ISSN 0361-7688, Programming and Computer Software, 2012, Vol. 38, No. 4, pp. 201–209. © Pleiades Publishing, Ltd., 2012.

FSM-Based Testing from User Defined Faults Adapted to Incremental and Mutation Testing[1]

K. A. El-Fakih[a], R. Dorofeeva[b], N. V. Yevtushenko[b], and G. V. Bochmann[c]

[a] *Department of Computer Science and Engineering, American University of Sharjah, Sharjah, UAE, PO Box: 26666*

[b] *Tomsk State University, 36 Lenin Str., Tomsk, 634050 Russia*

[c] *University of Ottawa, Ont., K1N 6N5, Canada*

e-mail: kelfakih@aus.edu, drf@kitidis.tsu.ru, ninayevtushenko@yahoo.com, bochmann@site.uottawa.ca

Received January 15, 2012

Abstract—We study the problem of deriving a test suite with guaranteed fault coverage from a given finite state machine specification with respect to some given user defined faults. We consider the case when an implementation under test can have more states than its specification while user defined faults are implemented in an arbitrary way. We show that our approach can be used for FSM-based incremental and mutation testing and correspondingly we investigate cases that can be used for reducing length of obtained test suites. In some cases, worst-case length of obtained test suite becomes polynomial. Experiments show significant gains is using our approach in comparison to testing the whole specification.

Keywords: conformance testing, finite state machines, model-based testing, mutation testing, incremental testing.

DOI: 10.1134/S0361768812040019

1. INTRODUCTION

In various application domains the functional behavior (or the specification) of a system can be represented as a finite state machine (FSM). FSMs are also used as the underlying models for formal description techniques, such as SDL and UML State Diagrams. An FSM has finite number of states and a finite number of transitions labeled by pairs of input and output symbols. If the initial state of the specification is known then the behavior can be described as the mapping of input sequences (sequences of input symbols) that can be applied to a machine into their corresponding output sequences (sequences of output symbols).

In FSM-based conformance testing, one usually assumes that not only the specification, but also an implementation under test (IUT) can be modeled as an FSM. In order to check if a given IUT conforms to its specification, a test suite (*TS*) of input/outputs pairs of test sequences (*test cases*) is derived from the given specification. The input sequences of a test suite are then applied to an IUT and outputs generated (observed) by the IUT are compared with expected outputs. If the outputs do not match, then the IUT has a fault. When a test suite should have the so-called guaranteed fault coverage appropriate types of implementation faults are described by a set of implementation FSMs, usually called a *fault domain*. A test suite is *complete* if it detects every faulty implementation of the considered fault domain.

Many conformance test derivation methods have been developed for deriving tests when the system specification and IUT can be represented as deterministic FSMs. For related surveys see (Bochmann and Petrenko, 1994), (Lee and Yannakakis, 1996), (Dorofeeva et al., 2005b), (Dorofeeva et al., 2010). Well-known methods are called the W (Vasilevskii, 1973), (Chow, 1978), Wp (Fujiwara et al., 1991), UIOv (Vuong et al., 1989), HSI (Petrenko et al., 1993), (Petrenko and Yevtushenko, 2005), (Yevtushenko and Petrenko, 1990), H (Dorofeeva et al., 2005a), (Koufareva et al., 2002), and SPY (Simoa et al., 2009). All these methods guarantee the following fault coverage: given a deterministic reduced (minimal) specification FSM $\mathcal{S}$ with n states, a test suite *TS* derived from $\mathcal{S}$ by any of these methods, detects (kills) every non-equivalent FSM IUT of $\mathcal{S}$ provided an upper bound m on the number of states of an IUT is given. It is usually assumed that m equals to or is greater than n. The worst-case length of a test suite generated by the above methods is of the order $O(k^{m-n+1}n^3)$, where k is the number of inputs of an FSM, thus, unfortunately, when $m > n$ a test suite can be of an exponential length.

Given the possibly non-reduced deterministic FSM specification $\mathcal{S}$, user defined faults are described as a set of suspicious transitions which can be faulty in an implementation; the obtained non-deterministic FSM is called *fault function* in Petrenko and Yevtush-

[1] The article is published in the original.

enko (1992) or a *mutation machine* in (Koufareva et al., 1999). Test derivation from user defined faults involves the generation of a test suite that guarantees the detection of any faulty IUT that is not equivalent to $\mathcal{S}$ and which is a deterministic submachine of the considered mutation machine. The method in (Petrenko and Yevtushenko, 1992) considers test derivation when the specification and an IUT have the same number of states, i.e. $m = n$, and the method in (Koufareva et al., 1999) considers the case when an IUT can have more states than its specification, i.e. when $m \geq n$. However, both methods rely on the assumption that for each transition of the specification FSM the number of faults defined by a user in the corresponding mutation machine is rather small. However, this is not the case in some application areas such as in FSM-based incremental testing of modified specifications and their (modified) implementations (El-Fakih et al., 2004). Accordingly, in this paper, we consider applications areas where suspicious transitions can be implemented in an arbitrary way, i.e., suspicious transitions are chaotic transitions in the mutation machine. In this case a mutation machine has only two types of transitions, namely, *deterministic* and *chaotic* transitions. Deterministic transitions correspond to non-suspicious transitions while chaotic transitions correspond to suspicious transitions. Several proper cases can be considered for shortening length of obtained test suites when considering such mutation machines. In some cases, worst-case length of an obtained test suite becomes polynomial. More precisely, we first generalize the incremental testing problem given in (El-Fakih et al., 2004) for the case an IUT can have more states than its specification and present an algorithm that determines which states of a mutation machine have to be identified in an IUT. We derive appropriate state identification tests for these states by utilizing properties of the mutation machine inherited from the specification. In addition, we identify the transitions of the mutation machine that should be checked in an IUT and derive corresponding transition testing tests. Several cases are studied for shortening a resulting test suite.

