Information and Software Technology 52 (2010) 1286–1297

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

FSM-based conformance testing methods: A survey annotated with experimental evaluation

Rita Dorofeeva [a], Khaled El-Fakih [b,*], Stephane Maag [c], Ana R. Cavalli [c], Nina Yevtushenko [a]

[a] Tomsk State University, 36 Lenin Str., 634050, Tomsk, Russia
[b] American University of Sharjah, College of Engineering, P.O. BOX 2666, Sharjah, United Arab Emirates
[c] TELECOM SudParis, CNRS UMR 5157, 9, rue Charles Fourier, 91011, Evry Cedex, France

ARTICLE INFO

Article history:
Received 8 December 2009
Received in revised form 12 April 2010
Accepted 9 July 2010
Available online 30 July 2010

Keywords:
Conformance testing
Protocol testing
Model based testing
Finite state machines

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. In this paper, we overview and 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 two realistic protocols called the Simple Connection Protocol and the ITU-T V.76 Recommendation.

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. The purpose of these tests is to determine whether a protocol implementation conforms to (i.e. is correct with respect to) its specification. Usually a conforming implementation is required to have the same input/output behavior as defined by the specification. In various application domains, such as telecommunication systems, communication protocols and other reactive systems, the specification can be represented in the form of a finite state machine (FSM). In particular, FSMs are the underlying models for formal description techniques, such as SDL and UML State Diagrams. 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 (FSM). Well-known methods are called the W [1,2], Wp [3], UIO [4], UIOv [5], DS [6], HSI [7–9] and the H [10,11] test derivation methods. For related surveys and tools the reader may refer to [12–17]. Moreover, in the last years an increasing research has been developed on the application of these methods to object oriented software [18].

In FSM-based testing, one usually assumes that not only the specification, but also an implementation can be modeled as an FSM. In order to check if a given implementation under test (IUT), assumed to be a black-box, conforms to its specification, input/outputs pairs of test sequences (*test cases*) are derived from the given specification. The inputs of these tests are then applied to an implementation and outputs generated (observed) by the implementation are compared with expected outputs. If the outputs do not match, then the implementation has a fault. A test suite is called *complete* if it detects every faulty implementation w.r.t. the considered fault model. All of the above methods, except of the UIO method, each provides the following *complete fault coverage* guarantee: if the specification can be modeled by a reduced deterministic FSM with n states and if an implementation can be modeled by a complete deterministic FSM with at most m states, then a test suite can be derived by the method (for this given m) such that the implementation passes this test suite if and only if it conforms to the specification (i.e., the implementation and the specification have the same input/output behaviors). Guessing the bound of m is an intuitive process based on the knowledge of a specification and the class of implementations that have to be tested for conformance and their interior structure [12]. Usually, it is assumed that m equals or is greater than n [2,3,5,8,9,12,15].

All of the above methods assume that a reset function (written as 'r') is available that allows the reliable reset of an implementation under test. This implies that the test suite can be composed of individual test cases, each starting with the reset operation. We note that there is another group of methods [6,19–21] that do not use this assumption. These methods usually yield much longer test suites and are not considered in the paper.

* Corresponding author. Tel.: +971 515 6 5152492; fax: +971 971 6 515 2979.
E-mail addresses: drf@kitidis.tsu.ru (R. Dorofeeva), kelfakih@aus.edu (K. El-Fakih), Stephane.Maag@it-sudparis.eu (S. Maag), Ana.Cavalli@it-sudparis.eu (A.R. Cavalli), ninayevtushenko@yahoo.com (N. Yevtushenko).

0950-5849/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2010.07.001

All the above methods use certain state distinguishing input sequences (called *state identifiers*) for test derivation, and thus, can be only applied when all states of the specification FSM are pair-wise distinguishable (i.e. the specification is reduced). Moreover, the length of a derived test suite essentially depends how state identifiers are selected. The DS method uses a distinguishing sequence of the specification FSM and 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 [22]. The Wp method is an improvement of the W method and the UIOv method can be thought of as a particular case of the Wp. The UIO method employs part of the UIOv method. However, UIO test suites are not always complete [5]. We also note that the W method (and correspondingly, Wp- and UIOv-methods) is not always applicable for partial reduced specifications [9,16]. The reason is not each even reduced partial FSM has a W set that distinguishes any pair of states of the specification. The HSI method, another improvement of W method, can be applied for each reduced deterministic specification FSM. The H method is an improvement of the HSI method and unlike all other methods, in the H method, for every state of the specification FSM, appropriate state identifiers are derived on-the-fly based on already derived part of a test suite in order to shorten the length of the resulting test suite. Similar to HSI method, H method is always applicable for any reduced specification.

During the past years a lot of research work has been done on developing complete FSM-based testing methods [1,3,5,8,9,15], studying the fault coverage of test suites [23,24], developing testing computer-aided tools [13,25], and using the FSM-based methods in the development of test cases from specifications modeled using other formalisms as extended FSMs (EFSMs) and labelled transition systems LTSs (related survey in [12]). Moreover, these methods have been used in order to derive tests for realistic protocols. For example, Refs. [26] and [15] include a comparison of the application of the UIO, DS and W methods to a subset of the NBS Class 4 transport (TP4) [27]. The application of the Wp method for a control subset of the XTP protocol is presented in [28] and the application of the HSI method [7,9,29] to the OSI session protocol has been presented in [25]. The application of the UIO method [30] to the ISDN protocol LAPD was conducted at AT&T and reported in [31]. These applications and others [26] have demonstrated that formal methods for conformance test generation are quite effective for rather complex communication protocols.

The ongoing research work on the development of complete testing methods focuses on two issues, minimizing the length of derived test sequences and enhancing the applicability of the testing methods. For example, it is known that the Wp method generates shorter tests than its ancestor the W method as demonstrated in [3]. Also, it is known that the UIO and UIOv methods are more applicable than the DS method but there exist complete reduced specifications where the UIO method cannot be applied and thus, UIO method is less applicable than the W, Wp, HSI, and H methods. The W and Wp methods are not always applicable for partial reduced specifications while the H and HSI methods are applicable to any complete or partial reduced specification [9,16]. Moreover, given a reduced FSM M with n states and k input symbols, the worst-case length of a test suite generated using all the above W, Wp, HSI, and H methods (except DS and UIO methods) is of order $O(k^{m-n+1}n^3)$, where m is the upper bound on the number of states of an implementation of M [2].

In [15], Sidhu and Leung offered a proper comparison of the W, DS, and UIO methods. However, to the best of our knowledge, there is no paper that includes a comprehensive (experimental) evaluation of all FSM-based methods using the same set of specification FSMs. To this end, in this paper, we first present a detailed overview of all above methods including recent methods such as the HSI [7–9] and the H [10,11] methods. We describe test derivation using simple formulae and we add corresponding application examples. Second, we implement and experiment, using randomly generated specifications, with the above methods in order to:

a. Determine average length and execution time of test suites for the cases when an implementation has equal or larger number of states than the specification, i.e., when $m = n$ and $m > n$.
b. Study the effects of increasing the number of states and transitions on the length of obtained test suites.
c. Determine how often the UIO and DS methods are not applicable.
d. Determine how often the UIO method (when applicable) generates incomplete test suites and the fault detection capability of the UIO-based test suites.
e. Compare length of obtained test suites with the estimated theoretic worst-case upper-bound length.

Moreover, we apply the above test derivation methods to two realistic protocols, namely the simple connection protocol (SCP) and the V76 protocol [32], and compare the length of test suites derived using randomly generated specifications and specifications of the realistic protocols. A summary of obtained results can be found in Section 6.

We note that a preliminary version of this work appeared in [33]. In this paper we extend that work in many ways. For example, in this paper, we include more experiments and more thorough analysis, for example, we add experiments for the case when an implementation has more states than the specification (i.e. when $m > n$). In addition, we model and include experiments with the V.76 protocol [32] and we provide a detailed description and application example of the recently developed H method [10,11].

