1 \) separating sequences. The \( W \) method consists in using the entire \( W \) set instead of the status message to test that the end state of each transition is the expected one. We have to care about \( W \) set, because it may contain many sequences, we have to visit for several times the same end state of every transition to apply all the separating sequences in a \( W \) and for this goal we can use a reset message and the sequences in a \( P \) set. The set of input is simply obtained concatenating every input in the \( P \) set with every input of \( W \) then apply them in order after a reset message to take the machine back to the initial state. In this way, each input sequence \( p_i \) is the concatenation of the \( i \)th sequence of a \( P \) set(to test the \( i \)th transition) with the \( j \)th sequence of a \( W \) set, with an initial reset input. For example and formally if we have two sets of input \( X \) and \( Y \), so we can denote with \( X.Y \) the set of input sequences that generated by concatenating all the input of \( X \) with all the input of \( Y \). The input sequences produced by the \( W \) method is equal to \{reset\}.\( P.W \). If we do not observe any fault, the implementation is proved to be correct [3][14]. By applying the sequence of \( P \) we can simply detect faulty-output, while any transfer fault is detected by the application of \( W \).

**Example[14]:** the characterizing set \( W \) of the machine in Fig.1 is \{a,b\}: For state \( s_1 \), transitions a/0 b/1, For state \( s_2 \), transitions a/1 b/1, For state \( s_3 \), transitions a/0 b/0. A distinguishes \( s_1 \) from \( s_2 \) and \( s_3 \) from \( s_2 \), b distinguishes \( s_1 \) from \( s_3 \). \( P = \{\varepsilon, a, b, ba, bb, bba, bbb\} \) the set of test sequences \( P.W \) is reported in the following table, where we indicate with \( r \) the reset message.[14]

<table>
<thead>
<tr>
<th>P</th>
<th>a</th>
<th>b</th>
<th>ba</th>
<th>bb</th>
<th>bba</th>
<th>bbb</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Test sequence P.W indicate with \( r \) the reset message.
and \( s_j \) gave the same output for each sequence in a \( W \). Machine \( M_1 \) is similar to \( M_5 \) if for every state \( s_i \) of \( M_5 \), the machine \( M_1 \) has similar state as \( s_i \), here we have \( n \) states (Assumption 6) then there exists a one-to-one correspondence between similar states of the two machines \( M_5 \) and \( M_1 \). If the fault is uncover by using the input within the first phase, we can pact that every state in \( M_5 \) has a similar state in the implementation, and for sure \( M_1 \) and \( M_5 \) are similar. But this is not enough to confirm that they are equivalent. The proof of the equivalence will be withen phase2.

**Phase 2:** The second phase tests all the transitions. To this aim, Wp method uses the identification sets. For every transition from state \( s_j \) to state \( s_i \) on input \( a \), we apply a sequence \( x \) (after a reset) which let the machine go to the state \( s_i \) along transitions already verified, then we apply the input \( a \), which let the machine go to \( s_j \), and then apply one identification sequence of \( W_1 \). Finally, repeat this test for every identification sequence in \( W_1 \), and if these tests do not uncover any fault, we have verified that the transition in the machine \( M_1 \) from a state that is similar to \( s_j \), on input \( a \) produces the right output (there is no faulty-output) and moves to a similar state to \( s_i \) (under no transfer fault). By applying these tests to every transition, we can prove that \( M_1 \) conforms to its specification. The set of input that covers every transition (and that is closed under the operation of selecting a prefix) is a \( P \) set. Therefore, the input of phase2 contain the sequences of a \( P \) ending in state \( s_j \) that are not contained in the \( Q \) set used by phase 1, concatenated with all sequences contained in the identification set \( W_1 \). Formally if \( R = P - Q \) and \( x_1 \) in \( R \) ends in \( s_k \), the applied sequences during the second phase is \{reset\}. \( R, W_1 \). A complete formal proof of correctness for the Wp method is given in the paper[4].

**Example.[15]** Fig. 1 machine has state cover set \( Q = \{c, b, bb\} \). During the first phase we generate the following test sequences:

<table>
<thead>
<tr>
<th>state to test</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q )</td>
<td>c</td>
<td>b</td>
<td>bb</td>
</tr>
<tr>
<td>( r.Q.W )</td>
<td>ra</td>
<td>rb</td>
<td>rba</td>
</tr>
<tr>
<td>output</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

During the second phase, we first compute the identification sets. \( W_1 = \{a, b\} \) all the sequences in \( W \) are needed to identify \( s_1 \), \( W_2 = \{a\} \) distinguishes the state \( s_2 \) from all other states, \( W_3 = \{b\} \) distinguishes the state \( s_3 \) from all other states, \( R = \{a, b, bb, bba, bbba\} \)
plying a UIO sequence, we can distinguish the first state in the sequence instead of another state. The original UIO method [5] builds a UIO sequence for each state of the FSM, and reaches that state by applying a UIO sequence. If a UIO sequence exists for every state of the FSM, then UIOs can be used to identify each state in the FSM. In this way, a UIO sequence can determine the state of the machine before its application. A UIO sequence has the opposite role with the machine; in this case the UIO sequence acts as a status message, except it moves the machine to another state.

