EXPERIMENTAL EVALUATION OF FSM-BASED TESTING METHODS

[1]Rita Dorofeeva [2]Khaled El-Fakih [3]Stephane Maag [3]Ana R.Cavalli [1]Nina Yevtushenko
[1]Tomsk State University, Russia, [2]American University of Sharjah, UAE, [3]Institut National des Télécommunications, France
drf@kitidis.tsu.ru, yevtushenko@elefot.tsu.ru,
kelfakih@aus.ac.ae{Ana.Cavalli,Stephane.Maag}@int-evry.fr

Abstract

The development of test cases is an important issue for testing software, communication protocols and other reactive systems. A number of methods are known for the development of a test suite based on a formal specification given in the form of a finite state machine. Well-known methods are called the W, Wp, UIO, UIOv, DS, H and HIS test derivation methods. These methods have been extensively used by research community in the last years; however no proper comparison has been made between them. In this paper, we experiment with these methods to assess their complexity, applicability, completeness, fault detection capability, length and derivation time of their test suites. The experiments are conducted on randomly generated specifications and on a realistic protocol called the Simple Connection Protocol.

1. Introduction

The development of test cases based on a formal model is an important issue for software testing including conformance testing of communication protocols and other reactive systems. A number of methods are known for the development of a test suite based on a specification given in the form of a finite state machine. Well known methods are called the W, Wp, UIO, UIOv, DS, H and HIS test derivation methods [4, 6, 15, 19, 22, 21, 17]. For related surveys the reader may refer to [18, 1, 20, 13]. In particular, in the last years an increasing research has been developed on the application of these methods to object oriented software [2].

In FSM-based testing, one usually assumes that not only the specification, but also an implementation can be modeled as a deterministic FSM. If the behavior of an implementation FSM is different than the specified behavior, the implementation contains a fault. Two types of implementation faults are usually considered, namely output and transfer faults. An implementation has an *output fault* if the output of one of its transitions is different from that of the specification and an implementation has a *transfer fault* if the next state of a transition is different from that of the specification. Moreover, an implementation has *multiple faults* if it has many output or transfer faults [1, 4, 6, 15, 18, 22, 21].

The above methods, except the UIO method, each provides the following fault coverage guarantee: If the specification can be modeled by an FSM with n states and if an implementation can be modeled by an FSM with at most m states, where m is larger or equal to n, then a test suite can be derived by the method (for this given m) and the implementation passes this test suite if and only if it conforms (i.e. is equivalent) to the specification (that is, the implementation does not contain output and transfer faults). Moreover, all of the above methods assume that a reliable reset is available for each implementation under test (written as 'r'). This implies that the test suite can be composed of several individual test cases, each starting with the reset operation.

All of the above methods use certain state distinguishing input sequences (called state identifiers) in test derivation, and thus, can be only applied when all states of the specification FSM are pairwise distinguishable. Moreover, the length of a derived test suite essentially depends how state identifiers are selected. The DS method can be thought of as a particular case of the W method. However, the DS method is not always applicable even for complete reduced specifications [7]. The Wp method is an improvement to the W method and thus is expected to generate shorter test suites. Also, the UIOv method can be thought of as a particular case of the Wp and is expected to generate shorter test suites. The UIO method employs part of the UIOv method and thus generates shorter test suites. However, UIO test suites are not always complete [22]. In the so-called H method [12, 5], unlike all other methods, for

every state of the specification FSM, appropriate state identifiers are selected on the fly (i.e. not in advance) in order to shorten the length of the resulting test suite. The H method generates a complete test suite and is always applicable for any complete reduced specification.

The methods have been studied and applied by research community in the last years. However, no comparison has been provided on the applicability, completeness and efficiency of these methods. In this paper we implement and experiment with the above methods (for the case when the number of states of an implementation is not more than that of the specification) in order to: (i) determine and compare the length and derivation time of their test suites, (ii) determine how often the UIO and DS methods are not applicable and how often the UIO method (when applicable) generates incomplete test suites, (iii) determine the fault detection capability of the UIO-based incomplete test suites, and (iv) compare length of obtained test suites with that obtained using the corresponding theoretic worst-case upper-bound. The experiments are conducted on randomly generated specifications and on a realistic protocol called the Simple Connection Protocol (SCP) [3].

This paper is organized as follows. Section 2 defines notations for describing finite state machines and Section 3 includes an overview of the W, Wp, HIS, H, UIOv, UIO, and DS test derivation methods. Sections 4 and 5 include the experimental results and Section 6 concludes the paper.

2. Finite State Machines

This section contains the definition of basic concepts that are used in the rest of the paper.

Definition 2.1 A deterministic *finite state machine* is an initialized complete deterministic Mealy machine that can formally be defined as a 6-tuple $M = (S,X,Y,\delta_M,\lambda_M,s_1)$ [7], where S is a finite set of states, s_1 is the *initial state*, X is a finite set of input symbols, Y is a finite set of output symbols, δ_M is a next state (or transition) function: δ_M: $S\times X \rightarrow S$, λ_M is an output function: λ_M: $S\times X \rightarrow Y$. In usual way, functions δ_M and λ_M are extended to input sequences.

