Formal methods for test sequence generation

Hasan Ural reviews formal methods for test sequence generation from FSM-based specifications

Six formal methods and their enhancements for generating test sequences from FSM-based specifications are reviewed. These methods are: the Transition Tour (T) method; the Distinguishing Sequence (D) method; the Characterizing Set (W) method; the Unique Input/Output Sequence (UIO) method; the Single UIO (SUIO) method; and the Multiple UIO (MUIO) method. Improved variations of the D-, W-, UIO- and MUIO-methods are included in the discussions. These formal methods are compared in terms of the upper bounds on the length of resulting test sequences. A tool implementing these methods and their enhancements is presented.

Keywords: formal methods, test sequence generation, finite state machines, protocol testing

The formal methods reviewed in this paper have been advocated for the generation of test sequences to facilitate an efficient and effective solution to the following problem: given a FSM (finite state machine[1]) representation of a system and an implementation of this FSM, determine whether the implementation conforms to (behaves in accordance to) the specification. Clearly, many systems such as telecommunication systems can be specified by the FSM model. For example, the control portion of a communications protocol can be represented as an FSM. Based on the FSM model of a protocol, formal methods can be employed to generate test sequences for testing the control portion of a protocol implementation under test. Some of these methods have been reviewed elsewhere[2-5].

Department of Computer Science, University of Ottawa, Ottawa, Ontario K1N 9B4 Canada
Paper received: 5 March 1991

In this paper, a comprehensive collection of formal methods and their enhancements for generating test sequences from FSM-based specifications is reviewed. These methods are: the Transition Tour (T) method[6-8]; the Distinguishing Sequence (D) method[3,7,9]; the Characterizing Set (W) method[3,7,10]; the Unique Input/Output Sequence (UIO) method[11,12]; the Single UIO (SUIO) method[13]; and the Multiple UIO (MUIO) method[14]. In this review, improved variations of the D-, W-, UIO- and MUIO-methods are also included. Some efficient algorithms for solving the Chinese Postman Problem (CPP)[15] and the Rural Chinese Postman Problem (RCPP)[13] are employed in the T-method and the SUIO- as well as MUIO-methods, respectively. The paper also shows that the solutions of the RCPP can be applied to the D-method and the W-method to construct minimum-length test sequences.

The rest of the paper is organized as follows: first, some related definitions are given. Then, six formal test sequence generation methods and their enhancements are reviewed. The features of a tool that incorporates the implementations of these methods and their enhancements are presented. Finally, the formal methods are compared in terms of the upper bounds on the length of resulting test sequences.

PRELIMINARIES

The formal methods that are reviewed in this paper are based on a deterministic FSM model[1] which can be characterized by a quintuple $M = (S, I, O, \delta, \lambda)$, where:

S : is a finite set of states $\{s_1, s_2, \ldots, s_n\}$;
I : is a finite set of inputs $\{i_1, i_2, \ldots, i_q\}$;
O : is a finite set of outputs $\{o_1, o_2, \ldots, o_r\}$;
δ : is the *transfer function*, $S \times I \rightarrow S$;
λ : is the *output function*, $S \times I \rightarrow O$.

0140-3664/92/005311-15 © 1992 Butterworth-Heinemann Ltd

In this model $s_s \in S$, 'r' $\in I$, and '$-$' $\in O$ denote the *initial* or *start state*, the *reset input*, and the *null output*, respectively. Moreover, δ and λ are defined as partial functions. This reflects the fact that a given FSM may be *incompletely specified*, i.e. there may not be a state transition defined for each input in I at each state in S. For an incompletely specified FSM, a *completeness assumption* is adopted to implicitly specify the response of FSM to each *unexpected input* (i.e. an input i is an unexpected input at state $s \in S$ if either $i \notin I$ or $i \in I$ but no transition defined for i at s).

An FSM M can be represented by a directed graph $G = (V,E)$, where a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ represents the set S of states of M, and a set of directed edges $E = \{(v_j, v_k; i/o): v_j, v_k \in V\}$ represents the transitions of M, i.e. each edge $(v_j, v_k; i/o) \in E$ represents a state transition (denoted henceforth by e_{jk} or t_{jk}) from v_j to v_k with input i and output o. For ease of reference, v_j, v_k, i/o are called the *head state*, the *tail state*, and the *label* of edge $(v_j, v_k; i/o) \in E$, and denoted by head(t_{jk}), tail(t_{jk}), and label(t_{jk}), respectively.

Throughout this paper, a given deterministic FSM M is assumed to be *minimal* (i.e. none of its states are equivalent[16] and *strongly connected* (i.e. there exists a path, a finite sequence of adjacent but not necessarily distinct edges, between any pair of vertices v_j and $v_k \in V$). Moreover, for an incompletely specified M, the following completeness assumption is adopted: M ignores each unexpected input by producing a null output and remaining in its current state. These state transitions implied by the completeness assumption are called *non-core edges* of M. Consequently, the state transitions defined by the transfer and output functions of M are called *core edges* of M (i.e. the edges of the graph G representing M). In subsequent sections, an FSM M and its graph representation G; the states of M and the vertices of G; the transitions of M and the edges of G are used interchangeably as long as it does not cause confusion.