### 5.3. UIO Methods

If a $W_i$ set contains only one sequence, this sequence is called state signature [2] or unique input/output (UIO) sequence [5], that is unique for the state $s_i$. By applying a UIO sequence, we can distinguish state $s_i$ from any other state, because the output produced applying a UIO sequence is specific to $s_i$. In this way, a UIO sequence can determine the state of the machine before its application. A UIO sequence has the opposite role of a homing sequence or a synchronizing sequence; it identifies the first state in the sequence instead of the last one. Note that not every state of the FSM has UIOs. If a UIO sequence exists for every state $s_i$, then UIOs can be used to identify each state in the machine; in this case the UIO sequence acts as a status message, except it moves the machine to another state. The original UIO method [5] builds a set of input sequences that visit every state from $s_i$ to $s_j$ by applying a transition cover set $P$ and then check the end state $s_j$ by applying its UIO sequence. In this case, UIO is used instead of a status message. Although used in practice, the UIO method does not guarantee to detect every fault in the implementation [6] because the uniqueness of the UIO sequences may not hold in a faulty implementation which may contain a state $s'$ with same UIO as another state $s$, and a faulty transition ending in $s'$ instead of $s$ may be tested as correct. Note that, for this reason, the Wp method uses the $W_i$ sets only in the second phase, while, in phase 1 it applies the complete $W$ instead. A modified version of the UIO method, called UIOv, generates correct checking sequences [6]. The UIOv method builds the test suite in three phases:

1. **$U_v$ process:** For every state $s$ in $M_3$ applying an input sequence $x$ that begins with a reset and reaches $s$ and then apply the UIO sequence of $s$. To reach each state use a $Q$ set. The set of input sequences consist of

   (2) **$\lnot U_v$ process:** visit every state $s$ and applies the input part of the UIO sequences of all other states and check that the obtained output differs from the output part of the UIO sequence applied. Skip UIO sequences that have the input part equal to a prefix $\alpha$ of the input part of the UIO sequence of $s$. Indeed, in this case, we have already applied $\alpha$ during the $U_v$ process, and we know that the output differs because two states cannot have the same input and output part of their UIO sequences. At the end of $U_v$ and $\lnot U_v$ process we have verified that $M_3$ is similar to $M_3$.

3. **Transition test phase:** check that every transition not already verified in 1 and 2 produces the right output and ends in the right state by applying its UIO sequence [15].

Note that the UIOv method can be considered as a special case of Wp method, where the $W$ set is the union of all the UIO sequences and phase 1 of the Wp method includes both $U_v$ process and $\lnot U_v$ process and phase 2 is the transition test phase.

#### Example

For the machine in Fig. 1 the UIO sequences are: $UIO_1 = ab/01$ distinguishes the state $s_1$ from all other states, $UIO_2 = a/1$ distinguishes the state $s_2$ from all other states, $UIO_3 = b/0$ distinguishes the state $s_3$ from all other states.

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$c$</th>
<th>$b$</th>
<th>$bb$</th>
</tr>
</thead>
<tbody>
<tr>
<td>state to test</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>r.Q.UIO</td>
<td>rab</td>
<td>rba</td>
<td>rbb</td>
</tr>
</tbody>
</table>

| output | 01  | 11  | 110 |

<table>
<thead>
<tr>
<th>state to test</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.Q.¬UIO</td>
<td>rb</td>
<td>rbab</td>
<td>rbb</td>
</tr>
</tbody>
</table>

| output | 1   | 11  | 110 |

#### Transition test phase:

<table>
<thead>
<tr>
<th>transition to test</th>
<th>a/0</th>
<th>a/1</th>
<th>b/0</th>
<th>a/0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S1$</td>
<td>$S2$</td>
<td>$S3$</td>
<td>$S3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input sequence</th>
<th>raab</th>
<th>rbaa</th>
<th>rbbab</th>
<th>rbbab</th>
</tr>
</thead>
</table>

| output | 001 | 111 | 1100 | 1100 |

The output fault of machine $M_3$ of Fig. 2 is detected during the $U_v$ process again by the input sequence $rbb$. Some transfer faults in machine $M_3$ are detected during the first phases, while others, like the transfer fault from state $s_3$ with input $a$ is detected only by the input sequences $rbbab$ during the transition test phase.
5.6. Distinguishing Sequence Method