In this paper, we also consider how our approach can be applied to FSM-based mutation testing. Mutation testing is fault-based testing technique that has been well investigated in the past thirty years. The reader may refer to Jia and Harman (2006) for a comprehensive survey on mutation testing including model-based mutation testing. Fabbri et al. (1994, 1999) applied mutation analysis for FSMs. Relevant mutation operators are introduced and benefits of using 2-order mutants are reported. Hierons and Merayo (2009) consider mutation testing from probabilistic and stochastic FSMs. Maldonado et al. (2004) and De Souza et al. (2000) consider mutation testing from SDL and Estelle specifications, respectively. Unfortunately, the number of possible mutants of an FSM is huge, and accordingly, in this paper, we propose an FSM-based mutation testing approach that uses a compact representation of many mutants, namely, a mutation machine, and then derive a complete test suite for all considered mutants without the explicit enumeration of mutants. In addition, we discuss how to derive the compact representation in such a way that a corresponding complete test suite is of polynomial length. However, in general, when FSM specification $\mathcal{S}$ is derived from a given code, the specification can be non-reduced; and thus, mutants derived from the specification can have more states than the reduced form of the specification. In this context, the method presented in this paper can be used for FSM-based mutation testing.

In order to get a feeling about the length of obtained test suites in practical situations, we have experimented with the testing method proposed in this paper. According to the conducted experiments, when the number of chaotic transitions represent up to 5, 10, 15, 20, and 30% of the mutation machine, on average, the length of obtained test suites are 66, 14, 24, 36, and 87 (10, 20, 30, 50, and 14) percent of the corresponding test suites derived for the whole specification when $m = n + 1$ ($m = n + 2$).

This paper is organized as follows. Section 2 includes notations and definitions used in the paper. Section 3 introduces the incremental and mutation testing problems in the context of test derivation from a mutation machine which includes only deterministic and chaotic transitions. In particular, we show how a mutation machine with only deterministic and chaotic transitions can be constructed for FSM-based incremental and mutation testing. Section 4 includes our test derivation algorithm illustrated through a simple working example. Section 5 includes experimental results and Section 6 concludes the paper.

2. PRELIMINARIES

A *finite state machine* (an FSM or a machine) is a 5-tuple $\mathcal{S} = (S, X, Y, h_S, s_1)$ (Starke, 1972), where S is a finite set of states, s_1 is the *initial state*, $X(Y)$ is a finite set of input (output) symbols, and h_S: $S \times X \longrightarrow 2^{S \times Y} \backslash \varnothing$ is a behavior function where $2^{S \times Y}$ is the set of all subsets of the set $S \times Y$. Given present state s_i and input symbol x, each pair $(s_j, y) \in h(s_i, x)$ represents a possible transition from state s_i under the input x to the next state s_j with the output y. A transition from state s_i under input x is *deterministic* if $|h_S(s_i, x)| = 1$. The transition from state s_i under input x is *chaotic* if $h_S(s_i, x) = S \times Y$. If each transition is deterministic then FSM $\mathcal{S}$ is said to be *deterministic*. In the deterministic FSM $\mathcal{S}$ instead of the behavior function h_S we use two functions, a transition function δ_S: $S \times X \longrightarrow S$ and an output function λ_S: $S \times X \longrightarrow Y$. An FSM $\mathcal{T} = (T, X, Y, h_T, t_1)$ is a *submachine* of $\mathcal{S} = (S, X, Y, h_S, s_1)$ if $T \subseteq S$, $t_1 = s_1$ and for each $(t, x) \in T \times X$ it holds that $h_T(t, x) \subseteq$

$h_S(t, x)$. The set of all deterministic complete submachines of $\mathcal{S}$ is denoted $Sub(\mathcal{S})$. As usual, the behavior function h_S can be extended to the set X^* of finite input sequences. Given state $s \in S$ and input sequence $\alpha = x_1x_2...x_k \in X^*$, the pair $(s_{k+1}, y_1y_2...y_k) \in h_S(s, \alpha)$, $y_1y_2...y_k \in Y^*$, if there exist states $s_1 = s, s_2, ..., s_k, s_{k+1}$ such that $(s_{j+1}, o_j) \in h_S(s_j, i_j), j = 1, ..., k$. The set $out(s, \alpha) = \{\gamma | \exists s' \in S[(s', \gamma) \in h_S(s, \alpha)]\}$ denotes the *output projection* of h_S, while the set $next\_state(s, \alpha) = \{s' | \exists \gamma \in Y^*[(s', \gamma) \in h_S(s, \alpha)]\}$ denotes the *state projection* of h. An FSM is *connected* if each state is reachable from the initial state. As usual, without loss of generality, we consider only connected FSMs. A state s_i is said *deterministically reachable* (or *d*-reachable) from the initial state if the state can be reached from the initial state through deterministic transitions. Let $\mathcal{S} = (S, X, Y, h_S, s_1)$ and $\mathcal{T} = (T, X, Y, h_T, t_1)$ be two FSMs. A set Q of input sequences is called a *state cover set* of $\mathcal{S}$ if for each state s_i of S, there is an input sequence $\alpha_i \in Q$ that takes $\mathcal{S}$ to state s_i from the initial state. We further consider prefix-closed state cover sets. A prefix closed cover set exists for each connected FSM. Given two complete deterministic FSMs $\mathcal{S}$ and $\mathcal{T}$, states s_j of $\mathcal{S}$ and t_i of $\mathcal{T}$ are *equivalent* (Gill, 1962) if $\forall \alpha \in X^*$ it holds that $out_S(s_j, \alpha) = out_T(t_i, \alpha)$. Otherwise, we say that states s_i and t_j are *distinguishable*. An input sequence α such that $out_S(s_j, \alpha) \neq out_T(t_i, \alpha)$ *distinguishes* the states s_j and t_i. The set V of input sequences deterministically (*d*-) *distinguishes* states t_i and s_j if there exists a sequence $\alpha \in V$ that *distinguishes* t_i and s_j and traverses only deterministic transitions when applied at these states.