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, HSI, 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 for demonstrating the test derivation methods.

2.1. Definition

A deterministic *finite state machine* (FSM) is an initialized complete deterministic Mealy machine that can be formally defined as a 6-tuple $M = (S,X,Y,\delta_M,\lambda_M,s_1)$ [22], 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 the next state (or transition) function: δ_M: $S \times X \to S$, λ_M is the output function: λ_M:$S \times X \to Y$. In usual way, functions δ_M and λ_M are extended to input sequences.

2.2. Definition

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.

2.3. Definition

The *concatenation* of two sets V_1 and V_2 of input sequences is defined as $V_1 \cdot V_2 = \{v_1 \cdot v_2 \mid v_1 \in V_1,\ v_2 \in V_2\}$, where $v_1 \cdot v_2$ denotes the

concatenation of the sequences v_1 and v_2. Given alphabet Z, Z^n denotes the n-time concatenation of Z ($Z^n = Z \cdot Z^{n-1}$). By definition, Z^0 only contains the empty word ε; Z^* is used to denote the union of the sets Z^n over all n, i.e. the set of all finite words over Z.

2.4. Definition

Given a set V of words over alphabet Z, the *prefix closure* of V, written *Pref*(V), consists of all the prefixes of all words in V, i.e. $Pref(V) = \{\alpha | \exists \gamma\ (\alpha \cdot \gamma \in V)\}$. The set V is *prefix-closed* if $Pref(V) = V$.

2.5. Definition

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 the FSM is *connected* then there always exists a prefix-closed state cover set. 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.

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 represents an implementation.

2.6. Definition

Two states s_j of M and t_i of I are *equivalent* [22], 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 $s_j \ncong t_i$. An FSM is said to be *reduced* if its states are pair-wise distinguishable. FSMs M and I are *equivalent*, written $M \cong I$, (*distinguishable*, written $M \ncong I$) if their initial state are equivalent (distinguishable).

2.7. Definition

Given distinguishable states s_j of M and t_i of I, an input sequence $\alpha \in X^*$ such that $\lambda_M(s_j, \alpha) \neq \lambda_I(t_i, \alpha)$ is said to *distinguish* states s_j and t_i, written $s_j \ncong_\alpha t_i$. An input sequence that distinguishes the initial states of distinguishable FSMs M and I *distinguishes* FSMs M and I, written $M \ncong_\alpha I$.

2.8. Definition

We say that I *conforms to* M if and only if FSMs I and M are equivalent. In other words, I conforms to M if and only if the output responses of FSMs M and I to each input sequence coincide [4,15,22].

2.9. Definition

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 [40] we let $J_m(X)$ denote the set of all FSMs with at most m states defined over the input alphabet X.

2.10. Definition

A *test suite* TS is a finite set of finite input sequences of the specification FSM M. 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 of the UIO, have two phases. Tests derived using the first phase check that each state presented in the specification also exists in an implementation under test (IUT), while tests derived using 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 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 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 of the set W, namely a corresponding state identifier, is used when checking the ending state of a transition during the transition checking phase (the second phase). In the HSI 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 reduced specification FSM.

In Section 3.2 we briefly describe the test derivation methods [1,3,5,8,11]. 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 exist in an IUT, the methods use certain input/output behaviors [4,8,16,19,20,22,29,34] that can distinguish the states of an FSM. Consider a reduced specification FSM $M = (S, X, Y, \delta_M, \lambda_M, s_1)$.

3.1.1. Definition

A *characterization set* of the FSM M, often simply called a *W set*, is a set of input sequences of M such that for any two different states s_i and s_j of M, the W set includes a sequence β that distinguishes these states, i.e., $\lambda_M(s_i, \beta) \neq \lambda_M(s_j, \beta)$. A W set always exists for a complete reduced FSM.

3.1.2. Definition

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* [16] (or a family of *harmonized identifiers* [9,29]) is a collection of state identifiers W_j, $s_j \in S$, which satisfy the condition: for any two different states s_j and s_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.

3.1.3. Definition

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* or a *unique input/output* (*UIO*) *sequence*, 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.

3.1.4. Definition

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 a *diagnostic sequence* (DS) for the specification machine M.

If α is a distinguishing sequence of FSM M, then the set $\{\alpha\}$ is a W set of M. Moreover, let $\beta_i = \lambda_M(s_i, \alpha)$, $i = 1, 2, \ldots, n$. Then α/β_1, $\alpha/\beta_2, \ldots, \alpha/\beta_n$ are UIO sequences for states $s_1, s_2, \ldots, s_n$, respectively. A distinguishing sequence may not exist for some reduced FSM [22].

We note that algorithms for deriving a DS can be found in classical textbooks like [22,34] and algorithms for deriving a separating family and a characterization set can be found in [14,35]. For deriving UIOs we can use the distinguishing tree algorithm given in [22] or an algorithm given in [22,36]. The reader may refer to the textbooks [41,42] for detailed description of these algorithms.

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

The specification M admits the set of sequences $\{x, y, yy\}$ as a W set from which 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 while adding y into W_3, and obtain a separating family $F = \{H_1, H_2, H_3, H_4\}$, where $H_1 = \{x, yy\}$, $H_2 = \{x, y\}$, $H_3 = \{x\}$, and $H_4 = \{x, yy\}$.

The FSM M has the following UIO sequences: $\mathrm{UIO}_1 = \{yy/01\}$, $\mathrm{UIO}_2 = \{y/1\}$, $\mathrm{UIO}_3 = \{x/0\}$, $\mathrm{UIO}_4 = \{yyy/000\}$. Thus, the set of input parts of all these UIOs is a W set $\{yy, y, x, yyy\}$.

The FSM M has a distinguishing sequence yyy. Output sequences 010, 100, 001, and 000 are the responses of M to yyy at states s_1, s_2, s_3, and s_4, 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. Let also FSM $I = (T, X, Y, \delta_I, \Delta_I, t_1)$ be an implementation of M such that $|T| = m$ where $m \geqslant n$. The test derivation methods have two phases in order to test the equivalence of I and M.

3.2.1. State identification phase

Tests of this phase check that each state specified by M also exists in the implementation I and each state of I that is reachable from the initial state corresponds to an appropriate state of the specification. For this purpose, a characterization set W (W, Wp, UIOv and DS methods), or a separating family F (HSI method) is used. 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 contains a single sequence, namely a DS of M.

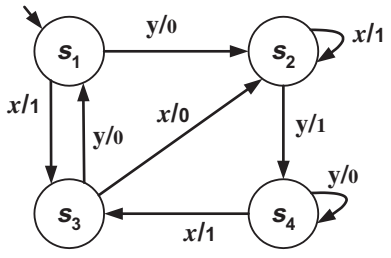


Fig. 1. Specification FSM M.

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

- $\boldsymbol{r \cdot \alpha_j \cdot \gamma \cdot H_t}$, in the HSI method, where $\gamma \in \mathit{Pref}(X^{m-n})$ and $H_t \in F$ is the state identifier of the state $s_t = \delta_M(s_1, \alpha_j\ \gamma)$ reached in the specification FSM M when the input sequence $r\ \alpha_j\ \gamma$ is applied, or
- $\boldsymbol{r \cdot \alpha_j \cdot \gamma \cdot W}$, $\gamma \in \mathit{Pref}(X^{m-n})$, in the W, Wp, DS, and UIOv methods.

Each test sequence is applied at 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_t after applying an input sequence $r \cdot \alpha_j \cdot \gamma$, all the sequences contained in W are applied to I, separately. However, in the HSI it is enough to apply the sequences of the state identifier set H_t of s_t. We note that when $m = n$ the set $\mathit{Pref}(X^{m-n})$ is empty.