Testing an FSM implementation

The formal methods reviewed in this paper have been advocated for the generation of test sequences to facilitate an effecient and effective solution to the following problem: given an FSM representation of a system (denoted henceforth as FSM_s) and an implementation of this FSM (denoted henceforth as FSM_i), determine whether FSM_i conforms to (behaves in accordance to) FSM_s by testing FSM_i as a black-box. Clearly, the black-box testing of the conformance of FSM_i to FSM_s:

(a) does not assume the availability of implementation details of FSM_i (such as the number of states and transitions of FSM_i);
(b) requires
  (1) generation of sequences S_i and S_e of inputs and expected outputs from FSM_s;
  (2) application of S_i to the input port of FSM_i;
  (3) observation of a sequence S_a of actual outputs at the output port of FSM_i;
  (4) comparison of S_a with S_e to determine the conformance of FSM_i to FSM_s.

As a consequence of black-box testing, the conformance of FSM_i to FSM_s can only be determined through the analysis of the observed responses of FSM_i. In the most general case, an FSM_i may differ from FSM_s in:

(a) the number of states (i.e. FSM_i may have missing or extra states);
(b) the number of transitions (i.e. FSM_i may have missing or extra transitions);
(c) the implementation of the completeness assumption;
(d) the implementation of the output and the transfer functions.

Obviously, sequences S_i and S_e derived from FSM_s cannot, in general, be expected to detect extra states or transitions in FSM_i. However, these sequences are expected to reveal all other instances where FSM_i differs from FSM_s, since such instances can be identified through the analysis of the observed responses of FSM_i. In fact, characterizing sequences (also called α-sequences[9]) can be used to identify missing states in FSM_i. Moreover, missing transitions as well as incorrect implementations of the completeness assumption, the output function, and the transfer function in FSM_i will exhibit themselves as instances of either output errors or transfer errors (which are called *faults*).

The conformance of an FSM_i to a given FSM_s is in general defined at two levels: weak or strong conformance[12]. An FSM_i has *weak conformance* to FSM_s if FSM_i correctly implements the output and the transfer functions specified in FSM_s (i.e. FSM_i has the same intput/output behaviour as the FSM_s with respect to the core edges only). An FSM_i has *strong conformance* to FSM_s if FSM_i correctly implements the output and the transfer functions specified in FSM_s, as well as the completeness assumption provided with (possibly an incompletely specified) FSM_s (i.e. FSM_i has the same intput/output behaviour as the FSM_s with respect to both the core edges and the non-core edges).

To determine the weak or strong conformance of an FSM_i to FSM_s, a transition-level approach originating from sequential circuit checking experiments[16] is adopted. In this approach, the weak (or strong) conformance of an FSM_i to FSM_s is determined by checking whether every transition corresponding to core (or core as well as non-core) edges of FSM_s is correctly implemented by FSM_i.

Accordingly, the procedure for checking whether a transition $t_{jk} = (s_j, s_k; i/o)$ of FSM_s is correctly implemented by FSM_i consists of three steps:

(1) FSM_i is brought to state s_j;
(2) input i is applied to FSM_i and the output produced by FSM_i is checked for whether it is o;
(3) the state FSM_i reaches after the application of input i is checked for whether it is s_k.

Steps (2) and (3) are used to detect the errors in the output function (i.e. output errors) and the transfer function (i.e. transfer, transition or tail state errors),

respectively. However, in certain cases, i.e. in protocol conformance testing[17, 18], this procedure is difficult to realize due to the limited controllability and observability of FSM_i. In this context, the limited controllability refers to the restrictions that require a transfer sequence (i.e. the shortest sequence of inputs to bring FSM_i to a designated state, e.g. s_j in Step 1), and the limited observability refers to the restrictions that require a characterizing sequence (i.e. the shortest sequence of inputs to verify that FSM_i reached a designated state, e.g., s_k in Step 3). Thus, the input sequence (i.e. a *test subsequence*) required to test $t_{jk} = (s_j, s_k; i/o)$, consists of:

(a) an optional transfer sequence to bring FSM_i from its current state to state s_j;
(b) input i for stimulating transition t_{jk};
(c) a characterizing sequence to verify that FSM_i reached state s_k.

Accordingly, the sequence of inputs (possibly interleaved with the corresponding expected outputs) that takes an FSM from its start state, contains a test subsequence for each transition of the FSM, and returns the FSM to its start state is called a *test sequence* (TS).