A deterministic FSM is *reduced* if its states are pairwise distinguishable. Given a reduced FSM $\mathcal{S}$ and a state $s_j \in S$, a set W_j of input sequences is called a *state identifier* of state s_j if for every other state $s_i \in S$ there exists $\alpha \in W_j$ such that $\lambda_S(s_j, \alpha) \neq \lambda_S(s_i, \alpha)$. A *separating family* (Yannakakis and Lee, 1995) or a family of *harmonized state identifiers* (Yevtushenko and Petrenko, 1990; Petrenko, 1991) is a collection of state identifiers W_j which satisfy the following 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 a common prefix α such that $\lambda_S(s_j, \alpha) \neq \lambda_S(s_i, \alpha)$. A separating family is known to exist for any reduced machine. Complete FSMs $\mathcal{S}$ and $\mathcal{T}$ are *equivalent*, written $\mathcal{S} \cong \mathcal{T}$, (*distinguishable*) if their initial states are equivalent (distinguishable). If FSMs $\mathcal{S}$ and $\mathcal{T}$ are distinguishable then an input sequence α that distinguishes their initial states is said to *distinguish* the machines $\mathcal{S}$ and $\mathcal{T}$. We say that an implementation $\mathcal{T}$ *conforms to* the specification $\mathcal{S}$ if $\mathcal{T}$ is equivalent to $\mathcal{S}$.

Given a complete deterministic specification FSM $\mathcal{S} = (S, X, Y, \delta_S, \lambda_S, s_1)$, a complete FSM $\mathcal{M} = (M, X, Y, h_M, m_1)$ is called a *mutation machine* (for $\mathcal{S}$) if $\mathcal{S}$ is equivalent to a submachine of $\mathcal{M}$. Let $Sub(\mathcal{M})$ denote the set of all complete deterministic submachines of $\mathcal{M}$. A *test suite* is a finite set of finite input sequences (called *tests* or *test cases*) of the specification FSM. Similar to FSM-based testing methods, we assume that a reset function (hereafter written "*r*") is available that allows the reliable reset of the implementation under test. This implies that each test case of the test suite starts with the reset operation. A test suite TS is *complete* w.r.t. the given $\mathcal{M}$, also called *complete* w.r.t. the fault model $(\mathcal{S}, \cong, Sub(\mathcal{M}))$, if for each complete deterministic submachine $\mathcal{P}$ of $\mathcal{M}$ that is distinguishable from $\mathcal{S}$ there exists a sequence $\alpha \in TS$ that distinguishes machines $\mathcal{S}$ and $\mathcal{P}$. In this paper, we consider mutation machines such that for each state and each input, the transition from the state under the input is either deterministic or chaotic. In the following section, we discuss two application problems where such mutation machines are of practical use.

3. DERIVING MUTATION MACHINES FOR INCREMENTAL AND MUTATION TESTING

Given a non-reduced deterministic specification machine $\mathcal{S}$, with n states, and a non-deterministic possibly non-reduced mutation machine $\mathcal{M}$ with m states, where $m \geq n$, testing from given user defined faults involves the derivation of a test suite that is complete w.r.t. to the fault model $(\mathcal{S}, \cong, Sub(\mathcal{M}))$ where the mutation machine is usually derived by augmenting the given specification machine $\mathcal{S}$ with a set of user defined faults. If a mutation machine has only two types of transitions, namely, deterministic or chaotic, then several cases can be used for shortening a resulting test suite, and in some cases (worst-case) length of a test suite becomes polynomial. In this section, we show that such mutation machines can be derived in the context of incremental and mutation testing, and thus, those application problems fit well our test derivation approach. Our test derivation method is presented in Section 4.

3.1. Mutation Machine for Incremental Testing

Given a reduced IUT and its corresponding system specification, a designer may change (or modify) the specification and its corresponding IUT based on changes in the user requirements. Incremental testing deals with the derivation of tests for checking that the modified parts of a modified specification are correctly implemented in a modified IUT (El-Fakih et al., 2004). In this paper we consider the general case when an initial IUT can be non-reduced, that is, when an IUT can have more states than its specification.

Given a reduced specification FSM $\mathcal{S}$ with m states, we assume that an IUT $\mathcal{T}$ with m states, the IUT has been tested and found to be conforming (i.e.,

equivalent) to its specification. The modified specification is obtained from the initial specification $\mathcal{S}$ by the application of some basic modifications such as changing the outputs and/or the ending-states of some transitions, adding/deleting transitions or states (El-Fakih et al., 2004), and correspondingly the reduced $\mathcal{S}'$ of the modified specification can have n states where $n < m$. Finally, it is assumed that the initial implementation $\mathcal{T}$ was modified in order to get an implementation that conforms to the modified specification. We assume that the modified implementation was obtained by modifying only those transitions of $\mathcal{T}$ which correspond to the modified specification transitions (El-Fakih et al., 2004). In other words, for each unmodified transition from state s_j under input a, it is assumed that the transition from a corresponding state t_j under a is left intact in the modified implementation. Incremental testing deals with the generation of a set of test cases, called *incremental test suite*, that guarantees that if a modified implementation passes the test suite then it is a conforming implementation of the modified specification. This means an incremental test suite provides complete fault coverage under the assumption that the transitions of the implementation corresponding to unmodified transitions in the specification are not modified. The incremental testing problem can be reduced to testing from given user defined faults, i.e. w.r.t. fault model ($\mathcal{S}'$, $\cong$, $Sub(\mathcal{M})$), where, in this case, the mutation machine $\mathcal{M}$ is derived by changing the modified transitions of $\mathcal{T}$ to chaotic transitions and the specification $\mathcal{S}'$ is the reduced form of the modified specification. Thus, the mutation machine includes only chaotic transitions corresponding to the modified transitions and deterministic transitions corresponding to the un-modified transitions of $\mathcal{T}$. In fact, the mutation machine $\mathcal{M}$ inherits the conforming part of the original implementation and includes as submachines the set of all possible implementations of the modified specification. However, the reduced form of the modified specification can have less states than the initial specification, and thus, the number of states of the mutation machine $\mathcal{M}$ can be greater that that of the specification $\mathcal{S}'$.