Definition 2.2 A FSM A is called *connected* if for each state $s\in S$ there exists an input sequence α_s that takes FSM A from the initial state to state s. The sequence α_s is called a *transfer* sequence for the state s.

Definition 2.3 A set Q of input sequences is called a *state cover set* of FSM M if for each state s_i of S, there is an input sequence $\alpha_i\in Q$ that takes FSM from the initial state to state s_i. If FSM is *connected*, i.e. if each state is reachable from the initial state, then a state cover set always exists. We further assume that the specification FSM M is a connected FSM[1] and consider only prefix-closed state cover sets, i.e. a state cover set contains all prefixes of each sequence.

Definition 2.4 Let $M = (S, X, Y, \delta_M, \lambda_M, s_1)$ and $I = (T, X, Y, \delta_I, \lambda_I, t_1)$ be two FSMs. In the following sections M usually represents a specification while I denoting an implementation. We say that two states s_j of M and t_i of I are *equivalent* [7], written $s_j \cong t_i$, if for each input sequence $\alpha \in X^*$ it holds that $\lambda_M(s_j, \alpha) = \lambda_I(t_i, \alpha)$. Otherwise, we say that states s_i and t_j are *distinguishable*, written $s_j \ncong t_i$.

Definition 2.5 An input sequence $\alpha \in X^*$ such that $\lambda_M(s_j, \alpha) \neq \lambda_I(t_i, \alpha)$ is said to *distinguish* the states s_j and t_i. FSMs M and I are *equivalent*, written $M \cong I$, (*distinguishable*, written $M \ncong I$) if their initial state are equivalent (distinguishable).

Definition 2.6 An FSM is said to be *reduced* if its states are pair-wise distinguishable.

Definition 2.7 We say that I *conforms to* M if and only if FSMs I and M are equivalent. In other words, for each input sequence the output responses of M and I coincide [7, 17, 18].

Definition 2.8 Given a specification FSM M, the *fault domain* $J(X)$ of M is the set of all possible implementations of M defined over the input alphabet X of M. Similar to [16] we let $J_m(X)$ denote the set of all complete FSMs defined over the input alphabet X with up to m states. A *test suite* TS is a finite set of finite input sequences of the specification FSM M.

Definition 2.9 A test suite TS is *m-complete* if for each implementation $I \in J_m(X)$ that is distinguishable from M, there exists a sequence in TS that distinguishes M and I.

[1] We consider only connected FSMs without loss of generality, since any state of an FSM that is unreachable from the initial state, does not influence the behavior of the FSM.

3. Overview of Test Derivation Methods

This section includes an overview of test derivation methods. All these methods, except the UIO, have two phases. Tests derived for the first phase check that each state presented in the specification also exists in the implementation, while tests derived for the second phase check all (remaining) transitions of the implementation for correct output and ending state as defined by the specification. For identifying the state during the first phase and for checking the ending states of the transitions in the second phase, certain state distinguishing input sequences are used.

The only difference between the above methods is how such distinguishing sequences are selected. In the original W method, a so-called characterization set W (or simply W set) is used to distinguish the different states of the specification. The Wp method uses the W set during the state identification phase (the first phase) while only an appropriate subset, namely a corresponding state identifier, is used when checking the ending state of a transition. In the HIS method a family of state identifiers is used for state identification as well as for transition checking. In the UIOv method, which is a proper sub-case of the Wp method, the state identifier of each state has a single Unique Input Output (UIO) sequence. Such an UIO allows to distinguish the expected ending state of a transition from all other states of the specification. A test suite is shortened; however, an *UIO* sequence may not exist for some states of a given specification FSM.

In subsection 3.2 we briefly describe the test derivation methods [4, 6, 15, 22, 12]. Meanwhile, we describe the state identification utilities used by these methods.

3.1. State identification facilities

In order to check that each state and each transition defined in the specification also exists in the implementation, the methods use certain input/output behaviors that can distinguish the states of an FSM. Consider a reduced specification FSM $M = (S,X,Y,\delta_M,\lambda_M,s_1)$.

A *characterization set* of the FSM M, often simply called a *W set*, is a set of input sequences defined over the input alphabet of M that satisfies the condition: For any two states s_i and s_j, $i \neq j$, $\exists\ \beta \in W$ such that $\lambda_M(s_i, \beta) \neq \lambda_M(s_j, \beta)$. That is for any two states of M, the W set includes a sequence that distinguishes these states. A W set always exists for a reduced FSM.

Given state $s_j \in S$ of FSM M, a set W_j of input sequences is called a *state identifier* (or a *separating set*) of state s_j if for any other state s_i there exists $\alpha \in W_j$ such that $\lambda_M(s_j, \alpha) \neq \lambda_M(s_i,\alpha)$. A *separating family* [20] (or a family of *harmonized identifiers* [14, 15]) is a collection of state identifiers W_j, $s_j \in S$, which satisfy the condition: For any two states s_j and s_i, $j \neq i$, there exist $\beta \in W_j$ and $\gamma \in W_i$ which have common prefix α such that $\lambda_M(s_j, \alpha) \neq \lambda_M(s_i, \alpha)$. A separating family always exists for a reduced FSM.