In the next section, formal methods for test sequence generation and their enhancements will be reviewed. In this review, the presentation of each method or enhancement will be followed by its application to the example FSM_s given in Figure 1. This FSM_s is minimal, strongly connected, and completely specified. Moreover, since some methods require that the given FSM_s has reset capability, it is assumed that FSM_s in Figure 1 has reset capability (i.e. a reset input brings FSM_s from any state to the start state with a single transition, called henceforth reset transition or reset edge, producing a null output). With this assumption, the reset transitions are included in the definition of FSM_s as core edges. Since a weak or strong conformance test sequence is expected to contain a test subsequence for each core edge, the test sequence that is obtained by the application of each method to the example FSM_s given in Figure 1 will also include a test subsequence for each reset transition. Furthermore, for ease of presentation, only weak conformance test sequences will be considered. Thus, each formal method will be applied to the example FSM_s in Figure 1 for the purpose of generating a test sequence to determine the weak conformance of any given implementation of FSM_s.

FORMAL METHODS FOR TEST SEQUENCE GENERATION

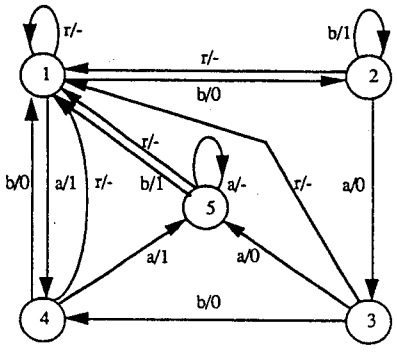
In this section, formal methods (i.e. T-method, D-method, W-method, UIO-method, SUIO-method, and MUIO-method) for test sequence generation are reviewed. The features of a tool that implements these methods are also presented. In this review and presentation, any given FSM_s, and in particular the one shown in Figure 1, will be denoted by M.

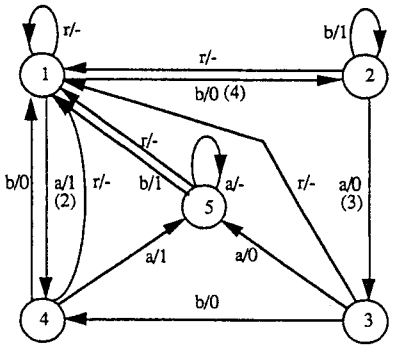
T-method

A *transition tour* (TT) of M is an input sequence which takes M from its start state s_s, executes every transition of M at least once, and then returns M to state s_s. This method of obtaining a test sequence is called the *T-method*[6]. The T-method is the simplest formal method, but it can only detect output errors, since the state that M reaches after the execution of a transition is not checked as specified in Step (3) of the transition checking procedure given in the previous section.

Previous approaches[6, 7] to generating a TT have been *ad hoc*, which causes redundancies and thus results in TTs that are not of minimum length. Recently, an efficient algorithm has been proposed for generating a minimum-length TT[8]. Before presenting this algorithm, the related terminology is given as follows:

A *tour* in G is a path in G which starts and ends at the same vertex of G. The *cost* of a tour is the sum of the costs of edges included in the tour. The cost of an edge is the number of i/o pairs in its label. From the definition of M, it follows that the cost of each edge in G representing a given M is one (i.e. unit cost). On the other hand, the *length* of a tour is the number of edges included in the tour. Thus, a minimum-cost tour or the minimum-cost test sequence is considered to be of minimum-length. A





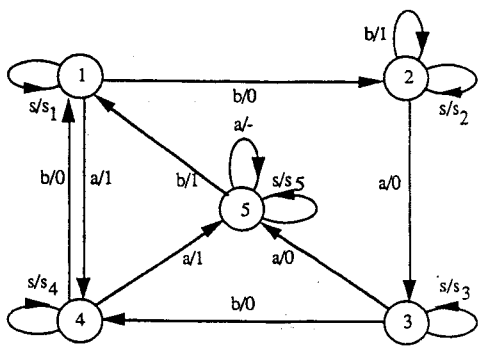


Figure 1. (left). Graph representation (G) of an FSM M. Figure 2. (centre). Symmetric augmentation G of G in Figure 1. Figure 3. (right). FSM in Figure 1 with* Status Message *(reset edges are not shown)*

Chinese Postman Tour (CPT) of G is a minimum-cost tour which contains every edge in E at least once. The *Chinese Postman Problem*[15] is the problem of finding a CPT of G.

Uyar and Dahbura showed that finding a minimum-length TT of M is equivalent to finding a CPT of G (representing M). They employed an efficient algorithm given by Edmonds and Johnson[19] that solves the Chinese Postman Problem to generate a minimum-length TT of M. This algorithm makes use of the fact that a CPT exists for any strongly connected digraph and a *Euler tour* of G (i.e. a tour of G that contains every edge in G exactly once), if one exists, is a CPT of G. An implementation of this algorithm is described below.

Given G, a strongly connected digraph (V,E) representing M, there are two cases which must be considered in finding a CPT of G which is a minimum-length TT of M. Before presenting these two cases, the related terminology is given as follows: the *indegree* and *outdegree* of a vertex v of G is defined as: $d_{in}^{E}(v) = |\{(w, v; i/o):(w, v; i/o) \in E\}|$ and $d_{out}^{E}(v) = |\{(v, w; i/o):(v, w; i/o) \in E\}|$. The *index* $\xi(v)$ of a vertex $v \in V$ is defined as $\xi(v) = d_{in}^{E}(v) - d_{out}^{E}(v)$. G is said to be *symmetric*, if for every $v \in V$, $d_{in}^{E}(v) = d_{out}^{E}(v)$, i.e. $\xi(v) = 0$.