3.2. Mutation Machine for FSM-Based Mutation Testing

Given a deterministic FSM representing a given system, the FSM can be non-reduced, for example, when the FSM is derived from a given code. Consider k disjoint sets of subsets of transitions of the given FSM. For each selected set of transitions, we derive a corresponding mutation machine $\mathcal{M}_j, j = 1, ..., k$, that includes as submachines all possible mutants of that subset. The mutation machine is derived in a special way that fits our purposes, i.e. the machine includes only deterministic and chaotic transitions. Then, we use the method presented in this paper for deriving a complete test suite w.r.t. the fault model ($\mathcal{S}$, $\cong$, $Sub(\mathcal{M})$) using $\mathcal{M}_j$ and $\mathcal{S}$. The derived test suite can detect any faulty IUT that can be distinguished from $\mathcal{S}$ using any set of transitions in the selected subset. Thus, if the collection of considered subsets includes all transitions of the specification FSM then the collection of derived test suites for such subsets of transitions detects mutants of the considered fault models which have up to m states. The mutation machine $\mathcal{M}_j$ is derived by changing each selected transition of the given FSM into chaotic. However, the reduced form $\mathcal{S}$ of the initial specification FSM can have less states than mutation machines $\mathcal{M}_j, j = 1, ..., k$. In order to obtain polynomial length tests, the k sets of transitions are selected in such a way such that the initial state of selected transition is reachable in $\mathcal{M}_j$ through deterministic transitions.

4. TEST DERIVATION APPROACH

In this section, we propose a test derivation approach w.r.t. the fault model ($\mathcal{S}$, $\cong$, $Sub(\mathcal{M})$), where $\mathcal{S}$ is a reduced deterministic specification FSM with n states and $\mathcal{M}$ is non-deterministic possibly non-reduced mutation machine with m states, $m \geq n$, such that $\mathcal{M}$ has only deterministic and chaotic transitions.

4.1. Test Derivation: State Identification Phase

Similar to other FSM-based testing methods, our proposed test derivation method has two phases. In the first "*state identification*" phase, tests, called TS_{Alg1}, are selected using Algorithm 1 in order to identify all the states of the IUT. In the "*transition testing*" phase, tests are selected to check chaotic, and in some cases deterministic transitions, when needed, for correct output and correct ending states.

When an IUT, i.e. a deterministic submachine of $\mathcal{M}$, can have more states than $\mathcal{S}$, in order to reach and identify all states in the implementation, we need to append the state cover set of the specification $\mathcal{S}$, with all input sequences of length $(m - n)$. Correspondingly, the length of a complete test suite is exponential in respect to the difference $(m - n)$ (Lee and Yannakakis, 1996; Bochmann and Petrenko, 1994). In order to reduce the length of a test suite, in this section, we present an algorithm that derives a state cover set for all possible implementations of $\mathcal{S}$, by utilizing accessible (known) information provided by the mutation machine $\mathcal{M}$.

In order to simplify our notations, hereafter, we denote $\mathcal{M} = (M, X, Y, h_M, m_1)$ and $\mathcal{S} = (S, X, Y, \delta, \lambda, s_1)$ the mutation machine and its corresponding reduced specification.

Algorithm 1. Deriving a state cover set for each submachine of mutation machine $\mathcal{M}$.

Input: The reduced specification $\mathcal{S}$ with $|S| = n$, a separating family $F = \{W_1, ..., W_n\}$ of $\mathcal{S}$, the mutation machine $\mathcal{M}$ with $|M| = m \geq n$.

Output: A set of test sequences TS_{Alg1} and a state Cover Set (CS) for all deterministic submachines of $\mathcal{M}$ that have the same output response to each sequence of the set TS_{Alg1} as the specification FSM.

Step 1. For each state $m_j \in M$ that is deterministically reachable from m_1 through a sequence β_j, include m_j in the (initially empty) set M_d, include β_j in the (initially empty) prefix-closed cover set Q_d of the set M_d, and include the state $s_j \in S$ where $s_j = \delta(s_1, \beta_j)$ into the (initially empty) set of states S_d. Then, derive a prefix-closed cover Q_{nd} of the set of states in $S \backslash S_d$ of $\mathcal{S}$ and obtain the union $Q = Q_d \cup Q_{nd}$.

For each state s_j of $\mathcal{S}$, determine the set $d(s_j)$ of all states of $\mathcal{M}$ that are deterministically distinguishable from s_j by the state identifier W_j.

Step 2. Include the sequences of Q into the (initially empty) cover set CS.

If $|Q| = |M|$ then Go-to **Step 3.** Otherwise, for every sequence $\alpha \in Q$ and each sequence $x\mu$ of length $m - |Q|$ such that (i) ax is not a prefix of a sequence in Q, (ii) $\alpha \in Q_d$ and the transition under input x from the state that is deterministically reachable from m_1 by α is chaotic or $\alpha \in Q_{nd}$, do the following:

Determine the shortest prefix χ of $x\mu$ such that there exist a set of k non-empty prefixes $\chi_1, \chi_2, ..., \chi_k$ (χ_1 is a prefix of χ_2, etc.) of the sequence $x\mu$ such that $k + |\hat{\mathbf{S}}|$, where $\hat{\mathbf{S}} = S_{nd} \cap \{\delta(s_1, \alpha\chi_i): i = 1, ..., k\}$, is greater than the cardinality of the set $(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)$.

Include into CS each sequence $\alpha\nu$, where ν is a proper prefix of χ_k. Include into the set TS_{Alg1} the sequences $\alpha\chi_k W_k$ where W_k is the identifier of the state $s_k = \delta(s_1, \alpha\chi_k)$.

If there is no such prefix χ then include into CS each sequence $\alpha\nu$ where ν is a prefix of $x\mu$.

Step 3. For every sequence γ_j in the set CS, include into the set of tests TS_{Alg1} all the sequences $r\gamma_j W$, where W is a state identifier of state $\delta(s_1, \gamma_j)$. End Algorithm 1.

We note that when $m = n$, a state cover set returned by Algorithm 1 coincides with a cover set of the specification FSM. If all transitions of the mutation FSM are chaotic then Algorithm 1 returns a cover set that coincides with a cover set obtained using traditional FSM methods. If $|Q|$ derived at Step 1 equals to m, then a cover set has m sequences, i.e., its length becomes polynomial.

Theorem 1. If an implementation $\mathcal{T}$ that is a submachine of the mutation machine $\mathcal{M}$ passes the sequences of the set TS_{Alg1} derived by Algorithm 1 then the set CS obtained by this algorithm is a state cover set of $\mathcal{T}$.