Given a state s_i of the specification FSM M, let α_i be an input sequence such that for any state s_j, $i \neq j$, $\lambda_M(s_i, \alpha_i) \neq \lambda_M(s_j, \alpha_i)$. Then α_i/β_i where $\beta_i = \lambda_M(s_i, \alpha_i)$, is said to be a *Simple Input/Output Sequence* [9] or a *Unique Input/Output* (*UIO*) *sequence* [17], for state s_i. Let each state of the specification FSM have an UIO α_i/β_i and $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ be the set of input parts of all these UIOs. Then the set $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ is known to be a W set. We note that a UIO sequence may not exist for some states of a reduced FSM.

Let α be an input sequence such that for any two states s_i and s_j , $i \neq j$, of the specification FSM M it holds that $\lambda_M(s_i, \alpha) \neq \lambda_M(s_j, \alpha)$. Then α is said to be a *distinguishing sequence* or *diagnostic sequence* (DS) [7, 8, 11] for the specification machine M. The set $\{\alpha\}$ is a W set.

For a distinguishing sequence α, let $\beta_i = \lambda_M(s_i, \alpha)$, $i = 1, 2, \ldots, n$. Then $\alpha/\beta_1, \alpha/\beta_2, \ldots \alpha/\beta_n$ are UIO sequences for the n states $s_1, s_2, \ldots, s_n$, respectively. A distinguishing sequence may not exist for some reduced FSMs [7].

As an application example, consider the specification FSM M shown in Fig. 1 with inputs $X = \{x, y\}$ and outputs $Y = \{0, 1\}$.

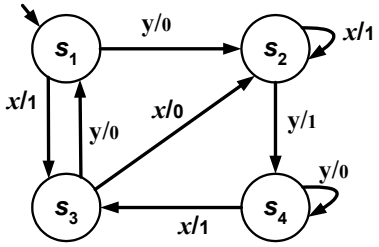


Figure 1. Specification FSM *M*

The specification M admits the set of sequences $\{x, y, yy\}$ as a W set. From the above, we get the following state identifiers, $W_1 = \{yy\}$, $W_2 = \{y\}$, $W_3 = \{x\}$, and $W_4 = \{x, yy\}$. In order to obtain a separating family from these identifiers, we

harmonize W_2 and W_3 by adding the input x into W_1 and W_2 and y into W_3, and obtain as a separating family the set $F = \{H_1, H_2, H_3, H_4\}$, where $H_1 = \{x, yy\}$, $H_2 = \{x, y\}$, $H_3 = \{x\}$, and $H_4 = \{x, yy\}$.

The FSM M has the sequence yyy as a distinguishing sequence. For states s_1, s_2, s_3, and s_4 of M, we have, in response to yyy, the output sequences 010, 100, 001, and 000, respectively.

3.2 Test Derivation Methods

Given a reduced complete specification FSM $M = (S, X, Y, \delta_M, \lambda_M, s_1)$, $|S|=n$, let W be a characterization set of M and $F = \{W_1,\ldots, W_n\}$ be a separating family of M.

The test derivation methods have two phases in order to test the equivalence of I and M.

Given a reduced complete specification FSM $M = (S, X, Y, \delta_M, \lambda_M, s_1)$, $|S|=n$, let W be a characterization set of M and $F = \{W_1,\ldots, W_n\}$ be a separating family of M.

The test derivation methods have two phases in order to test the equivalence of I and M.

State identification phase:

This phase checks that each state specified by M also exists in I using a characterization set W (W, Wp, UIOv and DS methods), or a separating family F (HIS method). We note that for the UIOv method, the W set consists of the input parts of the UIOs of M and for the DS method the W set consists of a single sequence, namely a DS of M.

Given a prefix-closed state cover set $Q = \{\alpha_1, \alpha_2,\ldots, \alpha_n\}$ of the specification FSM, for each state $s_j \in S$, the state identification phase comprises the sequences:

$r.\alpha_j.H_j$ (in the HIS method) or

$r.\alpha_j.W$ (in the W, Wp, DS, and UIOv methods).

Each test sequence starts from the initial state, after the application of the reset input r. In this case, in the W, Wp, DS and UIOv methods, to identify the ending state s_j after applying an input sequence α_j, all the sequences contained in W are applied to I, separately. However, in the HIS it is enough to apply the sequences of the state identifier set H_j of s_j.

Transition testing phase:

This phase assures that for each transition of M there exists a corresponding transition in I. For this purpose, for each sequence $\alpha_j \in Q$ that takes the specification FSM to appropriate state s_j, and each $x \in X$ that takes the M from state s_j to state s_k, the transition testing phase includes the following set of test sequences:

$r.\alpha_j.x.H_k$ in the HIS method,

$r.\alpha_j.x.W_k$ in the UIOv and Wp methods, where $W_k \subseteq W$ is a state identifier or a corresponding UIO of the state s_k or

$r.\alpha_j.x.W$ in the W method.

If FSM $I = (T, X, Y, \Delta_I, \Lambda_I, t_1)$ has at most n states and passes the test sequences of both testing phases, then I is equivalent to the specification FSM, i.e. I is a conforming implementation.

The UIO method [17] is based on UIO sequences and it employs only the second testing phase. It was originally claimed [17] that a test suite derived using the UIO method is complete w.r.t. all implementations with up to n states. However, a counter-example was later given in [22] that proves that the UIO method does not guarantee complete fault coverage.