Case A: if G is symmetric, then finding a CPT of G is equivalent to finding a Euler tour of G, since every strongly connected and symmetric digraph has a Euler tour[19]. In this case, a minimum-length TT of M which starts and terminates at s_s is found as follows:

(a) construct a spanning arborescence T of G (i.e. a *spanning arborescence* T is a spanning tree, in which each node $v \in T$ satisfies $d_{out}(v) = 1$ except the root s_s, i.e. $d_{out}(s_s) = 0$);
(b) construct a Euler tour of G such that the edges of T are traversed last in the tour, which ensures the traversal returns to the initial state s_s. The Euler tour of G is the CPT of G, which in turn is a minimum-length TT of M.

Case B: if G is not symmetric, then every edge of E is contained in a minimum-length tour (e.g. a CPT of G) at least once. This means that some edges of E may be contained more than once in the tour. In this case, a minimum-length TT of M which starts and terminates at s_s is found as follows:

(a) construct a *symmetric augmentation* of G (i.e. a symmetric and strongly connected digraph G^*) such that the number of replicated edges is minimized;
(b) construct a spanning arborescence T of G^*;
(c) construct a Euler tour of G^* such that the edges of T are traversed last in the tour, which ensures the traversal returns to the initial state s_s. The Euler tour of G^* is the CPT of G, which in turn is a minimum-length TT of M.

The key issue in constructing a G^* of G is to minimize the total number of additional copies of some edges in E, i.e. $Z = (\Sigma X_e : e \in E)$ subject to the constraints:

(1) $X_e \geqslant 0$, integer,
(2) $\Sigma(X_e : v = \text{head}(e)) - \Sigma(X_e : v = \text{tail}(e)) = \xi(v)$, $v \in V$.

This is a linear programming (LP) problem which can be solved by a min-cost max-flow algorithm[8].

The application of the T-method to M in Figure 1 consists of the following steps:

Step 1. Minimize $Z = \Sigma(X_e : e \in E)$ subject to the above conditions:

(1) $X_{12} + X_{14} - X_{41} - X_{51} - X_{21} - X_{31} - X_{41} - X_{51} = 4$
(2) $X_{21} + X_{23} - X_{12} = -1$
(3) $X_{31} + X_{34} + X_{35} - X_{23} = -2$
(4) $X_{41} + X_{41} + X_{45} - X_{14} - X_{34} = -1$
(5) $X_{51} + X_{51} - X_{35} - X_{45} = 0$

(reset edges are indicated by bold). The solution of the above LP problem is:

$X_{12} = 3$, $X_{23} = 2$, $X_{14} = 1$; all the rest of $X_{ij} = 0$; and $Z = 6$.

Step 2. Add three e_{12}, two e_{23}, and one e_{14} to G to obtain G^* of G, as shown in Figure 2 (the total number of edges e_{12}, e_{23}, e_{14} are given in parentheses in Figure 2);

Step 3. Find a Euler tour of G^*:
(a) Find a spanning arborescence T of G^*.
(b) Use T of G^* to find a Euler tour of G^*.

A Euler tour (of length 21) of G^* is given by the following state sequence:

1, 4, 1, 2, 3, 1, 2, 1, 2, 3, 5, 1, 1, 4, 5, 5, 1, 2, 2, 3, 4, 1

which corresponds to a minimum length TT of M:

a/1, r/-, b/0, a/0, r/-, b/0, r/-, b/0, a/0, a/0, r/-, r/-, a/1, a/1, a/-, b/1, b/0, b/1, a/0, b/0, b/0.

In certain implementations there is a desirable feature called *status message*. It is a single message that identifies each state s_i of M via a single transition which produces a unique output for s_i and causes M to remain at state s_i. For example, in the case of an FSM_i which allows the tester to access the state of the implementation, the status message would be 'read the state of the FSM_i'. In this case, Step (3) of the transition-level testing approach can be easily incorporated into the T-method by using the status message to verify the state after each transition is checked for the first time[8]. The observability problem, therefore, is solved in this special case. Note that the transition caused by the status message for each state is also checked as a core transition for the first time it appears in the test sequence.

As an example, a TT (of length 41) generated for the FSM in Figure 3 with *status message* 's' is listed below:

b/0, s/s_2, s/s_2, b/1, s/s_2, a/0, s/s_3, s/s_3, b/0, s/s_4, s/s_4, a/1, s/s_5, s/s_5, a/-, s/s_5, r/-, s/s_1, s/s_1, a/1, s/s_4, r/-, s/s_1, b/0, a/0, r/-, s/s_1, b/0, r/-, s/s_1, b/0, a/0, a/0, s/s_5, b/1, s/s_1, a/1, b/0, s/s_1, r/-, s/s_1.