3.2.2. Transition testing phase

Tests of this phase assure that for each state $t \in T$ that is reachable from the initial state in the implementation I and input $x \in X$, there exists a corresponding transition in I. For this purpose, for each sequence $\alpha_j \cdot \gamma$ (where $\alpha_j \in Q$ and $\gamma \in \mathit{Pref}(X^{m-n})$) and each $x \in X$, tests derived during the transition testing phase include the set of sequences:

- $\boldsymbol{r \cdot \alpha_j \cdot \gamma \cdot x \cdot H_k}$, in the HSI method, where $H_k \in F$ is a state identifier of the state s_k reached in the specification FSM M when the input sequence $r \cdot \alpha_j \cdot \gamma \cdot x$ is applied, or
- $\boldsymbol{r \cdot \alpha_j \cdot \gamma \cdot x \cdot W_k}$, in the UIOv and Wp methods, where $W_k \subseteq W$ is a state identifier or a corresponding UIO of the state s_k reached in the specification FSM M when the input sequence $r \cdot \alpha_j \cdot \gamma \cdot x$ is applied, or
- $\boldsymbol{r \cdot \alpha_j \cdot \gamma \cdot x \cdot W}$ in the W method.

If FSM I has at most m 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 [4], developed for the case when $m = n$, is based on UIO sequences and employs only the second testing phase. It was originally claimed [4] that a test suite derived using the UIO method is n-complete. However, a counter-example was later given in [5] that proves that the UIO method does not guarantee the complete fault coverage.

Example: As an application example of the above methods when the specification and implementation have equal number of states, i.e. $m = n$, consider the FSM M in Fig. 1. We recall that M admits a characterization set $W = \{x, y, yy\}$, state identifiers $W_1 = \{yy\}$, $W_2 = \{y\}$, $W_3 = \{x\}$, and $W_4 = \{x, yy\}$, and a separating family $F = \{H_1, H_2, H_3, H_4\}$, where $H_1 = \{x, yy\}$, $H_2 = \{x, y\}$, $H_3 = \{x\}$, and $H_4 = \{x, yy\}$. Moreover, a set $Q = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, where $\alpha_1 = \epsilon$, $\alpha_2 = y$, $\alpha_3 = x$, and $\alpha_4 = yy$, is a state cover set of the FSM M.

3.2.3. W method application

Based on the above sets, in the W method, the state identification phase yields the test sequences TS_W: $r \cdot \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} \cdot W$ and the transition testing phase yields the test sequences: $r \cdot \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}\{x,y\} \cdot W$. We replace the α's and W's in the above sequences by their corresponding values, 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.

3.2.4. Wp and HSI methods application

In the Wp method, in addition to the state identification sequences, the transition testing phase yields the sequences TS_{Wp}: $r \cdot \alpha_1 \cdot x \cdot W_3 + r \cdot \alpha_1 \cdot y \cdot W_2 + r \cdot \alpha_2 \cdot x \cdot W_2 + r \cdot \alpha_2 \cdot y \cdot W_4 + r \cdot \alpha_3 \cdot x \cdot W_2 + r \cdot$

$\alpha_3 \cdot y \cdot W_1 + r \cdot \alpha_4 \cdot x \cdot W_3 + r \cdot \alpha_4 \cdot y \cdot W_4$. We replace the αs 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 HSI method and obtain the test suite TS_{HSI} = {*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 HSI methods coincide and for this reason, the Wp returns a shorter test suite knowing that we do not need to harmonize, as in the HSI, the state identifiers.

3.2.5. DS method application

In the DS method, a characterization set has a single distinguishing sequence of the specification FSM. However, a shortest DS does not necessarily yield the shortest test suite. By direct inspection, one can assure that the FSM M of our working example has a shortest distinguishing sequence *yyy*. Thus, we can select state identifiers Z_1 = {*yy*}, Z_2 = {*y*}, Z_3 = {*yyy*}, and Z_4 = {*yyy*}. In this case, the DS method returns a test suite TS_{DS} = {*rxxy*, *rxyyy*, *ryxy*, *ryyxyyy*, *ryyyyyyy*} with total length 27.

3.2.6. UIO method application

In the running example, there exists an UIO for each state, namely UIO_1 = {*yy*/01}, UIO_2 = {*y*/1}, UIO_3 = {*x*/0}, UIO_4 = {*yyy*/000}. As the UIO method employs only the second testing phase, it returns a test TS_{UIO} = {*rxxy*, *rxyyy*, *ryxy*, *ryyxx*, *ryyyyyy*} of total length 25. Note that this test suite is 4-complete.

3.3. H method

The H method [10,11] can be regarded as an improvement to the HSI method. The main idea of the H method is not to use a priori derived state identifiers. In order to distinguish the ending states of transitions, state identifiers are constructed on-the-fly, based on already derived test cases. Thus, different state identifiers can be used when testing transitions with the same ending state. In particular, for every incoming transition of a state an appropriate state identifier is selected to shorten the length of the resulting test suite. In the following, we describe sufficient conditions and an algorithm for deriving a complete test suite [10]. We also apply the algorithm to our running example.

3.3.1. Theorem 1

Given a reduced deterministic specification M with n states and a state cover set Q of M, let TS be a finite set of finite input sequences of M that contains the set of sequences $Q.X^{m-n+1}$. The test suite TS is m-complete for $m \geqslant n$ if the following conditions hold:

1. For each two sequences α and β in Q which take FSM M to two distinct states, the TS has sequences $\alpha \cdot \gamma$ and $\beta \cdot \gamma$ where γ is a distinguishing sequence of states $\delta_M(s_1, \alpha)$ and $\delta_M(s_1, \beta)$.
2. For each sequence $\alpha \in Q$, each sequence $\beta \in Q$, and each non-empty sequence χ, $|\chi| \leqslant m - n + 1$, such that sequences α and $\beta \cdot \chi$ take the specification FSM M to two distinct states, the TS has the sequences $\beta \cdot \chi \cdot \gamma$ and $\alpha \cdot \gamma$, where γ is a distinguishing sequence of states $\delta_M(s_1, \beta\chi)$ and $\delta_M(s_1, \alpha)$.
3. For each sequence $\alpha \in Q$ and each two non-empty sequences χ_1 and χ_2, $|\chi_1| \leqslant m - n + 1$, $|\chi_2| \leqslant m - n + 1$, such that $\alpha \cdot \chi_1$ and $\alpha \cdot \chi_2$ take the specification FSM M to two distinct states, the TS has sequences $\alpha \cdot \chi_1 \cdot \gamma$ and $\alpha \cdot \chi_2 \cdot \gamma$, where γ is a distinguishing sequence of states $\delta_M(s_1, \alpha \cdot \chi_1)$ and $\delta_M(s_1, \alpha \cdot \chi_2)$.

According to the above Theorem, given a state s of M, different state identification sequences can be used when checking incoming transitions to state s, i.e. distinguishing sequences for the ending state of a transition can be derived on-the-fly using already a constructed part of a test suite.

Based on Theorem 1 the following algorithm, called the H method, has been developed for deriving an m-complete test suite.