The H method [12, 5] can be regarded as an improvement to the HIS method. The main idea of the H method is not to use a priori derived state identifiers. State identifiers are constructed, based on already derived test cases, in order to distinguish the ending states of transitions. Thus, different state identifiers can be used when testing transitions with the same ending state.

Example: As an application example of the above methods, consider the FSM M of Figure 1. We recall that M admits as a characterization set the set $W = \{x, y, yy\}$, as state identifiers the sets $W_1 = \{yy\}$, $W_2 = \{y\}$, $W_3 = \{x\}$, and $W_4 = \{x, yy\}$, and the set $F = \{H_1, H_2, H_3, H_4\}$ as a separating family of harmonized state identifiers, where $H_1 = \{x, yy\}$, $H_2 = \{x, y\}$, $H_3 = \{x\}$, and $H_4 = \{x, yy\}$. Moreover, M has the state cover set $Q = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, where $\alpha_1 = \varepsilon$, $\alpha_2 = y$, $\alpha_3 = x$, and $\alpha_4 = yy$.

W method application:

Based on the above sets, in the W method, the state identification phase yields the test sequences TS_W: $r.\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}.W$ and the transition testing phase yields the test sequences: $r.\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}\{x,y\}W$. We replace the α's and W's in the above sequences by their corresponding values and then remove from the obtained set those sequences that are proper prefixes of other sequences and obtain $TS_W = \{rxxx, rxxyy, rxyx, rxyyy, ryxx, ryxyy, ryyxx, ryyxyy, ryyyx, ryyyyy\}$ of total length 49.

Wp method application:

In the Wp method, in addition to the state identification sequences, the transition testing phase yields the sequences TS_{Wp}: $r.\alpha_1.x.W_3 + r.\alpha_1.y.W_2 + r.\alpha_2.x.W_2 + r.\alpha_2.y.W_4 + r.\alpha_3.x.W_2 + r.\alpha_3.y.W_1 + r.\alpha_4.x.W_3 + r.\alpha_4.y.W_4$. We replace the α's and W's by their corresponding values and obtain TS_{Wp} = {*rxxy*, *rxyyy*, *ryxy*, *ryyxx*, *ryyyx*, *ryyyyy*} of total length 29. Similarly, we apply the HIS method and obtain the test suite TS_{HIS} = {*rxxx*, *rxxy*, *rxyx*, *rxyyy*, *ryxx*, *ryxy*, *ryyxx*, *ryyyx*, *ryyyyy*} of total length 41. We note here that for this example, the first parts for the state identification phases in Wp and HIS methods coincide and for this reason the Wp returns a shorter test suite knowing that we do not need to harmonize, as in the HIS, the state identifiers.

H method application:

In the H method, in the state identification phase, we obtain the sequences {*rxx*, *ryx*, *ryyx*, *ryyyy*} using the same state identifiers W_1= {*yy*}, W_2 = {*y*}, W_3 = {*x*}, and W_4 = {*x*, *yy*}. In the transition testing phase, consider the transition from state s_3 under input *x* with the final state s_2. In order to test this transition, we apply the input sequence $r.\alpha_3.x.y = r.x.x.y$ instead of applying the sequences $r.\alpha_3.x.H_2 = r.x.x.\{x, y\}$ as in the HIS method. This is done since the input *y* is already applied, in the state identification phase, at each state of the implementation (if the implementation passes the sequences of the state identification phase) and thus we can use *y* in the transition testing phase instead of using as in the HIS method the harmonized state identifier H_2 = {*x*, *y*}. Similarly, we derive identifiers to check all other transitions. The obtained test suite TS_H = {*rxxy*, *rxyyy*, *ryxy*, *ryyxx*, *ryyyyyy*} is of total length 25.

DS method application:

When constructing a characterization set we derive a shortest distinguishing sequence for each pair of different states of the specification FSM. However, sometimes such sequences do not yield the shortest test suite. By direct inspection, one can assure that the FSM *M* of our working example has a distinguishing sequence *yyy*. Thus, we can select as harmonized state identifiers the sets Z_1 = {*yy*}, Z_2 = {*y*}, Z_3 = {*yyy*}, and Z_4 = {*yyy*}. In this case, the HIS method returns a test suite TS_{DS} = {*rxxy*, *rxyyy*, *ryxy*, *ryyxyyy*, *ryyyyyy*} with total length 27.

4. Experimental Results

In this section we experiment with the above described methods with the following objectives: (i) determine and compare the length and derivation time of their test suites, (ii) determine how often the UIO and DS methods are not applicable and how often the UIO method (when applicable) generates incomplete test suites, (iii) determine the fault detection capability of some UIO-based incomplete test suites, and (iv) compare length of the obtained test suites with the theoretical upper-bound.

For every pair of (different) states of *M*, we generate a shortest input sequence that distinguishes this pair [7, 19]. The set of all obtained distinguishing sequences is a W set of the specification FSM M. The subset of all obtained sequences that distinguish a state s_i from all other states of *M* is a state identifier (W_i) of s_i. The set of all state identifiers is a separating family (*F*) of *M*. An algorithm for deriving a distinguishing sequence (if exists) for a given FSM is given in [11]. We derive UIO sequences using the distinguishing tree algorithm given in [7].