For implementations which have no status message, other characterizing sequences such as Unique Input/Output (UIO) sequences, Distinguishing Sequences (DS)

and Characterizing Sets (W-set) can be used to realize Step (3) of the transition checking procedure. In the following sections, the implementations of the other formal methods for test sequence generation are described. The test sequences generated by these methods all have the capability of detecting output errors and transfer errors[3,13,14].

D-method

This method[9] is designed for an FSM which possesses a distinguishing sequence (DS). An input sequence is a DS of M if the output produced by M in response to the DS is unique for each state of M. The DS is used to check whether M reaches the expected state after the execution of each transition as specified in Step (3) of the transition checking procedure.

The main idea of the D-method proposed by Gonenc is to construct an i @ DS-diagram (where input $i \in I$ and '@' means concatenation of two strings), and then to find a minimal covering of the i @ DS-diagram, which constitutes a test sequence for M[9]. A *covering* is a partition of all the edges of a graph into simple paths (i.e. a path is *simple* if it does not contain any edge twice) which are disjoint (i.e. having no common edges). A *minimal covering* of G is a covering of G which involves the fewest possible simple paths[20].

The implementation of the D-method consists of the following steps:

(1) Compute a DS of minimum length by constructing a distinguishing-tree[16];
(2) Construct an i @ DS-diagram denoted by $G[E_c] = (V,E_c)$, where $E_c = \{(v_j, v_1; i \text{ @ DS}) : (v_j, v_k; i/o) \in E, i \in I, o \in O$, and $\text{tail(DS)} = v_1 \in V\}$, $(v_j, v_k; i/o)$ is the transition to be tested, and $|E| = |E_c|$;
(3) Find a minimal covering of $G[E_c]$ as follows: First, define three sets:

$$R = \{v \in V \mid d_{out}^{Ec}(v) = d_{in}^{Ec}(v)\}$$
$$F = \{v \in V \mid d_{out}^{Ec}(v) > d_{in}^{Ec}(v)\}$$
$$P = \{v \in V \mid d_{out}^{Ec}(v) < d_{in}^{Ec}(v)\}.$$

Then:

(a) if $R = V$ and the underlying undirected graph of $G[E_c]$ is connected, then $G[E_c]$ is symmetric and a Euler tour exists. This Euler tour is the minimal covering of $G[E_c]$ which can be found by using Steps (b) and (c) of the procedure given for case B of the T-method;
(b) otherwise, a minimal covering of $G[E_c]$ can be found by the following:
  (1) start from a state in F;
  (2) each time an edge is followed, erase it;
  (3) do not use an edge which is an isthmus (i.e. an edge whose deletion divides the graph into two components), if there exists any other edge which can be followed;
  (4) when it is not possible to go any further, start again from a state in F in such a way that eventually from each state $v_k \in F$ exactly λ_k starts will be made, where $\lambda_k = d_{out}^{Ec}(v_k) - d_{in}^{Ec}(v_k)$. To reach another restart $v_k \in F$, a 'jump', i.e. a transfer sequence (TR) from v_j to v_k must be used, where $v_j \in P$ represents the tail state of the current simple path. Let q be the sum of all the λ_ks. Since there are q paths, the total number of *TR*s needed is $(q - 1)$.

  Note that in Step (4), there is the problem of choosing the state from which a restart is to be made. Suppose that the minimal covering reaches state v_j, then always choose the restarting state, say v_k, which requires the shortest TR, and there is at least one unerased edge emanating from v_k.

As an example, consider the FSM M in Figure 1. A minimum-length DS for M is 'ab'. A $G[E_c]$ of M is given in Figure 4, where $R = \emptyset$, $F = \{2,3,4,5\}$, $P = \{1\}$, $q = \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = 10$. The unlabelled edges in Figure 4 correspond to r @ DS.

Hence, the minimal covering consists of 10 paths and nine TRs if the test sequence starts from state 2, 3, 4 or 5 in F. One of the possible minimal coverings is (state sequence only):

211411-241-21-31-31-31-41-51-51-51

where the hyphens represent the shortest TRs between paths. The input portion of a test sequence (of length 61) based on the minimal covering is:

b,a @ DS, *a* @ DS, *b* @ DS, *a* @ DS, *r* @ DS,
b,b @ DS, *b* @ DS, *b,r* @ DS, *b,a,a* @ DS, *b,a,b* @ DS,
b,a,r @ DS, *a,r* @ DS, *a,a,a* @ DS, *a,a,b* @ DS, *a,a,r* @ DS.

It should be noted that there are some drawbacks of this method. First, the application of the D-method is limited to very few real specifications which have distinguishing sequences. Second, the resulting TS may not start and end with the initial state of the FSM. An enhancement of the D-method that eliminates the last drawback is presented at the end of this section.