3.4. Algorithm 1. The H test generation method

Input: a reduced deterministic specification FSM $M = (S, X, Y, \delta_M, \lambda_M, s_1)$ with n states, a prefix-closed state cover set Q of M, and the upper bound m on the number of states of an implementation FSM, where $m \geqslant n$.
Output: an m-complete, $m \geqslant n$, test suite TS.
Step 1. Derive the set of sequences $TS = QPref(X^{m-n+1})$
Step 2. For each two sequences α and β of the state cover set Q which take the FSM M to two distinct states, check if the set TS has sequences $\alpha \cdot \gamma$ and $\beta \cdot \gamma$ such that γ distinguishes states $\delta_M(s_1, \alpha)$ and $\delta_M(s_1, \beta)$ in the specification FSM. If there are no such sequences select a sequence γ that distinguishes states $\delta_M(s_1, \alpha)$ and $\delta_M(s_1, \beta)$ and add to TS sequences $\alpha \cdot \gamma$ and $\beta \cdot \gamma$.
Step 3. For each sequence $\alpha \in Q$, each sequence $\beta \cdot \chi \in QPref(X^{\leqslant m-n+1})$, $\beta \in Q$, such that α and $\beta \cdot \chi$ take the FSM M to two distinct states, check if the set TS has sequences $\beta \cdot \chi \cdot \gamma$ and $\alpha \cdot \gamma$ where γ distinguishes states $\delta_M(s_1, \beta \cdot \chi)$ and $\delta_M(s_1, \alpha)$ in the specification FSM. If there are no such sequences select sequence γ that distinguishes states $\delta_M(s_1, \beta \cdot \chi)$ and $\delta_M(s_1, \alpha)$ and add to TS sequences $\alpha_j \cdot \chi \cdot \gamma$ and $\alpha_i \cdot \gamma$.
Step 4. For each sequence $\alpha \in Q$, and each two non-empty sequences χ_1 and χ_2 in $QPref(X^{\leqslant m-n+1})$ which take the specification FSM to two distinct states from state $\delta_M(s_1, \alpha)$, check if the set TS has sequences $\alpha \cdot \chi_1 \cdot \gamma$ and $\alpha \cdot \chi_2 \cdot \gamma$ such that γ distinguishes states $\delta_M(s_1, \alpha \cdot \chi_1)$ and $\delta_M(s_1, \alpha \cdot \chi_2)$ in the specification FSM. If there are no such sequences select sequence γ that distinguishes states $\delta_M(s_1, \alpha \cdot \chi_1)$ and $\delta_M(s_1, \alpha \cdot \chi_2)$ and add to TS sequences $\alpha \cdot \chi_1 \cdot \gamma$ and $\alpha \cdot \chi_2 \cdot \gamma$.

Due to Theorem 1, the following statement holds.

3.4.1. Theorem 2

The set of input sequences TS returned by Algorithm 1 is an m-complete test suite for the given specification FSM M.

Example: As an application example of the H method when the specification and implementation have equal number of states, i.e. $m = n$, consider the FSM M in Fig. 1. We recall that a set Q = {α_1, α_2, α_3, α_4}, where $\alpha_1 = \varepsilon$, $\alpha_2 = y$, $\alpha_3 = x$, and $\alpha_4 = yy$, is a state cover of M. In the H method, in the state identification phase, we obtain the sequences {*rxx*, *ryx*, *ryyx*, *ryyyy*} using the same state identifiers as in the Wp method, W_1 = {*yy*}, W_2 = {*y*}, W_3 = {*x*}, and W_4 = {*x*, *yy*}.

In the transition testing phase, we need to test the following transitions: $s_3 - x \to s_2$, $s_3 - y \to s_1$, $s_2 - x \to s_2$, $s_4 - x \to s_3$, $s_4 - y \to s_4$. In order to test the first transition, we can use either the distinguishing sequence *xx* or the sequence *y*. As the sequence *y* is shorter, we derive the test sequence $r \cdot \alpha_3 \cdot x \cdot y = r \cdot x \cdot x \cdot y$ instead of the sequences $r \cdot \alpha_3 \cdot x \cdot H_2 = r \cdot x \cdot x \cdot \{x, y\}$ as in the HSI method. This can be done since the input *y* is already applied, in the state identification phase, at each state of the implementation, and when the implementation passes the state identification phase sequences, we can use only *y* in the transition testing phase instead of using both *x* and y (of state identifier H_2 = {*x*, *y*}) as in the HSI method. The same situation occurs for the transition $s_2 - x \to s_2$. Note that to test transitions we can use different distinguishing sequences. For example, to test transition $s_3 - y \to s_1$, it is sufficient to apply the sequence *yy* to distinguish state s_1 from all other states, where in the HSI-method we use both *x* and *yy* of the identifier H_1 = {*x*, *yy*}. To test transition $s_4 - x \to s_3$, we use *x* as the distinguishing sequence, and to test $s_4 - y \to s_4$ it is sufficient to apply *yyy* instead of *x* and *yy* of the identifier H_4 = {*x*, *yy*}. The obtained test suite TS_H = {*rxxy*, *rxyyy*, *ryxy*, *ryyxx*, *ryyyyyy*} is of total length 25 where,

Table 1
Summary of FSM-based methods.

Method	Fault detection power of a returned test suite	Specification FSM	Applicability	Remarks
W, Wp, UIOv	Test suite is complete	Reduced, complete	Always	Wp is an improvement to the W method, where state identifiers that are subsets of W are used in transition testing phase UIOv is a particular case of the Wp where each state identifier is a single sequence
UIO	Test suite is not always complete	Reduced, complete	When UIO exists for each state	No state identification phase
DS	Test suite is complete	Reduced, complete	When DS exists	Particular case of the W method, where W set has the only sequence DS
HSI, H	Test suite is complete	Reduced, possibly partial	Always	HSI is an improvement to the W method, where harmonized state identifiers are used in both state identification and transition testing H is an improvement to the HSI method, where harmonized state identifiers are derived on-the-fly

as described above, the Wp and HSI methods produce test suites of total length 29 and 41, respectively.

Intuitively, unlike the HSI method, the identification sequence of the ending state of the tested transition has to be harmonized only with other state identification sequences of the states of the state cover set Q and of the states reachable from the initial state by the prefix of the sequence. Moreover, for the same state of the specification we can use different state identification sequences when testing different incoming transitions to this state.

The following Table 1 includes a comparative summary of the FSM-based methods given in this paper.

4. Experimental evaluation

In this section, we experiment with the above described methods for the purposes described in the introduction and illustrated in details below. The experiments are based on randomly generated reduced specifications with a varying number of states (n) and inputs (k). Since state-oriented specifications of real systems may have somehow different characteristics than randomly generated FSM specifications, we perform experiments for two realistic protocols and compare the obtained experimental results.

The state identification facilities of the generated (reduced) specifications are derived (when possible) as follows: for every pair of different states of a given specification FSM M, we generate a shortest input sequence that distinguishes this pair [2,22]. The set of all obtained distinguishing sequences is a characterization set (W set) of 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 specification M is given in [34]. We derive UIO sequences (if exist) using the distinguishing tree algorithm given in [22].

Table 2 provides a comparison between the testing methods when $m = n$. Each row of Table 2 corresponds to a group of 50 randomly generated reduced specifications. For each of these specifications we use the W, Wp, HSI, UIOv and H methods to derive corresponding test suites. Moreover, we also derive test suites using the UIO and DS methods when the specification has 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–XII of Table 2, 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 of Table 2, respectively.

4.1. Test suite length

Fig. 2 depicts the average length of test suites and sorted according to Column V (number of transitions) of Table 2, derived using the testing methods when $m = n$. We note that the data are sorted according to the number of transitions. The reason is that when $m = n$ the worst-case complexity of length of obtained test suites is $O(kn^3)$ and thus, the complexity mainly depends on the number of transitions kn. According to Fig. 2, 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. The HSI and Wp test suites have 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 HSI method. The HSI and Wp methods return shorter suites than the W method, as expected. The UIOv did not perform better than the HSI and Wp methods; that also is expected as UIOv is a particular case of the Wp method.

Fig. 3 depicts the ratios of length of the test suites of the HSI(Wp), UIOv, H and UIO methods over the length of the W-based test suites for the (groups of) experiments depicted in rows 1–22 of Table 2. Length of the HSI(Wp) (UIOv, H, UIO, DS) test suites are on average 61 (64, 40, 28, 37) percent of length of those returned by the W method. According to these experiments these ratios, for all methods, 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 compared with the number of states (see for example rows 17 and 20 of Table 2), 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, length of UIOv test suites is 65% of length of the W test suites.