Table 1 provides a comparison between the length of test suites obtained by these methods and their derivation time in seconds. The comparison is based on randomly generated completely specified reduced specifications with a varying number of inputs/outputs and states (*n*).

Each row of Table 1 corresponds to a group of 50 randomly generated completely specified reduced specifications. For each of these specifications we use the W, Wp, HIS, UIOv and H methods to derive corresponding test suites. Moreover, we also derive test suites using the UIO and DS methods when the specifications have UIO and DS sequences. Then, we calculate the average length and derivation time (in seconds) of test suites generated for each group using each of these methods as shown in Columns VI to XII, respectively. We also calculate for each group how many times (out of 50) the UIO and DS methods were applicable and the average length for their test suites as shown in Columns XIII and XIV, respectively.

Table 1: A Summary of Conducted Experiments

I- Groups of 50 Experiments	II- Number of State, n	III- Number of Inputs, k	IV- Number of Outputs	V- No. of Transitions (nxk)	VI- Average Length of W Test Suites (average time of test derivation)	VII- Average Length of Wp Test Suites (average time of test derivation)	VIII- Average Length of HIS Test Suites (average time of test derivation)	IX- Average Length of UIOv Test Suites (average time of test derivation)	X- Average Length of H Test Suites (average time of test derivation)	XI Average Length of UIO Test Suites (average time of test derivation)	XII- Average Length of DS Test Suites (average time of test derivation)	XIII- No. of Times (out of 50) UIO is Applicable	XIV- No. of times (out of 50) DS is Applicable
1	30	6	6	180	2545 (0.82)	1626 (0.06)	1649 (0.26)	1406 (0.78)	1105 (0.12)	783 (0.58)	934 (0.83)	50	19
2	30	8	8	240	2985 (1.09)	1919 (1.10)	1946 (0.40)	1406 (0.92)	1320 (0.16)	1012 (0.76)	1281 (1.00)	50	34
3	30	10	10	300	3393 (1.48)	2175 (2.06)	2243 (0.59)	1477 (1.54)	1568 0.24()	1196 (1.48)	1493 (1.83)	50	47
4	50	4	4	200	4213 (1.15)	2637 (1.68)	2635 (0.53)	4077 (2.12)	1711 (0.21)	996 (1.29)	0	46	0
5	50	6	6	300	5203 (2.98)	3261 (3.04)	3261 (0.97)	3447 (3.06)	2142 (0.30)	1383 (2.04)	1777 (3.00)	50	6
6	50	8	8	400	6049 (4.49)	3828 (3.22)	3828 (1.59)	3167 (2.54)	2534 (0.47)	1820 (2.02)	2177 (7.67)	50	9
7	50	10	10	500	6773 (5.77)	4305 (5.54)	4375 (2.37)	3126 (4.38)	2852 (0.69)	2233 (3.58)	2710 (8.00)	50	30
8	60	6	6	360	6891 (6.26)	4138 (4.90)	4138 (1.58)	4933 (5.32)	2697 (0.48)	1,690 (3.34)	2250 (19.0)	50	1
9	60	8	8	480	7814 (6.79)	4844 (5.14)	4844 (2.67)	4250 (4.30)	3185 (0.71)	2229 (3.34)	2660 (48.0)	50	12
10	60	10	10	600	8978 (10.37)	5443 (8.48)	5490 (3.91)	4178 (6.20)	3656 (1.16)	2758 (5.16)	3269 (9.50)	50	15
11	70	6	6	420	8245 (6.76)	5047 (7.40)	5047 (2.27)	6795 (9.18)	3274 (0.68)	2010 (5.20)	2650 (11.0)	50	1
12	70	8	8	560	9271 (9.71)	5899 (7.80)	5899 (3.30)	5539 (6.72)	3898 (1.01)	2639 (4.86)	3625 (22.0)	50	2
13	70	10	10	700	11012 (14.65)	6673 (13.6)	6694 (4.83)	5280 (10.2)	4344 (1.42)	3262 (8.16)	3892 (55.5)	50	13
14	80	6	6	480	9890 (10.25)	6082 (12.5)	6078 (3.04)	9296 (17.0)	3882 (0.87)	2328 (8.72)	-	49	0
15	80	8	8	640	11243 (15.58)	6980 (11.5)	6980 (4.64)	7132 (10.6)	4588 (1.48)	3055 (7.18)	4200 (47.0)	50	1
16	80	10	10	800	13144 (20.93)	7892 (17.7)	7892 (6.79)	6624 (13.8)	5216 (2.15)	3771 (10.4)	4466 (35.5)	50	3
17	90	6	6	540	11522 (14.09)	7005 (18.4)	7045 (4.19)	12004 (27.7)	4426 (1.18)	2659 (13.8)	-	50	0
18	90	8	8	720	13254 (20.36)	8138 (15.5)	8159 (6.30)	8624 (15.2)	5285 (2.02)	3466 (9.58)	4716 (39.0)	50	3
19	90	10	10	900	15129 (28.59)	9194 (25.1)	9194 (9.13)	7911 (20.0)	6078 (2.78)	4293 (14.8)	5935 (76.0)	50	8
20	100	6	6	600	13261 (18.35)	8031 (23.9)	8031 (5.52)	14576 (37.1)	4976 (1.57)	3012 (17.6)	-	50	0
21	100	8	8	800	15091 (26.61)	9332 (22.1)	9332 (8.43)	10486 (23.1)	6020 (2.57)	3892 (14.3)	-	50	0
22	100	10	10	1,000	17204 (41.99)	10503 (33.7)	10503 (11.79)	9248 (27.7)	6880 (3.75)	4810 (20.0)	6602 (36.0)	50	7