W-method

A set of input sequences which can distinguish the behaviour of every pair of states in an FSM is called a

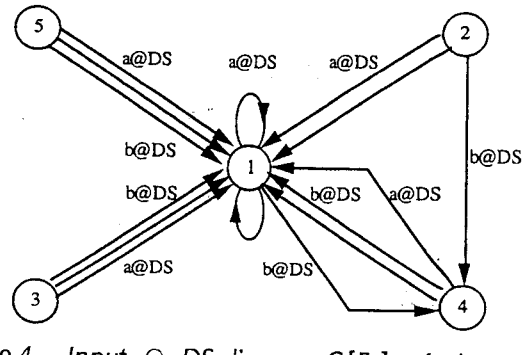


Figure 4. Input @ DS-diagram $G[E_c]$ of the FSM in Figure 1

characterizing set, and is denoted henceforth by *W-set*. In the W-method[10] a W-set is applied to check whether M reaches the expected state after the execution of each transition as specified in Step (3) of the transition checking procedure. Every FSM satisfying the assumptions listed in the previous section has a W-set, therefore, for FSMs which do not possess distinguishing sequences, the W-method is an alternative method for generating test sequences. However, this method yields longer test sequences compared with some other formal methods in most cases[3].

Given M, the implementation of the W-method consists of four steps:

(1) Compute minimal W-set $W = \{w_1, w_2, \ldots, w_k\}$ by applying a multiple-experiment on M^1.
(2) Construct a *Testing-Tree* T such that each branch of T is a transition in M, each node of T is a state of M, and each transition of M appears in T exactly once.
(3) Obtain the set $P = \{p_1, p_2, \ldots, p_m\}$ of all non-maximal partial paths in T that start at the root s_s of T. Each non-maximal partial path is a sequence of inputs with minimum length which takes M from its s_s to a particular state s_j, from which the next transition to be tested emanates. The empty sequence p_0 is also considered a member of P:
(4) Concatenate P and W to obtain a test sequence

$$(TS) = \bigcirc_{i=0}^{|P|} \bigcirc_{j=1}^{|W|} (p_i @ w_j @ r).$$

As an example, consider the FSM M in Figure 1:

(1) a W-set is found by applying a multiple-experiment on M^1: e.g. $W = \{a, aa, b\}$;
(2) build a Testing-Tree T;
(3) attach the W-set to each node of T as shown in Figure 5;
(4) the set of test subsequences based on the concatenations of sets P and W is:

W,b @ W, bb @ W, br @ W, ba @ W, bab @ W, bar @ W, baa @ W, r @ W, a @ W, ab @ W, ar @ W, aa @ W, aab @ W, aar @ W, aaa @ W

Thus, the input portion of a test sequence of length 211 is:

ar aar br bar baar bbr bbar bbaar bbbr brar braar brbr baar baaar babr babar babaar babbr barar baraar barbr baaar baaaar baabr rar raar rbr aar aaar abr abar abaar abbr arar araar arbr aaar aaaar aabr aabar aabaar aabbr aarar aaraar aarbr aaaar aaaaar aaabr.

UIO-method

As pointed out earlier, a DS or a W-set when applied to an FSM results in a sequence of outputs which uniquely identifies the state of the FSM before the application of the DS or the W-set. However, as suggested by Sabnani and Dahbura[11,12]; the identification of the actual state

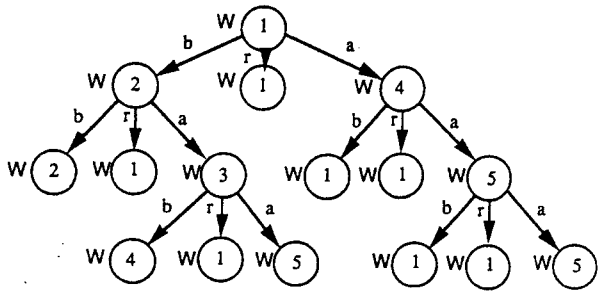


Figure 5. Testing-tree T of the FSM in Figure 1

reached after the execution of a transition is not necessary. Indeed, it is sufficient to determine whether the expected state is reached or not. Based on this observation, the use of UIO sequences is proposed[11,12] to check whether the FSM reaches the expected state after the execution of each transition as specified in Step (3) of the transition checking procedure.

An input/output behaviour that is unique for state s_j (i.e. not exhibited by any other state) of a given FSM is called a *UIO sequence* for s_j, and denoted by UIO(s_j) or UIO_j. Here, by a UIO sequence of a state, we mean either a Unique Input/Output Sequence or a Unique Signature for that state[11,12].

Given a digraph $G = (V,E)$ representing an FSM M, the implementation of the UIO-method consists of the following four steps:

Step 1. Compute a minimum length UIO sequence for each state of M using the procedures in Sabnani and Dahbura[12]. A minimum-length UIO sequence is a UIO sequence that does not contain any other UIO sequence.
Step 2. Construct a test subsequence for each transition $t_{jk} = (s_j, s_k; i/o)$ of M by concatenating
(a) a transfer sequence from the initial state to state s_j;
(b) input i for stimulating transition t_{jk};
(c) the UIO sequence for state s_k.
(The concatenation of the last two sections of a test subsequence, i.e. i @ UIO, will be called a *test segment*).
Step 3. Eliminate each test subsequence that is completely contained in another test subsequence.
Step 4. Construct the test sequence by connecting remaining subsequences using reset inputs.