Proof. Let $\mathcal{T} \in Sub(\mathcal{M})$, $\mathcal{T} = (T, X, Y, \delta_T, \lambda_T, t_1)$, and the output responses of $\mathcal{S}$ and $\mathcal{T}$ to each input sequence of the set TS_{Alg1} coincide. Then sequences of the set Q take the FSM $\mathcal{T}$ from the initial state to $|Q|$ different states. Correspondingly, the FSM $\mathcal{T}$ can reach each reachable state from one of these states by an input sequence with length at most $m - |Q|$. Consider an input sequence $a\gamma$ of length $m - |Q|$ that should be applied after an appropriate input sequence $\beta \in Q_d$. If the transition from state m, which is reached in $\mathcal{M}$ (and $\mathcal{T}$) after β, under input a is deterministic then state m' reached in $\mathcal{M}$ (and correspondingly in $\mathcal{T}$) after βa can also be reached via an appropriate sequence β' of the set Q_d; and thus, each state that is reachable via some prefix of $a\gamma$ from the state reached by β, is also reachable from a state reachable after β' through a shorter prefix. For this reason, we do not include the sequence $\beta a\gamma$ into the set CS.

We now assume that there exist non-empty prefixes $\chi_1, ..., \chi_k$ of the sequence $a\gamma$ such that $(k + |\hat{\mathbf{S}}|)$, where $\hat{\mathbf{S}} = S_{nd} \cap \{\delta(s_1, \alpha\chi_i) : i = 1, ..., k\}$, is larger than the cardinality of the set $(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)$. We observe that $|\hat{\mathbf{S}}|$ different states of the implementation $\mathcal{T}$ are already reached after sequences of the set Q_{nd} if the output responses of FSMs $\mathcal{T}$ and $\mathcal{S}$ to each input sequence of the set TS_{Alg1} coincide. If after some prefix χ_j in the FSM $\mathcal{T}$ there occurs a state m of the set M_d of d-reachable states then all states reachable from this state can be also reached from the state m. Therefore, there are at most $|(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)| - |\hat{\mathbf{S}}|$ new states in the FSM $\mathcal{T}$ that can be reached after the prefixes $\chi_1, ..., \chi_k$ and thus, if $k + |\hat{\mathbf{S}}| > |(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)|$ and the output responses of FSMs $\mathcal{T}$ and $\mathcal{S}$ to each input sequence of the set TS_{Alg1} coincide, then the prefix $\alpha\chi_k$ will take the FSM $\mathcal{T}$ into a state that can be reached via another input sequence of the set CS.

In the following, given state m_j of the set M_d, we denote β_j the sequence of the set Q_d that deterministically takes the mutation machine $\mathcal{M}$ into state m_j from the initial state. Given state s_i of the set $S \backslash S_d$, we denote α_i the sequence of the set Q_{nd} that takes $\mathcal{S}$ to state s_i from the initial state.

Example. As an application example of Algorithm 1, consider the mutation FSM in Fig. 1 where non-suspicious (deterministic) transitions are shown and for simplicity of presentation suspicious (chaotic) transitions corresponding to user defined faults are not shown. The specification FSM $\mathcal{S}$ is shown in Fig. 2. The number n of states of $\mathcal{S}$ is 4 while that of $\mathcal{M}$ is $m = 7$.

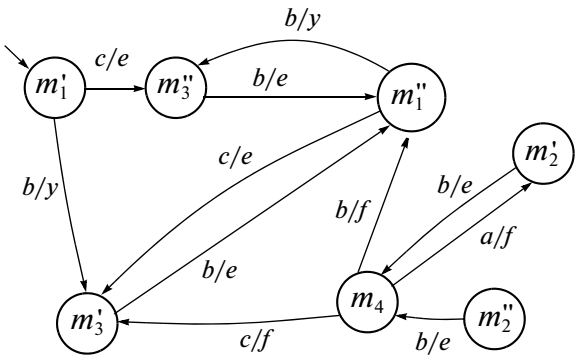

Fig. 1. Mutation FSM $\mathcal{M}$ where chaotic ransitions are not shown.

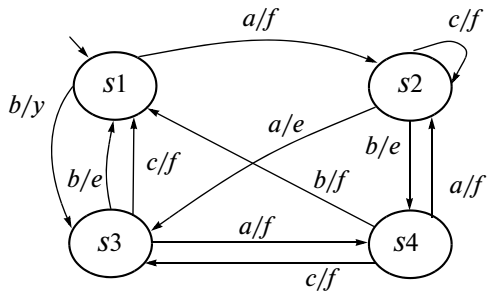

Fig. 2. Reduced specification $\mathcal{S}$.

The FSM $\mathcal{S}$ has a separating family $F = \{W_1 = \{b\}, W_2 = \{bb\}, W_3 = \{bb\}, W_4 = \{b\}\}$. Moreover, the state identifier W_1 d-distinguishes s_1 from m_2', m_2'', m_3', m_3'', m_4; W_2 d-distinguishes s_2 from m_1', m_1'', m_3', m_3'', m_4; W_3 d-distinguishes s_3 from m_1', m_1'', m_2', m_2'', m_4; W_4 d-distinguishes s_4 from m_1', m_1'', m_2', m_2'', m_3', m_3''.

States $M_d = \{m_1', m_1'', m_3', m_3''\}$ of $\mathcal{M}$ are d-reachable from the initial state; thus, we have $Q_d = \{\beta_1' = \varepsilon, \beta_1'' = bb, \beta_3' = b, \beta_3'' = c\}$ and the corresponding set of states $S_d = \{s_1, s_3\}$ of $\mathcal{S}$. The set $S_{nd} = S \backslash S_d = \{s_2, s_4\}$ and the corresponding cover set Q_{nd} is $\{\alpha_2 = a, \alpha_4 = ba\}$. The union $Q = Q_d \cup Q_{nd}$ is prefix-closed and $|Q|$ is 6. We include the sequences of Q into the set CS.

At Step 3 of Algorithm 1, for every sequence in the set Q, we derive and add into TS_{Alg1} corresponding test sequences. Thus, we add into TS_{Alg1}: $rb + rbbb + rbbb + rcbb + rabb + rbab$. If an implementation $\mathcal{T} \in Sub(\mathcal{M})$ passes these sequences then there exist 6 different states in $\mathcal{T}$. We note that the sequences $x\mu$ of length $m - |Q| = 1$ are a, b, and c.