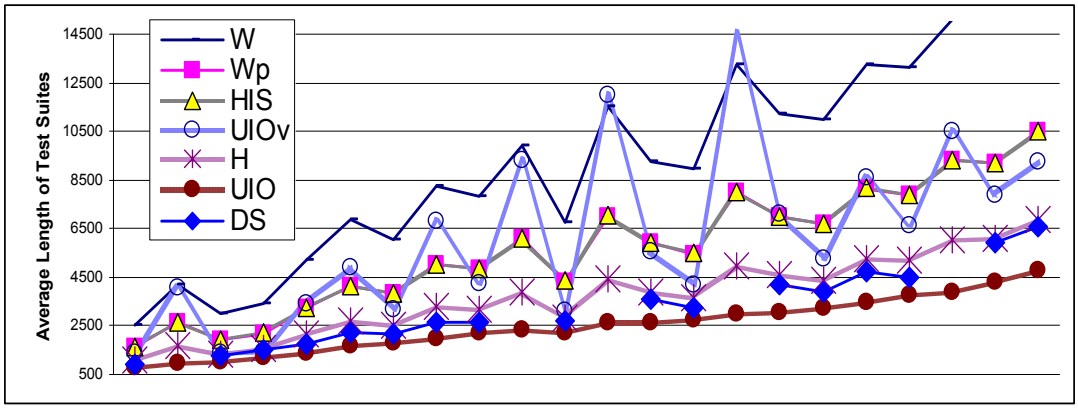


Figure 2. Average length of the W, Wp, HIS, UIOv, H, UIO and DS Test Suites.

Figure 2 depicts the average length of test suites and sorted according to Column V (number of transitions), derived using the testing methods. The UIO method generates shorter test suites than all other methods. However, on average, most of the UIO test suites are incomplete (more analysis on the completeness of the UIO test suites is given below). The DS and H methods generate test suites of comparable length. However, unlike the H method, the DS method is not always applicable and the experiments show that it becomes less applicable as the ratio of the number of outputs to the number of transitions decreases (more detailed analysis is given below). The HIS and Wp test suites are of comparable length. The reason for that could be that state identifiers used by Wp method do not need to be harmonized and thus, are shorter than those used by HIS method. The HIS/Wp methods generate shorter suites than those of the W method, as expected. The UIOv did not perform better than the HIS and Wp method; that also is expected as UIOv is a particular case of the Wp-method. Moreover, differently from Wp-method only UIO sequences are used as state identifiers in the transition checking phase.

Figure 3 depicts the ratios of length of the test suites of the HIS/Wp, UIOv, H and UIO methods over the length of the W-based test suites for the (groups of) experiments depicted in rows 1 to 22 of Table 1. The HIS/Wp (UIOv, H, UIO, DS) test suites are on average 0.61 (0.65, 0.4, 0.27, 0.36) percent of those of the W method. According to these experiments these ratios, for all except the UIOv, are almost independent of the size of the specification. For the UIOv, in some cases, for large machines, when the number of inputs/outputs is small in comparison to the number of states (see for example rows 17 and 20 of Table 1), the UIOv produces test suites that are longer than those of the W method. The reason can be that the W set that contains all UIO is worse for test derivation than a distinguishability set that contains a shortest distinguishing sequence for each pair of states. On average, the ratio of the length of UIOv test suites over the length of the W test suites is 0.65.

For a given reduced FSM M with n states and k input symbols, the worst-case length of the test suite generated using the W, Wp, HIS, H, UIOv and UIO methods is of the order kn^3 for a reduced completely specified FSM [4, 20]. In practice, according to the conducted experiments, the test suites derived by these methods have length of the order cn^2, where the constant c is 5 for the W method and 4 for the Wp, HIS, UIOv and H methods. We also experimented how long are UIO sequences when those exist. According to the obtained results, the average length of UIO sequences equals to two and is independent of the size of the specification.

Columns XIII and XIV of Table 1 show how many times (out of the 50) the UIO and DS methods are applicable. On average, the UIO method is applicable to 99% of all conducted experiments and it seems that its applicability is independent of the size of the specification. On average, the DS method is applicable only to 19% of all conducted experiments and its applicability significantly decreased as the ratio of the number of outputs to the number of states of the specifications decreases. For example, on average, the DS was applicable to 65% of all conducted experiments when the ratio of the number of outputs to the number of states is more than 0.2 (rows 1, 2 and 3 of Table 1). However, this applicability drops to 14% when the ratio is between 0.1 and 0.2.