As an example, consider the FSM M in Figure 1. UIO sequences for the states of M are given in Table 1a. A test sequence of length 63 obtained by this method is:

r/-, a/1, a/1, r/-, b/0, b/1	r/-, a/1, a/1, a/-, a/-
r/-, a/1, b/0, b/0, b/1	r/-, b/0, a/0, r/-, b/0, b/1
r/-, b/0, r/-, b/0, b/1	r/-, b/0, b/1, b/1, b/1
r/-, a/1, a/1, b/1, b/0, b/1	r/-, a/1; r/-, b/0, b/1
r/-, b/0, a/0, b/0, a/1, b/1	r/-, b/0, a/0, a/0, a/-
r/-, r/-, b/0, b/1	r/-, b/0, a/0, a/0, b/1

(Test subsequences for t_{12}, t_{14}, and t_{45} are completely contained in other test subsequences.)

Table 1a. UIO sequences for the FSM in Figure 1

States	*UIOs*	*Tails*
1	b/0 b/1	2
2	b/1 b/1	2
3	a/0 b/1	1
4	a/1 b/1	1
5	a/-	5

Table 1b. Transition @ UIO-table for the FSM in Figure 1

Head	*Transition (I/O)*	*Tail*	*UIOs*	*UIO Tail*
1	r/-	1	b/0 b/1	2
1	a/1	4	a/1 b/1	1
1	b/0	2	b/1 b/1	2
2	r/-	1	b/0 b/1	2
2	a/0	3	a/0 b/1	1
2	b/1	2	b/1 b/1	2
3	r/-	1	b/0 b/1	2
3	a/0	5	a/-	5
3	b/0	4	a/1 b/1	1
4	r/-	1	b/0 b/1	2
4	a/1	5	a/-	5
4	b/0	1	b/0 b/1	2
5	r/-	1	b/0 b/1	2
5	a/-	5	a/-	5
5	b/1	1	b/0 b/1	2

Experiences show that for most FSMs, UIO sequences are no longer than six *i/o* symbols, and the resulting test sequences are much shorter than those constructed by the W-method or the D-method[3, 13].

In the following subsections, several improvements of this method will be discussed. The first improvement of the UIO-method (i.e. the UIOv-method) ensures that the UIO sequences selected from an FSM_s are all unique in a given FSM_i[21]. The second improvement (i.e. the SUIO-method) shortens the resulting test sequence by searching for the shortest connections between test segments rather than forming test subsequences and connecting them by reset inputs[13]. The third improvement (i.e. the MUIO-method) attempts to further reduce the length of the resulting test sequence by using multiple UIO sequences[14].

UIOv-method

The UIOv-method is based on the observation that although each UIO sequence is unique in FSM_s, the uniqueness of UIO sequences may not hold in a faulty FSM_i. That is, for some FSM_i, the input/output behaviour related to a UIO sequence may be exhibited by more than one state. This method requires the following procedures to be performed between Steps (1) and (2) of the UIO-method to ensure that the UIO for each state is unique in a given FSM_i:

(1) For each state $s_i \in S$, apply the UIO sequence for state s_i after FSM_i is brought to state s_i. (This procedure verifies that each state in FSM_s exists in FSM_i. In fact, a similar procedure is suggested for the D- and W-methods for the same purpose where the DS or W-set derived from FSM_s is applied to a given FSM_i after FSM_i is brought to state s_i, for each $s_i \in S$.)

(2) For each state $s_i \in S$, apply the UIO sequence for state s_i after FSM_i is brought to each state s_j, where the input part of the UIO sequence for state s_j is different than that of the UIO sequence for state s_i. (This procedure verifies that the UIO for each state is unique in the given FSM_i. Provided that the first procedure is performed, a similar procedure is not necessary for the D- and W-methods, since the output sequences generated by the FSM_i in response to the DS or W-set must be unique for each state.)

As an example, consider the FSM *M* given in Figure 1. The first procedure requires the application of the UIO sequences listed in Table 1a to the corresponding states. The second procedure requires the application of the UIO sequence for s_1 (or s_2) to states s_3, s_4 and s_5; the UIO sequence for s_3 (or s_4) to states s_1, s_2 and s_5. Since the input part of the UIO sequence for state s_5 is a prefix of that of for state s_3 or state s_4, it is not applied to the other states. The rest of the UIOv-method is the same as the UIO-method.