In Step 2, for $\beta_1' = \varepsilon$ in Q; we do not consider $x\mu = a$, $x\mu = b$, $x\mu = c$, since a, b, c are in Q. We consider $\beta_3' = b$ in Q_d and the ending state $m_3' \in M_d$. We consider $x\mu = \{c, a\}$, since $\beta_3' \in M_d$ takes $\mathcal{M}$ to state m_3' from m_1 and c and a label the outgoing chaotic transitions from state m_3'. First, we consider $x\mu = c$, the ending state s_1 of the transition from state s_3 under input c is s_1; the state identifier $W_1 = b$ d-distinguishes s_1 from m_2', m_2'', m_3', m_3'', m_4. The cardinality of the set $|(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)|$ for $k = 1$ equals 0, since states m_1' and m_1'' are already covered by sequences in Q. Thus, we determine the shortest prefix $\chi = c$ of $x\mu = c$. We generate $r\beta_3' cW_1 = rbcb$ and add it into TS_{Alg1}. If $\mathcal{T}$ passes this sequence, then we are sure that bc takes an IUT to m_1' or m_1'' which are already reachable by sequences of CS. Thus, we do not need to add $\beta_3' = b$ into CS with the input c. We do not consider $x\mu = a$ since ba is in Q. In the same way, we consider $\beta_3'' = c$ in Q; and include into TS_{Alg1}: $r\beta_3'' cW_1 = rccb$ and $r\beta_3'' aW_4 = rcab$. For $\beta_1'' = bb$ in Q, we include $\beta_1'' a = bba$ into CS.

Now we consider $\alpha_4 = ba$ in Q, the ending state s_4 reached by ba is in Q_{nd}. We consider every sequence $x\mu = \{a, b, c\}$. First consider $x\mu = a$, $\delta(s_1, \alpha_4 a)$ is s_2 and has the state identifier $W_2 = bb$ that d-distinguishes s_2 from m_1', m_1'', m_3', m_3'', m_4. The cardinality of $|(M \backslash \bigcap_{j=1}^{k} d(\delta(s_1, \alpha\chi_i)) \backslash M_d)|$ for $k = 1$ equals to 2, the cardinality of $\hat{\mathbf{S}}$ where $\hat{\mathbf{S}} = S_{nd} \cap \{\delta(s_1, \alpha\chi)\}$ equals to 1, thus, we add the sequence $rbaa$ into CS. For the same reasons considered in the previous paragraph, we do not include into CS the sequence ab, while $r\alpha_4 cW_3 = rbacbb$ is included into TS_{Alg1}. Similarly, we consider sequence $\alpha_2 = a$ in Q and add into TS_{Alg1} the sequences $r\alpha_2 aW_3 = raabb$, $r\alpha_2 bW_4 = rabb$ and $r\alpha_2 cW_2 = racbb$. Also, we add into the set CS the sequence $\alpha_2 c = ac$. At the last step we add into TS_{Alg1} all sequences in the set CS appended by the corresponding state identifiers.

The obtained set CS contains the sequences $\{\varepsilon, bb, b, c, a, ba, ac, baa, bba\}$ and is of length 16. In comparison the, the HSI method generates a test suite of length more than 400.

4.2. Test Derivation: Transition Testing Phase

In this section, we show how test sequences for testing transitions of the mutation machine can be derived. Let $F = \{W_1, ..., W_n\}$ be a separating family of $\mathcal{S}$. We construct a state cover set CS and a set of state identification test sequences TS_{Alg1} as described in Algorithm 1.

Algorithm for the transition testing phase

(i) For each sequence γ that takes $\mathcal{S}$ from the initial state to s_i do the following:

(a) If $\gamma \in Q_d$, we only need to check the outgoing transitions of the state to which γ takes $\mathcal{M}$ from the initial state. For each input x such that the transition under x at state s_i is chaotic in $\mathcal{M}$, we include into the test suite the set of test sequences:

$$r\gamma xWk, \quad (2)$$

where W_k is a state identifier of state s_k that is reached in $\mathcal{S}$ when x is applied as an input at state s_i.

(b) If γ traverses a chaotic transition in $\mathcal{M}$ but state s_i of $\mathcal{S}$ is d-distinguished by W_j from $(m-1)$ states of $\mathcal{M}$, for each outgoing transition at state s_i under input x, the transition test sequences are formed as in **(a)** given above. The reason is that $\mathcal{S}$ is equivalent to a submachine of $\mathcal{M}$ and if an IUT has the expected output response to sequences in TS_{Alg1} then there is only one fault-free option left for a deterministic transition from the state reached in the IUT under the sequence γ.

(c) If γ traverses a chaotic transition in $\mathcal{M}$ and state s_i is not d-distinguished by W_i from at least two states of $\mathcal{M}$, the transition test sequences are also derived by applying Formula (2) for each outgoing transition of state s_i.

(ii) Include into the (initially empty) test suite TS the sequences obtained above and the set of sequences TS_{Alg1} obtained in the state identification phase using Algorithm 1. Delete from the set TS all sequences that are prefixes of other sequences and all sequences that traverse only deterministic transitions in $\mathcal{M}$ as the output responses of all implementations to these sequences are known.

Theorem 2. Given the specification $\mathcal{S}$ and the mutation machine $\mathcal{M}$, let $F = \{W_1, ..., W_n\}$ be a separating family of $\mathcal{S}$. If the implementation $\mathcal{T} \in Sub(\mathcal{M})$ passes the test sequences returned by Algorithm 2 then the implementation $\mathcal{T}$ is equivalent to $\mathcal{S}$.

Example. In our working example, consider bb of CS (obtained using Algorithm 1), where $\delta(s_1, bb) = s_1$. We apply Case (i)–(a) since $bb \in Q_d$ and obtain the sequence $rbbaW_2$. Consider ba of CS, where ba traverses chaotic transitions in $\mathcal{M}$ and $\delta(s_1, ba) = s_4$. All outgoing transitions of s_4 are deterministic; however, we do not need to test these transitions since Case (i)–(b) applies. Now consider a of CS, where a traverses chaotic transitions in $\mathcal{M}$ and $\delta(s_1, a) = s_2$. Case (i)–(c) applies, accordingly, we derive the sequences $raaW_3$, $rabW_4$, and $racW_2$ to test all outgoing transitions at s_2. Similarly, we consider all other sequence in CS, include all derived test sequences and the sequences of TS_{Alg1} into TS, apply (ii) above and obtain a TS of length 78. In comparison, the HSI method generates a test suite of length more than 1000 if the whole specification $\mathcal{S}$ is used for test derivation.