In case we can find one sequence that can be used as UIO sequence for every state, we call such sequence distinguishing sequence (DS). In this situation, we can apply the DS method using the reset message [7]. Note that this DS method can be viewed as a particular case of the W method when the characterizing set of states of M contains only a preset distinguishing sequence. The test sequences are obtained simply by combining a P set with x.

**Example:** For the machine in Fig. 1 we can take the sequence x = ab as a preset distinguishing sequence. In fact $d_M(s_1, x) = 01$, $d_M(s_2, x) = 11$, and $d_M(s_3, x) = 00$.

<table>
<thead>
<tr>
<th>P</th>
<th>ε</th>
<th>a</th>
<th>b</th>
<th>ha</th>
<th>bb</th>
<th>hba</th>
<th>rbbba</th>
</tr>
</thead>
<tbody>
<tr>
<td>r.P.</td>
<td>ra</td>
<td>raa</td>
<td>rbba</td>
<td>rbbaa</td>
<td>rbbbaa</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| x | b | b | b | b |

**Cost and Length:** All the methods presented in Section 3 share the same considerations about the length of the checking sequence and the cost of producing it. For the W method, the cost to compute a W set is $O(pn^2)$ and a W set contains no more than $n - 1$ sequences of length no more than $n$. The cost to build a tree $T$ using the Algorithm 10 is $O(pn)$, and its maximum level is $n$. The generation of a $P$ set, by visiting $T$, takes time $O(pn^2)$ and produces up to $pn$ sequences with a maximum length $n$. Since we have to concatenate each transition from a $P$ set with each transition in a $W$ set, we obtain up to $pn^2$ sequences of length $n + n$, for a total length of $O(pn^3)$ and a total cost of $O(pn^3)$. The Wp method has the same total cost $O(pn^3)$ and same length $O(pn^3)$. Experimental results [4] show that checking sequences produced by the Wp method are generally shorter than the checking sequences produced by the W method. The UIO method and the method using a preset distinguishing sequence are more expensive, because determining if a state has UIO sequences or a preset distinguishing sequence was proved to be PSPACE hard[8]. There are specification machines with a reset message that require checking sequences of length $O(pn^3)$ [9].

6. Using Identifying Sequences Instead of Distinguishing Sequences

Not every finite state machine has distinguishing sequences. In case the machine has no reset message, no status message, no UIO sequences, and no distinguishing sequences, we cannot apply the methods proposed so far. We still can use Assumption 1 and utilize the existence of separating sequences which can distinguishing the states. So conformance testing is still possible [10], and the resulting checking sequences could be exponentially long.

As usual, we first check that $M_1$ is similar to $M_2$. We display for each state $s_i$ the responses to all the separating sequences in a characterizing set $W$ (Definition 2). Suppose that $W$ has two separating sequences $w_1$ and $w_2$. We want to apply the steps shown (in square boxes) in Fig. 5 (a): take $M_1$ to $s_i$, apply $w_1$ (step 1), take the machine back again to $s_i$ (step 2) and then apply $w_2$ (step 3). If we observe the right output, we can say that the machine $M_1$ has a state $q_i$ similar to $s_i$. We can start from $i = 1$ and proceed to verify all the states without using neither reset nor a distinguishing sequence. Our problem here presented because we do not know how to bring $M_1$ back to $s_i$ in a verifiable way, because in a faulty machine, as shown in Fig. 5 (b), transfer sequence $τ(t_i, s_1)$ (step 2) can make the machine moves to another state $s_i'$ where we can look for the expected output applying the $w_2$ sequence, without needing for verifying that $s_i'$ is $s_i$ and without able to apply again $w_1$. We use now the Assumption 6, namely that $M_1$ with only $n$ states. Let $x$ be the input sequence and $n$ is an integer, $X^n$ is the concatenation $n$ times of $x$.

**Theorem 1.** Let $s$ be a state of $M_1$, $x$ is an input sequence, $o$ the expected output sequence produced applying $x$ to $s$, i.e. $o = δ(s, x)$, $τ$ a transfer sequence from $t = δ(s, x)$ back to $s$, and $o'$ the expected output produced applying $τ$ to $o$. By applying the input sequence $(x τ)^n$ to state $s$ in $M_1$, if we observe the output sequence $(o o')^n$, then the machine ends in a state where applying again $x$ we observe the same output $o$.