REFERENCES

1 **Gill, A** *Introduction to the Theory of Finite-state Machines,* McGraw-Hill, New York (1962)
2 **Dahbura, A T, Sabnani, K K and Uyar, M U** 'Formal methods for generating protocol conformance test sequences', *Proc. IEEE* Vol 78 No 8 (1990) pp 1317–1325
3 **Sidhu, D and Leung, T-K** 'Formal methods for protocol testing: A detailed study', *IEEE Trans. Softw. Eng.,* Vol 15 No 4 (1989) pp 413–426
4 **Sidhu, D** 'Protocol testing', in **L Logrippo, R L Probert and H Ural** (eds) *Protocol Specification, Testing and Verification 10,* North-Holland, Amsterdam (1990) pp 47–68
5 **Wang, B and Hutchison, D** 'Protocol testing techniques', *Comput. Commun.* Vol 10 No 2 (1987) pp 79–87
6 **Naito, S and Tsunoyama, M** 'Fault detection for sequential machines by transition tours', *Proc. 11th IEEE Fault Tolerant Computing Symposium* (1981) pp 238–243
7 **Sarikaya, B and Bochmann, G v** 'Synchronization and specification issues in protocol testing', *IEEE Trans. Commun.* Vol 32 (1984) pp 389–395
8 **Uyar, M U and Dahbura, A T** 'Optimal test sequence generation for protocols: Chinese postman algorithm applied to Q.931', *Proc. IEEE Global Telecommun. Conf.* (1986) pp 68–72
9 **Gonenc, G** 'A method for the design of fault detection experiments', *IEEE Trans. Comput.* Vol 19 (1970) pp 551–558
10 **Chow, T S** 'Testing software design modelled by finite-state machines', *IEEE Trans. Softw. Eng.* Vol 4 No 3 (1978) pp 178–187
11 **Sabnani, K K and Dahbura, A T** 'A new technique for generating protocol tests', *Proc. 9th Data Commun. Symposium,* Whistler Mountain, BC, Canada (1985) pp 36–43
12 **Sabnani, K K and Dahbura, A T** 'A protocol test generation procedure', *Comput. Networks & ISDN Syst.* Vol 15 (1988) pp 285–297
13 **Aho, A V, Dahbura, A T, Lee, D and Uyar, M U** 'An optimization technique for protocol conformance test sequence generation based on UIO sequences and rural Chinese postman tours', in **S Aggarwal and K Sabnani** (eds) *Protocol Specification, Testing, and Verification 8,* North-Holland, Amsterdam (1988) pp 75–86
14 **Shen, Y N, Lombardi, F and Dahbura, A T** 'Protocol conformance testing using multiple UIO sequences', in **E Brinksma, G Scollo and C A Vissers** (eds.) *Protocol Specification, Testing and Verification 9,* North-Holland, Amsterdam (1989) pp 131–144
15 **Kuan, M-K** 'Graphic programming using odd or even points', *Chinese Math.,* Vol 1 (1962) pp 273–277
16 **Kohavi, Z** *Switching and Finite Automata Theory,* McGraw-Hill, New York (1978)
17 **D Rayner** (ed), ISO/TC97/SC21, *OSI conformance testing methodology and framework – Part 1-5, ISO 2nd DP9646-1 revised text,* Vancouver, Canada (December 1987)
18 **Rayner, D** 'OSI conformance testing', *Comput. Networks & ISDN Syst.* Vol 14 (1987) pp 79–98
19 **Edmonds, J and Johnson, E L** 'Matching, Euler tours and the Chinese postman', *Math. Program.* Vol 5 (1973) pp 88–124
20 **Busacker, R G and Saaty, T L** *Finite Graphs and Networks,* McGraw-Hill, New York (1965)
21 **Chan, W Y L, Vuong, S T and Ito, M R** 'An improved protocol test generation procedure based on UIOS' *Proc. ACM SIGCOMM'89* (1989) pp 283–294

22 **Lenstra, J K and Rinnooy Kan, A H G** 'On general routing problems', *Networks,* Vol 6 (1976) pp 273–280
23 **Ural, H and Lu, Y** *An improved method for test sequence generation,* Technical Report, TR-90-12, Department of CSI, University of Ottawa (March 1990)
24 **Dahbura, A T and Sabnani, K K** 'An experience in estimating the fault coverage of a protocol test', *Proc. IEEE INFOCOM'88* (1988) pp 71–79
25 **Sidhu, D and Leung, T-K** 'Fault coverage of protocol test methods', *Proc. IEEE INFOCOM'88* (1988) pp 80–85
26 **Chen, M-S, Choi, Y and Kershenbaum, A** 'Approaches utilizing segment overlap to minimize test sequences', in **L Logrippo, R L Probert and H Ural** *Protocol Specification, Testing and Verification 10,* North-Holland, Amsterdam (1990) pp 85–98
27 **Miller, R E and Paul, S** 'Generating minimal length test sequences for conformance testing of communication protocols', *Proc. 3rd Int. Workshop on Protocol Test Systems* (1990)
28 **Yang, B and Ural, H** 'Protocol conformance test generation using multiple UIO sequences with overlapping', *Proc. ACM SIGCOMM'90,* Philadelphia, PA (1990) pp 118–125
29 **Boyd, S and Ural, H** 'On the complexity of optimal test sequence generation', *IEEE Trans. Softw. Eng.,* Vol 17 No 9 (1991) pp 976–978