5. EXPERIMENTAL RESULTS

For a given reduced FSM $\mathcal{S}$ with n states and k input symbols, the worst-case length of the test suite generated by the FSM methods is of the order $O(k^{m-n+1}n^3)$ (Chow, 1978) when an implementation FSM can have at most m states. The worst-case length of a test suite derived using our approach is of the same order when all the transitions of the mutation machine are suspicious (chaotic). In some other cases, the length of a test suite derived using our approach can be of a lower order. Due to Algorithm 1, the length of a state cover set CS and correspondingly of a test suite is proportional to $k^{m-|Q|+1}$, where $|Q| \geq n$. When $|Q| = m$; for example, this happens when all states of the mutation machine are reachable through deterministic transitions, the worst-case length of a test suite is $O(kn^3)$.

In order to get a feeling about the length of test suites in practical situations, we have experimented with the testing method described in this paper. Figure 3 provides a comparison between the test suite length obtained by the HSI method and by the method presented in this paper for the case when $m > n$. The comparison is based on randomly generated completely specified reduced specifications, with a varying number of states (n) and inputs (k), and corresponding randomly generated completely specified mutation machines with varying number of states m (specifically, $m = n + 1$, $m = n + 2$). Although state-oriented specifications and implementations of real systems may have somehow different characteristics than randomly generated FSMs, we think that the conclusions of Fig. 3 also apply to system specifications and implementations that occur in practice.

For each pair (n, k) shown on the X-axis of Figs. 3a and 3b, we randomly generated 50 specifications with n states and k inputs. For each of these specifications, $m = n + 1$ (and $m = n + 2$) we generated 50 corresponding mutation machines with m states with a certain percentage of randomly selected chaotic transitions. Ten mutation machines had 5% of their transitions chaotic, ten had 10% chaotic, ten had 15% chaotic, ten had 20% chaotic, and the last ten had 30% of their transitions chaotic. For each of these mutation machines, we applied the testing method presented in this paper and calculated the average length of the obtained test suites of each group of ten mutation machines. Moreover, for all generated specifications with n states and k inputs, we computed the average length of test suites generated by HSI method. The ratios (length of test suites using our method)/(length of HSI test suites) are shown on the Y-axis of Fig. 3.

According to the experiments when chaotic transitions represents up to 5, 10, 15, 20, and 30% of the whole mutation machine, on average, the length of the obtained test suites are 66, 14, 24, 36, and 87 (10, 20, 30, 50, and 14) percent of the corresponding HSI test

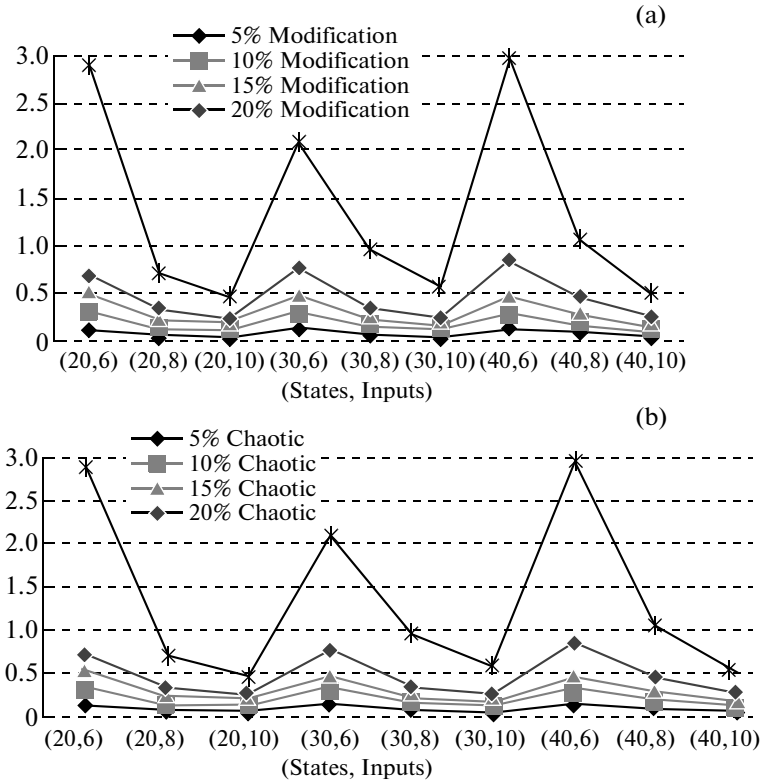

Fig. 3. (a) Experiments with $m = n + 1$, (b) experiments with $m = n + 2$.

suites when $m = n + 1$ ($m = n + 2$). We observe that these percentages do not significantly depend on the sizes of the specifications. It is clear from the above results that the gain of using our testing method is tremendous and this gain significantly increases as the difference $(m - n)$ increases. Correspondingly when performing a mutation testing we can check 20% of the specification FSM at each step. After five steps each transition will be covered by a corresponding mutation machine, i.e., our test derivation method will be applied five times. The length of a corresponding test suite is expected to be around 2.5% percent of the corresponding HSI test suites when $m = n + 2$.

6. CONCLUSIONS

In this paper, we have presented a test generation method that reduces the length of tests derived from a mutation machine with only deterministic and chaotic transitions. We consider the general case when the number m of states of the mutation machine may be greater than the number n of states of the specification. We show how the method can be used for FSM-based incremental and mutation testing. In some cases, the length of obtained test suites becomes polynomial. According to our experiments, when chaotic transitions represent up to 5, 10, 15, 20, and 30% of the mutation machine, on average, the length of obtained test suites are 66, 14, 24, 36, and 87 (10, 20, 30, 50, and 14) percent of the corresponding test suites, derived for the whole specification, when $m = n + 1$ ($m = n + 2$).