We note that according to the conducted experiments, for all except of the DS method, as the number of outputs increases, the average length of corresponding test suites decreases. However, as the number of outputs increases, the applicability (i.e. the number of FSMs which have a distinguishing sequence increases) of the DS method increases.

Table 3 provides a comparison between the HSI and H testing methods when $m > n$. In particular we experimented with $m = n + 1$ and $m = n + 2$. For $m > n + 2$ test suite length is huge and hard to obtain in practice. We note that we did not experiment with the Wp, W, and UIOv methods since, as shown above, the

Table 2
Summary of conducted experiments when $m = n$.

I-groups of 50 experiments	II-number of state, n	III-number of inputs, k	IV-number of outputs	V-no. of transitions ($n \times k$)	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 HSI 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)	1690 (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	11,012 (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	11,243 (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	13,144 (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	11,522 (14.09)	7005 (18.4)	7045 (4.19)	12,004 (27.7)	4426 (1.18)	2659 (13.8)	–	50	0
18	90	8	8	720	13,254 (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	15,129 (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	13,261 (18.35)	8031 (23.9)	8031 (5.52)	14,576 (37.1)	4976 (1.57)	3012 (17.6)	–	50	0
21	100	8	8	800	15,091 (26.61)	9332 (22.1)	9332 (8.43)	10,486 (23.1)	6020 (2.57)	3892 (14.3)	–	50	0
22	100	10	10	1000	17,204 (41.99)	10,503 (33.7)	10,503 (11.79)	9248 (27.7)	6880 (3.75)	4810 (20.0)	6602 (36.0)	50	7

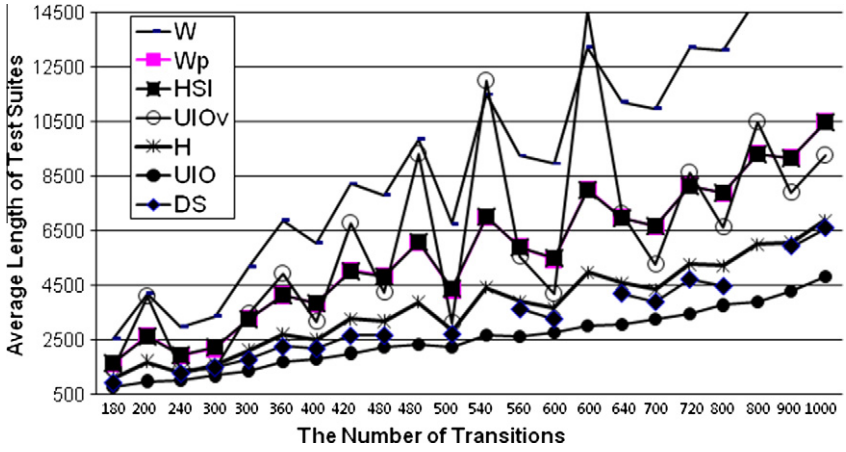


Fig. 2. Average length of the W, Wp, HSI, UIOv, H, UIO and DS test suites when $m = n$.

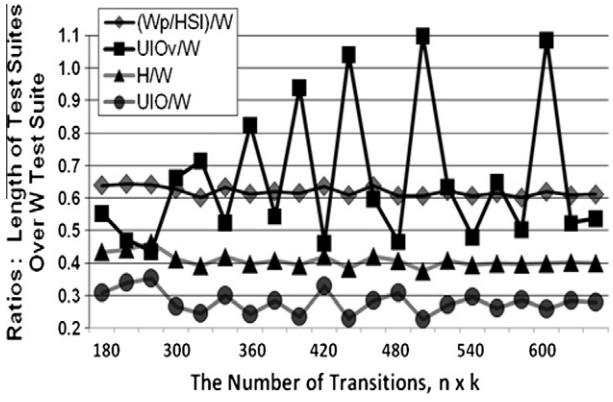


Fig. 3. Ratios of length of Wp (HSI), UIOv, H and UIO test suites over length of W test suite.

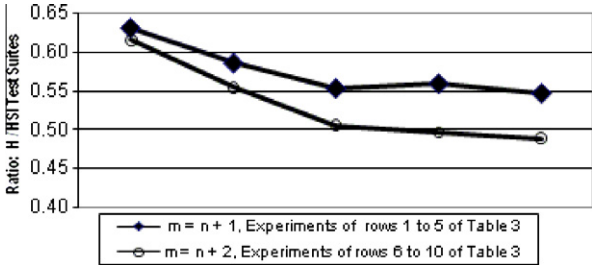


Fig. 4. Ratios of average length H/HSI test suites when $m = n + 1$ and $m = n + 2$.

HSI and Wp methods derive comparable length test suites that are shorter than those of the W and UIOv methods.

Fig. 4 depicts the ratios of length of the test suites of the H method over the length of the HSI-based test suites for the experiments depicted in rows 1–5 and rows 6–10 of Table 3. On average, the (length of) H test suites are 55% of the (length of) HSI test suites. Moreover, according to the experiments this ratio slightly decreases as the size of the specification increases and/or as the upper bound of the number of states of an implementation increases. We note that similar results are obtained when $m = n$. According to Table 2, on average, the H test suites are 60% of the HSI test suites.

In this paragraph, we compare length of obtained test suites with corresponding estimated worst-case upper bounds reported in [1,16,34,35]. According to the conducted experiments, when $m = n$, on average, the length of the test suites derived by the W, HSI and UIO, H, UIOv and DS, is only 0.005, 0.003, 0.002, and 0.001 of the worst-case length. Moreover, all methods have length of the order cn^2, where the constant c approximately equals two for the W method, 1.6 for the Wp and HSI, 1.5 for the UIOv and 1.1 for the H, UIO, and DS methods. We also experimented how long are UIO sequences (when those exist). According to the obtained results, the average length of UIO sequences for randomly generated FSMs equals two and is independent of the size of the specification. When $m > n$, according to Table 3, on average, the length of the test suites derived by the HSI and H methods are 0.034 and 0.02 of the length of the corresponding worst-case test suites.

According to the conducted experiments, on average, test suite derivation times of the Wp, HSI, UIOv, H, UIO, and DS methods are

Table 3
A summary of experiments for the case when number of inputs/outputs equals four and $m = n + 1$ and $m = n + 2$.

I-groups of 50 experiments	II-number of states of the specification	III-number of states of an implementation	IV-number of transitions	V-average length HSI test suites	VI-average length H test suites
1	10	11	40	1399	883
2	20	21	80	3773	2212
3	30	31	120	6432	3560
4	40	41	160	9272	5188
5	50	51	200	12,494	6837
6	10	12	40	6790	4181
7	20	22	80	17,238	9565
8	30	32	120	29,860	15,115
9	40	42	160	44,137	21,919
10	50	52	200	58,949	28,813

97%, 32%, 72%, 10%, 52%, and 21% of the test suite derivation time of the W method.

4.2. Applicability of UIO and DS methods

Column XIII of Table 2 shows how many times (out of the 50) the UIO method is applicable. On average, the UIO method is applicable to 99% of all conducted experiments and it seems that its applicability for randomly generated FSMs is independent of the size of the specification.

Column XIV of Table 2 shows how many times (out of the 50) the DS is applicable when the numbers of inputs equals the number of outputs. On average, 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 states of the specifications decreases. For example, according to Table 2, 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. However, this applicability drops to 14% when the ratio is between 0.1 and 0.2. Moreover, the DS seems to be always applicable when the ration out/n is greater than 0.5.

Knowing that the UIO method returns (when applicable) shorter test suites than all other methods, we conducted experiments to have an idea about the fault detection power of these test suites. We consider in our experiments small size specifications $n = (2–6)$ and two inputs ($k = 2$) and outputs, since explicit enumeration is only possible for small size implementations. For each considered combination (n, k), we derive 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 implementations of the specification. Then, we determine how many of faulty implementations are killed (detected) by the derived test suite. According to the conducted experiments, on average, when applicable, 41% of the UIO test suites are incomplete.