Knowing that the UIO method generates (when applicable) shorter test suites than all other methods, we conducted experiments to have an idea about the fault detection capability of these suites. Table 2

contains the details of these experiments. Each row of Table 2 corresponds to a group of 50 randomly generated completely specified reduced specifications. For each of these specifications, we derive a test suite using the UIO method (when applicable) and explicitly derive all possible faulty implementations of the specification. Then, we determine how many of these implementations are killed (detected) by the derived test suite. We consider in our experiments small size specifications with two to six states (n = 2, 3, 4, 5, 6), two inputs (k = 2) and outputs, since explicit enumeration is only possible for small size implementations.

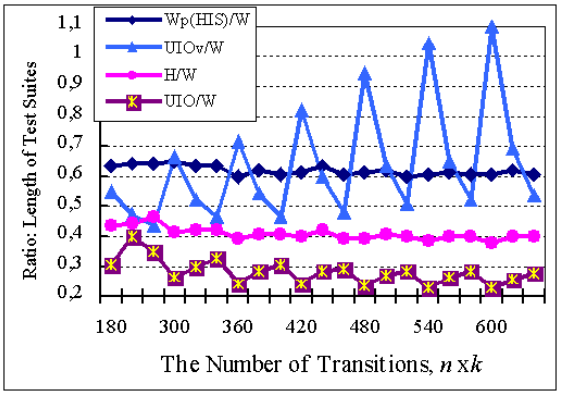


Fig. 3 Ratios: HIS/Wp, UIOv, H and UIO Test Suites over W test suites

As shown in Table 2, on average, when applicable, 41% of the UIO test suites are incomplete. Moreover, when the number of states compared with the number of outputs increases the possibility having incomplete test suites increases.

Table 2. Fault Coverage of UIO Test Suites

I- No. of States, n	II-No. of Inputs, k	III- No. of Outputs	IV- No. of Experiments (out of 50) where UIO is Applicable	V- No. of Complete Test Suites	VI- % of Complete UIO Test Suites
2	2	2	50	50	100
3	2	2	50	50	100
4	2	2	38	27	71
5	2	2	43	11	26
6	2	2	12	0	0
				Average	59

4.1 Case Study

This section presents the application and the comparison of the different test generation methods to the *Simple Connection Protocol* (*SCP*). The *SCP* allows to connect an entity called the *upper layer* with the entity called the *lower layer* (see Appendix). The upper layer performs a dialogue with SCP to fix the quality of service (*QoS*) desirable for the future connection. Once the negotiation is reached, SCP dialogues with the lower layer to ask for the establishment of a connection satisfying the quality of service previously negotiated. The lower layer accepts or refuses this connection request. If it accepts the connection, SCP informs the upper layer that the connection was established and the upper layer can start to transmit data towards the lower layer via SCP. Once the transmission of the data finished, the upper layer sends a message to close the connection. On the other hand, if the lower layer refuses the connection, the system allows SCP to make three requests before informing the upper layer that all the connection attempts failed. If the upper layer again wishes to be connected to the lower layer, it is necessary to restart the QoS negotiation with SCP from the beginning. This protocol illustrates the applicability of the above methods to real protocols.

We use the Specification and Description Language (SDL) [10] to describe the specification of the SCP protocol and then obtain an equivalent (with the same set of traces) FSM. The obtained FSM has n=26 states, 11 inputs, 12 outputs, and 286 transitions. We apply the W, Wp, HIS, and H test derivation methods to the obtained FSM and we get test suites of length 24490, 5229, 6125, and 4223, respectively. The Wp (HIS, H) test suites are on average 0.21 (0.25, 0.17) percent of those of the W method. Moreover, similar to the experiments reported in Table 1, the SCP W, Wp, HIS, and H test suites have length of the order cn^2, where the constant c is 5. We note that the UIO and DS methods are not applied to the SCP protocol since its FSM does not have a DS and UIOs.

5. Conclusion

In this paper, experiments with the W, Wp, HIS, H, UIOv, UIO, and DS FSM-based conformance testing methods have been presented. These experiments allow to determine and to compare the length and the effectiveness of the test suites generated by these methods. In particular, the experiments show that W

test suites are significantly longer than those derived by all other methods. The length of the Wp and HIS test suites almost coincide and are larger than those of the suites derived by DS, UIO, and H methods. Test suites derived by the UIO method are the shortest and the UIO method is applicable to 99% of the conducted experiments. However, when applicable, experiments with small size specifications show that 41% of the UIO test suites are incomplete. Test suites derived by the DS and H methods are comparable and significantly shorter than those derived by the W, Wp, HIS, and UIOv methods. However, the DS method is applicable only to 19% of all conducted experiments and its applicability significantly decreases as the ratio of the number of outputs to the number of transitions decreases. The order of all derived test suites is $O(cn^2)$ which is lower than the theoretical worst-case order $O(kn^3)$, where n is the number of states of the specification/implementation machines, k is the number of inputs, and c is a constant less than or equal to 5. The experiments are conducted on randomly generated specifications and on a realistic protocol called the Simple Connection Protocol.

6. References

[1] G. v. Bochmann, A. Petrenko, "Protocol testing: review of methods and relevance for software testing," *Proc. International Symposium on Software Testing and Analysis*, Seattle, 1994, pp. 109-123.