REFERENCES

1. Bochmann, G. and Petrenko, A., Protocol Testing: Review of Methods and Relevance for Software Testing, *Proc. of the International Symposium on Software Testing and Analysis (ISSTA 1994)* (Seattle, 1994), pp. 109–123.
2. Chow, T.S., Test Design Modeled by Finite-State Machines, *IEEE Trans. Software Eng.*, 1978, vol. 4, no. 3, pp. 178–187.
3. De Souza, S. de R., Maldonado, J.C., Fabbri, S.C.P.F., and De Souza, W.L., Mutation Testing Applied to Estelle Specifications, *Proc. of the 33rd Annual Hawaii International Conference on System Sciences (HICSS)*, Hawaii: IEEE Computer Society, 2000.
4. Dorofeeva, R., El-Fakih, K., and Yevtushenko, N., An Improved FSM-Based Conformance Testing Method, *Proc. of the IFIP 25th International Conference on Formal Methods for Networked and Distributed Systems*, Taiwan: Lecture Notes in Computer Science 3731, 2005a, pp. 204–218.
5. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., and Yevtushenko, N., Experimental Evaluation of FSM-Based Testing Methods, *Proc. of the IEEE International Conference on Software Engineering and Formal Methods (SEFM05),* Germany, 2005b, pp. 23–32.

6. Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A.R., and Yevtushenko, N., FSM-Based Conformance Testing Methods: a Survey Annotated with Experimental Evaluation, *Inform. Software Technol. J., Elsevier*, 2010, vol. 52, pp. 1286–1297.
7. El-Fakih, K., Yevtushenko, N., and Bochmann, G.V., FSM-Based Incremental Conformance Testing Methods, *IEEE Trans. Software Eng.*, 2004, vol. 30, no. 7, pp. 425–436.
8. Fabbri, S.C.P.F., Maldonado, J.C., Masiero, P.C., and Delamaro, M.E., Mutation Analysis Testing for Finite State Machines, *Proc. of the 5th International Symposium on Software Reliability Engineering. Monterey* (California, 1994), pp. 220–229.
9. Fabbri, S.C.P.F., Maldonado, J.C., Delamaro, M.E., and Masiero, P.C., Proteum/FSM: A Tool to Support Finite State Machine Validation Based on Mutation Testing, *Proc. of the XIX International Conference of the Chilean Computer Science Society (SCCC1999)* (Talca, Chile, 1999), pp. 96–104.
10. Fujiwara, S., Bochmann, G.V., Khendek, F., Amalou, M., and Ghedamsi, A., Test Selection Based on Finite State Models, *IEEE Trans. Software Eng.*, 1991, vol. 17, no. 6, pp. 591–603.
11. Gill, A., *Introduction to the Theory of Finite-State Machines*, McGraw-Hill, 1962.
12. Hierons, R.M. and Merayo, M.G., Mutation Testing from Probabilistic and Stochastic Finite State Machines, *J. Systems Software*, 2009, vol. 82, no. 11, pp. 1804–1818.
13. Jia, Y. and Harman, M., An Analysis and Survey of the Development of Mutation Testing, King's College, Londin, Crest Center, Technical Report TR-09-06.
14. Koufareva, I., Petrenko, A., and Yevtushenko, N., Test Generation Driven by User-Defined Fault Models, *Proc. of the 11th International Conference on Testing of Communicating Systems (TestCom 1999)* (Hungary, 1999), pp. 215–233.
15. Koufareva, I. and Dorofeeva, R., A Novel Modification of W-Method, *Joint Bull. Novosibirsk Comput. Center and A.P. Ershov Inst. Inform. Systems*, 2002, vol. 18, pp. 69–81.
16. Lee, D. and Yannakakis, M., Testing Finite-State Machines: State Identification and Verification, *IEEE Trans. Comput.*, 1994, vol. 43, no. 3.
17. Lee, D. and Yannakakis, M., Principles and Methods of Testing Finite State Machines-a Survey, *Proc. IEEE*, 1996, vol. 84, no. 8, pp. 1090–1123.
18. Maldonado, J.C., Sugeta, T., and Wong, W.E., Mutation Testing Applied to Validate sdl Specifications, *Proc. of the 16th IFIP International Conference on Testing of Communicating Systems (TestCom 2004)*, Springer LNCS 2978, pp. 193–208.
19. Petrenko, A., Checking Experiments with Protocol Machines, *Proc. of the 4th International Workshop on Protocol Test Systems (IWPTS 1991)* (Netherlands, 1991), pp. 83–94.
20. Petrenko, A. and Yevtushenko, N., Test Suite Generation for a fsm with a Given Type of Implementation Errors, *Proc. of the 12th International Workshop on Protocol Specification, Testing and Verification* (Canada, 1992), pp. 229–243.
21. Petrenko, A., Yevtushenko, N., Lebedev, A., and Das, A., Nondeterministic State Machines in Protocol Conformance Testing, *Proc. of the IFIP 6th Sixth International Workshop on Protocol Test systems* (France, 1993), pp. 363–378.
22. Petrenko, A. and Yevtushenko, N., Testing from Partial Deterministic FSM Specifications, *IEEE Trans. Comput.*, 2005, vol. 54, no. 9, pp. 1154–1165.
23. Simao, A., Petrenko A., and Yevtushenko, N., Generating Reduced Tests for FSMs with Extra States, *Proc. of the 21th International Conference on Testing of Communicating Systems and the 9th International Workshop on Formal Approaches to Testing of Software (TestCom/Fates 2009)*, LNCS 5826, pp. 129–145.
24. Starke, P., *Abstract Automata*, American Elsevier, 1972.
25. Vasilevskii, M.P., Failure Diagnosis of Automata, *Kibernetika*, 1973, vol. 4, pp. 98–108.
26. Vuong, S.T., Chan, W.W.L., and Ito, M.R., The UIOv-Method for Protocol Test Sequence Generation, *Proc. of the IFIP International Workshop on Protocol Test Systems*, 1989, pp. 161–175.
27. Yannakakis, M. and Lee, D., Testing Finite State Machines: Fault Detection, *J. Comput. System Sci.*. 1995, vol. 50, pp. 209–227.
28. Yevtushenko, N. and Petrenko, A., Test Derivation Method for an Arbitrary Deterministic Automaton, in *Automatic Control and Computer Sciences*, USA: Allerton Press Inc., 1990, p. 5.