**Proof.** The scenario of the theorem is shown in Fig. 6. Suppose that $M_1$ is initially in state $s$. Applying $x$ the machine should come back to $s$. However, due to some faults, the machine $M_1$ may go to another state $q_i$ even if the output we observe is the one expected, i.e. $o o'$. Assume that applying $n$ times $x τ$ , we observe every time the same output $o o'$. Let $qr$ be the state of $M_1$ after the application of $(x τ)^n$. 

![Figure 5. Using two separating sequences to identify the state](image-url)
Note that even if the \( n \) applications of the \( \tau \) produce \( n \) times the same correct output \( o \), we are not sure that \( s, q_1, \ldots, q_n \) are the same state yet. However, the \( n+1 \) states \( s, q_1, \ldots, q_n \) cannot be all distinct, because \( M_f \) has \( n \) states. Hence \( q_n \) is equal to some \( qr \) with \( r < n \) and, therefore, it would produce the same output \( o \) if we apply \( x \) [8].

**Example:** Consider the machine in Fig. 1 and take any alleged implementation \( M_f \). Apply the input \( a \) (in this case \( \tau = e \)) to the initial state \( s_1 \) of \( M_f \) and check that the output is 0. We are not sure that \( M_f \) is now in state \( s_1 \) as well. We can apply again \( a \) and observe the output 0 and so on. When we have applied \( aaa \) and observed the output 000, \( M_f \) may have traversed states \( s_1, q_1, q_2, \) and the final state \( q_3 \). Our machine \( M_f \) has only three states, so for sure \( q_3 \) is equal to one of \( s, q_1, \) or \( q_2 \) and certainly if we apply \( a \) we should observe 0.

We use Theorem 1 as follows. Assume that \( M_f \) has the characterizing set \( W = \{ w_1, w_2 \} \) and let \( s \) be the state we are going to verify. Let \( \tau \) be the transfer sequence that takes \( M_f \) back to \( s \), from \( t_i = \delta(s_i, w_i) \). We first apply \( (w_1 \tau)^m \) to \( s \). If we observe a wrong output we have proved that \( M_f \) does not conform to \( M_f \). Otherwise we can apply theorem with \( x = w_1 \) and we are sure that \( M_f \) ends in a state that would produce the same output as if we applied \( w_1 \). We apply \( w_2 \) instead. If we observe the specified output we can conclude that \( s \) has a similar state in \( M_f \) and we can generalize this method when the characterizing set \( W \) contains \( m \) separating sequences. Suppose that the characterizing set is \( W = \{ w_1, \ldots, w_m \} \). Let \( \tau \) be the transfer sequence that takes the machine back to \( s \) after the application of \( w_j \), i.e. \( \tau = \pi(\delta(s, w_j), s) \). We can define inductively the sequences \( \beta \tau \) as follows:

\[
\beta_1 = w_1 \\
\beta_\tau = (\beta \tau - 1 \tau - 1)^{\tau wr} 
\]

By induction, one can prove that applying \( \beta \tau - 1 \) after applying \( (\beta \tau - 1 \tau - 1)^{\tau} \) would produce the same output. Considering how \( \beta \tau \) are defined, this means that applying \( w_1, \ldots, w_j, l \) would produce the same output. For this reason we apply \( w_1 \) after \( (\beta \tau - 1 \tau - 1)^{\tau} \). Therefore, one can prove that \( \beta \tau \) is an identifying sequence of \( s \), in the following sense: if the implementation machine \( M_f \) applying \( \beta \tau \) produces the same output as that produced by the specification machine starting from \( s \), then \( M_f \) has a state that is similar to \( s \) and such state is the state right before the application of the last \( w_m \) (regardless of which state \( M_f \) started from). We indicate the identifying sequence for state \( s \) with \( \beta \tau \). Since we have computed the identifying sequence for every state, we can apply a method similar to that explained in former section to visit each state, to confirm the response of the machine to the identifying sequence then moving to the next state. Let \( \beta_0 \) denote the identifying sequence of state \( s \) and \( \tau(t_1, s_{n+1}) \) be the transfer sequence from \( t_1 = \delta(s_1, \beta_1) \) to \( s_{n+1} \), by applying the following sequence we can emphasize the similarity between \( M_f \) and \( M_f \).