[2] I. Bourdonov, A. Kossatchev, A. Petrenko, and D. Galter. "KVEST: Automated Generation of Test Suites from Formal Specifications". Lecture Notes in Computer Science, N 1708, pp. 608-621, *Proceedings of World Congress of Formal Methods, Springer-Verlag*, Toulouse, France, 1999.

[3] W.-H. Chen, Executable test sequence for the protocol data flow property, *Proc. of Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, *FORTE/PSTV'01*, 2001, Cheju Island, Korea.

[4] T. S. Chow, "Test design modeled by finite-state machines," *IEEE Trans. SE*, vol. 4, no.3, 1978, pp. 178-187.

[5] R. Dorofeeva, K. El-Fakih, N. Yevtushenko, "An improved FSM-based conformance testing method", *Proc. of the IFIP 25th International Conference on Formal Methods for Networked and Distributed Systems*, Taiwan, Oct. '05 (to appear).

[6] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. SE*, vol. 17, no. 6, 1991, pp. 591-603.

[7] A. Gill, *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, 1962.

[8] F. C. Hennie, "Fault detecting experiments for sequential circuits", in *Proc. of 5th Annual Symposium on Switching Circuit Theory and Logical Design*, Princeton, 1964, pp. 95-110.

[9] E. P. Hsieh, "Checking experiments for sequential machines", *IEEE Trans. on Computers*, Vol. 20, No. 10, 1971, pp. 1152-1166.

[10] ITU-T, *Recommendation Z. 100-Specification and Description Language SDL* Geneva, 1992.

[11] Z. Kohavi, *Switching and Finite Automata Theory*. New York, McGraw- Hill, 1978, p. 658.

[12] I. Koufareva, M. Dorofeeva. "*A novel modification of W-method*". Joint Bulletin of the Novosibirsk computing center and A.P. Ershov institute of informatics systems. Series: Computing science, issue: 18, 2002, NCC Publisher, Novosibirsk. - PP. 69-81.

[13] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey", *Proceedings of the IEEE*, vol. 84, no. 8, 1996, pp. 1090-1123.

[14] A. Petrenko, "Checking experiments with protocol machines," *Proc. 4th Int. Workshop on Protocol Test Systems* (IWPTS), 1991, pp. 83-94.

[15] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das, "Nondeterministic state machines in protocol conformance testing," *Proc. of the IFIP 6th* IWPTS, France, 1993, pp. 363-378.

[16] A. Petrenko and N. Yevtushenko, "On test derivation from partial specifications", *Proceedings of the IFIP Joint International Conference, FORTE/PSTV'2000, on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification,* Italy, 2000, pp. 85-102.

[17] K. Sabnani and A. Dahbura, "A protocol test generation procedure," C*omputer Networks and ISDN Systems*, vol. 15, no. 4, 1988, pp. 285-297.

[18] D. P. Sidhu, and T. K. Leung, "Formal methods for protocol testing: a detailed study," *IEEE Trans. SE*, vol. 15, no. 4, 1989, pp. 413-426.

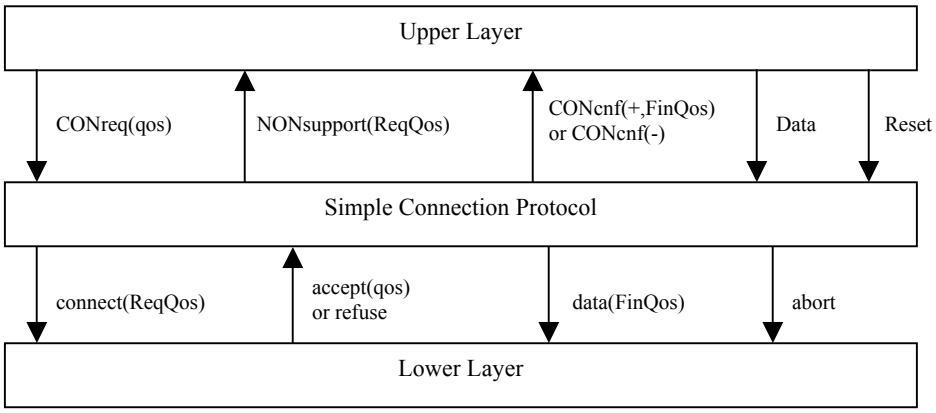
[19] M. P. Vasilevskii, "Failure diagnosis of automata," translated from Kibernetika, No.4, 1973, pp. 98-108.

[20] M. Yannakakis and D. Lee, "Testing finite state machines: fault detection", *Journal of Computer and System Sciences*, 50, 1995, pp. 209-227.

[21] N. Yevtushenko and A. Petrenko, *Test derivation method for an arbitrary deterministic automaton*, Automatic Control and Computer Sciences, Allerton Press Inc., USA, #5, 1990.

[22] S. T. Vuong, W.W.L. Chan, and M.R. Ito, "The UIOv-method for protocol test sequence generation," *Proc. of the IFIP TC6 2nd* IWPTS, North-Holland, 1989, pp. 161-175.

Appendix: The SCP Protocol Messages



CONreq(qos) with qos∈[0,3], connect(ReqQos) with ReqQos∈[0,3], accept(qos) with qos∈[0,3], CONcnf(+,FinQos) with FinQos∈[0,3] and data(FinQos) with FinQos∈[0,3].