5. Case studies with realistic protocols

This section includes the application of the different test generation methods to the *simple connection protocol* (*SCP*) [37] and the V.76 protocol [38]. We use the specification and description language (SDL) [40] to describe the protocol specifications. SDL is based on the semantic model of extended finite state machines (EFSMs) and is used to specify the functional aspects of the system behavior. We unfold the SDL EFSM description of *SCP* and V.76 protocols and obtain corresponding equivalent (with the same set of traces) reduced FSMs. Then, we apply (when applicable) the test derivation methods to the two obtained FSMs.

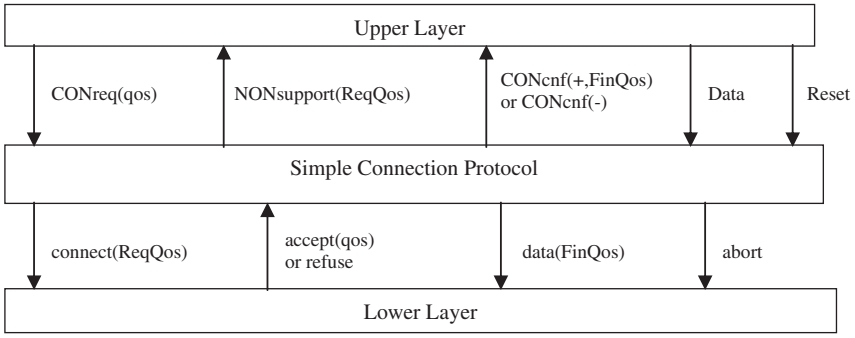
5.1. The SCP protocol

The *SCP* allows to connect an entity called the *upper layer* with the entity called the *lower layer*. Fig. 5 depicts the *SCP* protocol messages. 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.

Given the SDL specification of the *SCP* protocol, the corresponding FSM has $n = 26$ states, 11 inputs, 12 outputs, and 286 transitions. We apply the W, Wp, HSI, and H test derivation methods to the obtained FSM and we get test suites of length 24,490, 5229, 6125, and 4223, respectively. The Wp (HSI, H) test suites are 21 (25, 17) percent of those of the W method. These results are slightly better than those given in Table 2 (for randomly generated machines), where the Wp (HSI, H) test suites are 60 (60, 40) percent of those of the W. Moreover, similar to the experiments reported in Table 2, the *SCP* W, Wp, HSI, and H test suites have length of the order cn^2, where the constant c equals 36, 7, 9, and 6, respectively. We note that the UIO, UIOv, and DS methods are not applied to the *SCP* protocol since the corresponding FSM does not have a DS and UIOs.

5.2. The V.76 protocol

In this subsection, we apply the test derivation methods to the specification of a simplified version of the ITU-T V.76 Recommendation [38] protocol based on link access procedure for modems [32]. The V.76 protocol allows to establish data link connections (DLCs) between two modems and to transfer data over these connections. Fig. 6 illustrates the communication between two service users (SUs), the layer on top of V.76, using primitives and frames.



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].

Fig. 5. SCP protocol messages.

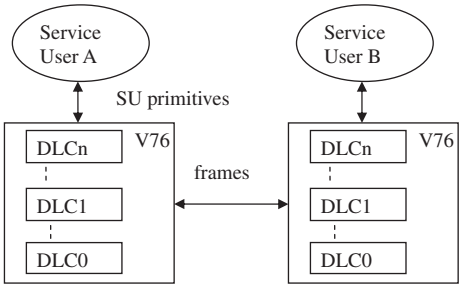


Fig. 6. Data link connections between SUA and SUB using the V.76 protocol.

Several connections may exist in parallel. For example, SUA may establish DLC0 to transmit voice and DLC1 to transfer the data to/from SUB. In order to better understand the protocol, we illustrate its behavior using four message sequence charts (MSCs) [39] and an SDL model [40].

Fig. 7 describes four phases of the protocol. In the first phase, Fig. 7a, the SUs perform an exchange of identification procedure (XID transmission) that is initiated by DLCa. Phase 2, Fig. 7b, illustrates the establishment of a data link connection to transmit user data (through the SUs) that transfers the protocol from a *disconnected* to a *connected* state. On receipt of an L-ESTABLISH request primitive (L_EstbReq) from its SU, the V.76 attempts to establish the DLC. The DLC entity transmits a set asynchronous balanced mode extended (SABME) frame. Then, a DLC entity receiving a SABME and being able to establish the DLC, has to respond with a unnumbered acknowledge (UA) primitive and then the protocol transfers to the *connected* state. If the entity was unable to establish the DLC, SUB responds with an L-RELEASE primitive (L_ReleaseReq) and DLCb transmits a disconnect mode (DM) command. In all cases, DLCa transfers the received information to its SUA. Once in the *connected state*, transferring of information may start. DLC receives data from SU with the use of an L-DATA request primitive (L_DataReq) as shown in Fig. 8c. The data is transmitted using an I frame. At the last phase, the data link connection is ended after a SU initiates an L-RELEASE request primitive (Fig. 8d). The SDL structures of some of the above messages are shown in details in Fig. 7.

We model the protocol using the SDL specification shown in Fig. 8. In our case, the highest level of abstraction of this specification describes the SUs, and the lowest includes the specification of the primitives, tasks, etc. The functional aspects of the system are described using architectural and behavioral properties. The former denotes the connection and the organization of the architectural elements (block, processes, etc.) with the environment and amongst themselves. The latter denotes states, states transitions, and the interactions between the entities and the environment, represented as EFSMs. Actions associated with transitions include verification based on local variables (that can impose conditions, predicates, to move to the next state), the execution of tasks (assignment or informal text), procedure calls, dynamic creation of processes (SDL contains the concepts of "type" and "instance of type"), arming and disarming timers, etc.

The V.76 specification contains three main blocks. Two of these are instantiated from a block type and the third consists of the medium between the two DLCs. The SDL system environment denotes the SUs behaviors and provides the SUs' primitives.

In order to apply the test derivation methods, we unfold the EFSM description of the V.76 protocol and obtain an equivalent reduced FSM. The obtained FSM has $n = 195$ states, 13 inputs, 5 outputs, 2535 transitions, and state cover set W (that consists of a shortest distinguishing sequence of each pair of states of the specification FSM) of 112 sequences with total length equals 504. We apply the Wp, HSI, and H test derivation methods to the obtained FSM and we get test suites of length 614,725, 354,293, and 196,192, respectively. Unlike previous experiments, the HSI test suite is almost twice shorter than that of the Wp. The reason for this is that the distinguishing sequences in the W set are longer than the state identifiers and the tail sequences of these distinguishing sequences are not prefixes of the state identifiers that test the following transitions. Therefore, the Wp-based test suite, which uses the W set at the state identification phase, has many