Once we have proved that \( M_f \) is similar to \( M_f \) we have to verify the transitions. To do this, we can use any \( \beta_0 \) as reliable reset. For example, we can take \( \beta_0 \) as a reset to the state \( t_1 = \delta I \) and use \( \beta_0 \) as the initial state to test every transition. Indeed, we are sure that if we do not observe any fault, \( \beta_0 \) takes the machine to \( t_1 \). If we want to reset the machine from the state \( s_k \) to \( t_1 \) we apply \( \tau(s_k, s_1) \beta_0 \) and even if \( \tau(s_k, s_1) \beta_0 \) fails to take the machine to \( s_1 \), we are sure that \( \beta_0 \) will take it to \( t_1 \). Now we proceed as explained in Section VI. To test a transition between \( s_i \) and \( s_j \) we can apply a reset \( \beta_0 \) to \( t_1 \), then a transfer sequence to \( s_1 \), then we apply the input, and get the output, finally apply the identifying sequence \( \beta_0 \) to make sure of the end state is \( s_i \).

**Example.** Consider the machine \( M_f \) in Fig. 1. \( W = \{ a, b \} \). For \( s_1, \beta_0 = e, I = (w_1 \tau)^3 w_2 = aaa b, \) For \( s_1, \beta_0 = e, I = (w_1 \tau)^3 w_2 = aaa b \) and For \( s_1, \beta_0 = e, I = (w_1 \tau)^3 w_2 = aaa b \). The sequence (2) becomes:

\[
|I| \quad |\tau(t_1, s_1) \beta_0| \quad |\tau(t_2, s_2) \beta_0| \quad |\tau(t_3, s_3) \beta_0| \\
\text{input sequence} \quad \text{aaa} \quad b \quad e \quad \text{aaa} \quad b \quad e \quad \text{aaa} \quad b \quad e \\
\text{Length and Cost: Identifying sequence lengths grows exponentially with the number of separating sequences and with } n \text{ the number of the states. Indeed, by equation 2, every } \beta_i \text{ is } n \text{ times longer than } \beta_i - 1, \text{ the identifying sequence } I \text{ is equal to } \beta m, \text{ the number of separating sequences is represented by } m, \text{ and that can be up to } n. \text{ The resulting checking sequence is exponentially long. We can optimize the IS method by using a different separating family } Z_i \text{ for every state } s_i. [2][14].

**RESULTS**

The W, Wp, UIOv, and DS methods detect faults of any kind, while the UIO method may miss some faults.

Methods W, Wp, and DS with an adaptive distinguishing sequence produce minimum coast while the others have a greater cost.
If the status message is not available (which is mostly occurs in software black box testing), then we should use some extra inputs to verify the states and that inputs should be unique, like in Wp, UIO and DS.

Table2. summery for choosing a best method

<table>
<thead>
<tr>
<th>Absent message</th>
<th>Present message</th>
<th>Method to use</th>
<th>checking sequence</th>
<th>Length and coast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status message</td>
<td>Reset message</td>
<td>W, Wp, UIO (Unique Input Output), DS (Distinguishin g sequence)</td>
<td>Length O(pn^3) Coast O(pn^3) (the minimum Cost)</td>
<td></td>
</tr>
<tr>
<td>Reset message</td>
<td>Distinguishing sequence</td>
<td>DS method uses transfer sequence instead of reset</td>
<td>Length O(pn^3) Coast O(pn^3) when used in conjunction with adaptive distinguishing sequence</td>
<td></td>
</tr>
<tr>
<td>Not even Distinguishing sequence</td>
<td>Identifying sequence IS (not covered)</td>
<td>It produces exponentially long checking sequence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nor UIOs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Future work

The main weakness of some methods is that it needs the set message, which may not be available. To avoid the use of a set and to possibly shorten the test suite, we can generate a sequence that traverses the machine and throw every state and every transition at least once without needing to restart and get back to the initial state after each test and without using a set message. Such sequence is called transition tour[13] which can be used and compared with our explained methods.

Conclusions

Two machines M_s and M_i are equivalent if they will produce the same output for every input sequence, we looking for a checking sequence that prove that the two machines are equal. There are many methods for generating a checking sequence and some of them require only that all states are reachable from the initial one, permit the machines with deadlocks or states without any exiting transition. Anywise, these methods must require a reset message that take back the machine to its initial state, furthermore a deadlock may stop the test experiment. Our methods depend on the present or absent of set, reset, and status messages s,[2][13].

References

doi: 10.1109/5.533956

doi: 10.1109/12.272431

doi: 10.1109/TSE.1978.231496

doi: 10.1109/32.87284


doi: 10.1109/32.16602


doi: 10.1147/rd.104.0278