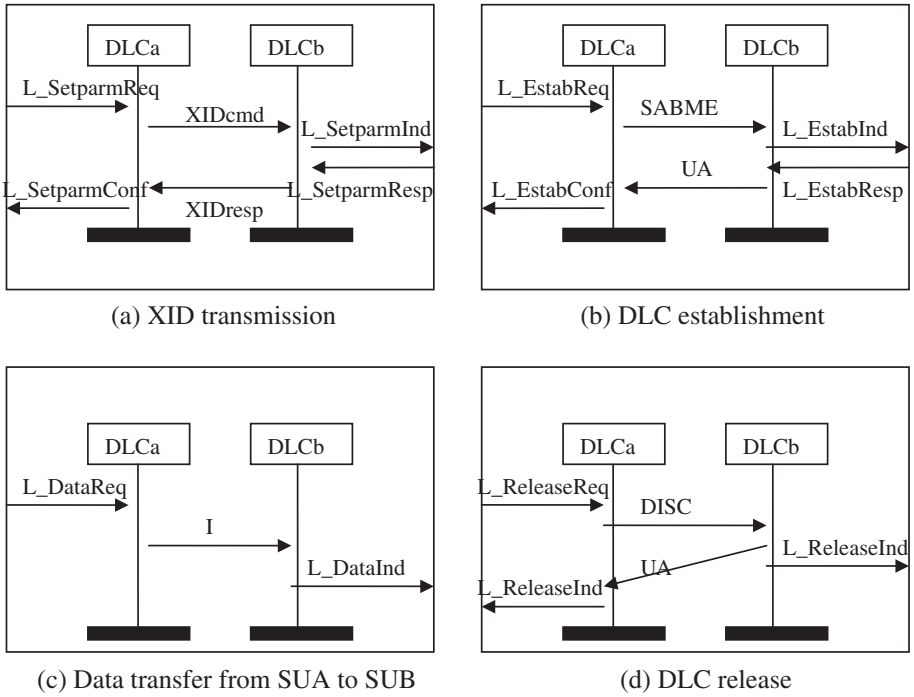


Fig. 7. Behavior of the V.76 protocol in four phases.

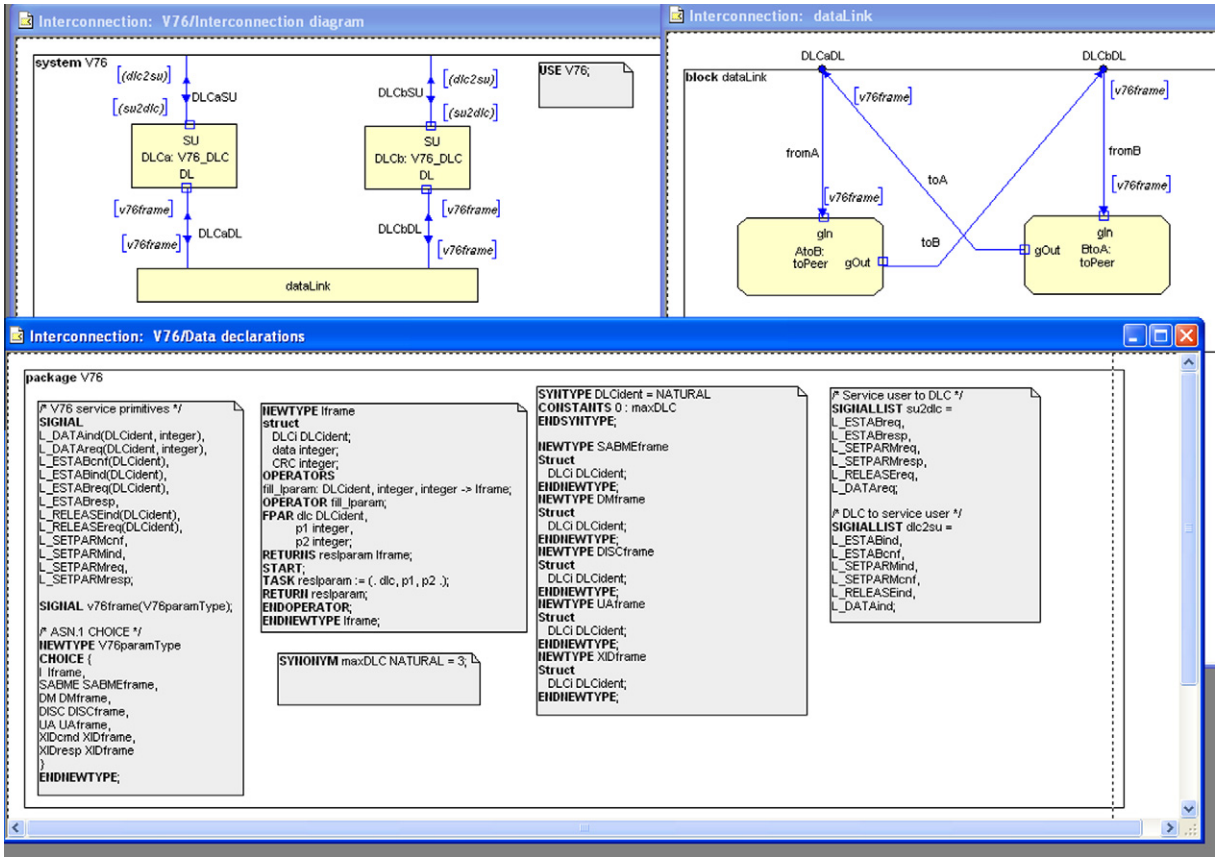


Fig. 8. SDL specification of the V76 protocol.

sequences that are not necessary for checking the ending states of transitions than the HSI-based test suite which uses a set of harmonized state identifiers at the state identification phase. Moreover, we observe that the H test suite is 55% of the HSI. This result is slightly better than the results reported in Tables 2 and 3 (for randomly generated specifications) and for the SCP protocol, where the H test suite is 60%, 60%, and 55% of the HSI test suites.

We also mention that for the SCP protocol, the H based test suite was generated 10 times faster than the Wp and HSI-based test suites. For the V.76, Wp, HSI, and H test suites have length of the order cn^2, where the constant c equals 16, 9, and 5, respectively. We note that the W method algorithm did not produce a test suite even after running the algorithm for several hours. The UIO and DS methods are not applied to the V.76 protocol since the corresponding FSM does not have a DS and UIOs.

6. Conclusion and summary

In this paper, we presented an overview and detailed experiments with the W, Wp, HSI, H, UIOv, UIO, and DS FSM-based conformance testing methods. The experiments with randomly generated (reduced) specifications, when the specification and implementation are assumed to have equal number of states, i.e. when $m = n$, show that HSI(Wp) (UIOv, H, UIO, DS) test suites are on average 61 (64, 40, 28, 37) percent of those returned by the W method. According to these experiments these ratios, for all except the UIOv, are almost independent of the size of the specification. The W test suites are significantly longer than those derived by all other methods. The length of the Wp and HSI 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, HSI, and UIOv methods. However, the DS method is applicable only to 19% of all conducted experiments when the number of outputs equals the number of inputs and this number is rather small compared to the number of states. The applicability of the DS method increases when the number of outputs *out* increases.

The experiments with the methods that always return complete test suites and that are always applicable to complete reduced specifications, i.e. the W, Wp, HSI, and H methods, show that the H method outperforms the other methods and the HSI and Wp suites are of comparable length. The H test suites are on average 40 (60) percent of the W(HSI) test suites. The experiments conducted when $m > n$ show that the H test suites are on average 55% of those returned by the HSI method.

The experiments done with the *SCP* protocol show that the Wp, (HSI, UIOv) test suites are on average 21 (25, 17) percent of those returned by the W method. These results show better performance, than randomly generated specifications, of the Wp, HSI, and UIOv methods in comparison with the W method. Also, the results show that the H test suite is 55 (80) percent of the HSI(Wp) test suite. The experiments with the V.76 protocol show that the H test suite is 55% of the HSI(Wp) test suite. The UIOv, UIO, and DS methods are not applicable to the *SCP* and V.76 protocols. Also, the W based algorithm did not produce a test suite even after running for several hours.

In practice, when $m = n$, according to the conducted experiments, on average, the length of the test suites derived by the W,

HSI and UIOv, H, UIO and DS, is only 0.005, 0.003, 0.002, and 0.001 of the theoretic worst-case length. Moreover, all methods generate test suites with length of the order cn^2, where the constant c approximately equals 2 for the W method, 1.6 for the Wp and HSI, 1.5 for the UIOv and 1.1 for the H, UIO, and DS methods. For the *SCP* protocol, c equals 36 for the W method, 7 for the Wp, 9 for the HSI, and 6 for the H method. For the V.76 protocol, c equals 16 for the Wp, 9 for the HSI, and 5 for the H method. When $m > n$, the length of the test suites derived by the HSI and H methods are 0.032 and 0.02 of the length of the corresponding theoretic worst-case test suites. For the two considered realistic protocols, the outperformance of the H method in comparison with the HSI method is bigger than that of randomly generated FSM specifications.

As the HSI and H methods are the only methods that can always be applied to partial reduced specifications, it would be reasonable to experimentally compare these methods using partial randomly generated FSM specifications. In addition, it also would be beneficial to conduct experiments with randomly generated FSMs which have special properties, such as, FSMs with long state identifiers, FSMs with special topological structure, etc.

In the past few years many methods are presented for test derivation from non-deterministic FSMs and for FSM-based testing of embedded components. As an extension of the work presented in this paper, it would be useful to experimentally evaluate these methods. In addition, it would be useful to assess the applicability of the FSM-based methods to new application areas such as GUI and security testing.

References

[1] T.S. Chow, Test design modeled by finite-state machines, IEEE Trans. Softw. Eng. 4 (3) (1978) 178–187.
[2] M.P. Vasilevskii, in: Failure Diagnosis of Automata, Translated from Kibernetika, vol. 4, 1973, pp. 98–108.
[3] S. Fujiwara, G.v. Bochmann, F. Khendek, M. Amalou, A. Ghedamsi, Test selection based on finite state models, IEEE Trans. Softw. Eng. 17 (6) (1991) 591–603.
[4] K. Sabnani, A. Dahbura, A protocol test generation procedure, Comput. Networks ISDN Syst. 15 (4) (1988) 285–297.
[5] S.T. Vuong, W.W.L. Chan, M.R. Ito, The UIOv-method for protocol test sequence generation, in: Proc. of the IFIP TC6 2nd IWPTS, North-Holland, 1989, pp. 161–175.
[6] G. Gonenc, A method for the design of fault detection experiments, IEEE Trans. Comput. C-19 (6) (1970) 551–558.
[7] A. Petrenko, N. Yevtushenko, Testing from partial deterministic FSM specifications, IEEE Trans. Comput. 54 (9) (2005) 1154–1165.
[8] A. Petrenko, N. Yevtushenko, A. Lebedev, A. Das, Nondeterministic state machines in protocol conformance testing, in: Proc. of the IFIP 6th IWPTS, France, 1993, pp. 363–378.
[9] N. Yevtushenko, A. Petrenko, Test derivation method for an arbitrary deterministic automaton, Automatic Control and Computer Sciences, vol. 5, Allerton Press Inc., USA, 1990.
[10] R. Dorofeeva, K. El-Fakih, N. Yevtushenko, An improved FSM-based conformance testing method, in: Proc. of the IFIP 25th International Conference on Formal Methods for Networked and Distributed Systems, vol. 3731, Taiwan, Lecture Notes in Computer Science, 2005, pp. 204–218.
[11] I. Koufareva, R. Dorofeeva, A novel modification of W-method, joint bulletin of the Novosibirsk computing center and A.P. Ershov institute of informatics systems, Ser.: Comput. Sci. 18 (2002) 69–81.
[12] G.v. Bochmann, A. Petrenko, Protocol testing: review of methods and relevance for software testing, in: Proc. International Symposium on Software Testing and Analysis, Seattle, 1994, pp. 109–123.
[13] S.T. Chanson, A.A.F. Loureiro, S.T. Vuong, On tools supporting the use of formal description techniques in protocol development, Comput. Networks ISDN Syst. 25 (1993) 1–5.
[14] D. Lee, M. Yannakakis, Principles and methods of testing finite state machines – a survey, Proc. IEEE 84 (8) (1996) 1090–1123.
[15] D.P. Sidhu, T.K. Leung, Formal methods for protocol testing: a detailed study, IEEE Trans. Softw. Eng. 15 (4) (1989) 413–426.
[16] M. Yannakakis, D. Lee, Testing finite state machines: fault detection, J. Comput. Syst. Sci. 50 (1995) 209–227.
[17] R. Lai, A survey of communication protocol testing, J. Syst. Softw. 62 (2002) 21–46.
[18] I. Bourdonov, A. Kossatchev, A. Petrenko, D. Galter, KVEST: automated generation of test suites from formal specifications, Lect. Notes Comput. Sci. 1708 (1999) 608–621.
[19] 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.
[20] E.P. Hsieh, Checking experiments for sequential machines, IEEE Trans. Comput. 20 (10) (1971) 1152–1166.
[21] H. Ural, X. Wu, F. Zhang, On minimizing the lengths of checking experiments, IEEE Trans. Comput. 46 (1) (1997) 93–99.
[22] A. Gill, Introduction to the Theory of Finite-state Machines, McGraw-Hill, 1962.
[23] A. Dahbura, K. Sabnani, Experience in estimating fault coverage of a protocol test, in: Proc. INFOCOM, 1988, pp. 71–79.
[24] M. Yao, A. Petrenko, G.v. Bochmann, Fault coverage analysis in respect to an FSM specification, in: Proc. INFOCOM, 1994.
[25] V. Shevelkov, Development of methods and tools for session interconnection provision in open systems networks, Ph.D. Thesis, Riga, 1991.
[26] D.P. Sidhu, T.K. Leung, Experience with test generation for real protocols, in: Proc. SIGCOMM Symp.: Communication Architectures and Protocols, 1988, pp. 257–261.
[27] D.P. Sidhu, T.P. Blumer, Verification of NBS class 4 transport protocol, IEEE Trans. Softw. Eng. 34 (1986) 781–789.
[28] F. Liu, Test generation based on the FSM model with timers and counters. M.Sc. Thesis, Université de Montréal, DIRO, 1993.
[29] A. Petrenko, Checking experiments with protocol machines, in: Proc. 4th Int. Workshop on Protocol Test Systems (IWPTS), 1991, pp. 83–94.
[30] A.V. Aho et al., An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours, in: Proc. Protocol Specification, Testing, and Verification (PSTV), 1988, pp. 75–86.
[31] M.H. Sherif, M. Uyar, Protocol modeling for conformance testing: case study for the ISDN LAPD protocol, AT&T Tech. J. 69 (1) (1990) 60–83.
[32] ITU-T Recommendation V.42, Error-correcting Procedures for DCEs Using Asynchronous-to-synchronous Conversion, AAP29-03/02, 2002.
[33] R. Dorofeeva, K. El-Fakih, S. Maag, A.R. Cavalli, N. Yevtushenko, Experimental evaluation of FSM-based testing methods. in: Proc. of the IEEE International Conference on Software Engineering and Formal Methods (SEFM), Germany, 2005, pp. 23–32.
[34] Z. Kohavi, Switching and Finite Automata Theory, McGraw-Hill, New York, 1978.
[35] D. Lee, M. Yannakakis, Testing finite state machines: state identification and verification, IEEE Trans. Comput. 43 (3) (1994) 306–320.
[36] K. Naik, Efficient computation of unique input/output sequences in finite-state machines, IEEE Trans. Network 5 (4) (1997) 585–599.
[37] W.H. Chen, Executable test sequence for the protocol data flow property. in: Proc. of Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification, FORTE/PSTV, Cheju Island, Korea, 2001, pp. 285–299.
[38] ITU-T Recommendation V.76, Generic Multiplexer Using V.42 LAPM-Based Procedures. Series V: Data Communication Over the Telephone Network, 1996.
[39] ITU-T Recommendation Z.120, Annexe B: Algebraic Semantics of Message Sequence Charts, Geneva, 1995.
[40] ITU-T, Recommendation Z, 100-Specification and Description Language SDL, Geneva, 1992.
[41] A. Mathur, Foundations of Software Testing, Addison Wesley, 2008.
[42] K. Naik, P. Tripathy, Software Testing and Quality Assurance: Theory and Practice, Wiley, 2008.