Software Qual J (2015) 23:423–452
DOI 10.1007/s11219-014-9242-6

Fault domain-based testing in imperfect situations: a heuristic approach and case studies

Fevzi Belli · Mutlu Beyazıt · Andre Takeshi Endo · Aditya Mathur · Adenilso Simao

Published online: 18 June 2014

Abstract Model-based testing (MBT) involves creating an abstraction, called a model, to represent the system and automatically deriving test cases from this model. MBT can be performed using various approaches that generally employ certain assumptions or requirements affecting the test performance in practice. Here, we consider the harmonized state identifiers (HSI) method, which is based on finite state machine (FSM) models and generates test sets that cover all faults in a given domain under certain conditions. We are interested in the application of the HSI method in practical scenarios where some conditions do not hold or are not straightforward to satisfy. Thus, we propose a heuristic extension to the HSI method, called heuristic HSI (HHSI), to consider imperfect situations

The authors are alphabetically ordered.

Electronic supplementary material The online version of this article (doi:10.1007/s11219-014-9242-6) contains supplementary material, which is available to authorized users.

F. Belli · M. Beyazıt
University of Paderborn, Paderborn, Germany
e-mail: belli@adt.upb.de

M. Beyazıt
e-mail: beyazit@adt.upb.de

F. Belli
Izmir Institute of Technology, Izmir, Turkey

A. T. Endo (✉)
Universidade Tecnologica Federal do Parana, Cornelio Procopio, Brazil
e-mail: andreendo@utfpr.edu.br

A. Mathur
Singapore University of Technology and Design, Singapore, Singapore
e-mail: apm@purdue.edu

A. Simao
Universidade de Sao Paulo, Sao Carlos, Brazil
e-mail: adenilso@icmc.usp.br

as they often occur in practice. To analyze the characteristics of HHSI, we empirically compare it to random testing and coverage-based testing using non-trivial case studies. The experiments include model-based mutation analyses over several FSM models.

Keywords Model-based testing · Fault domain-based · Finite state machines · HSI method · Imperfect situation · Heuristic HSI

Abbreviations

MBT	Model-based testing
FSM	Finite state machine
SUC	System under consideration
HSI	Harmonized state identifiers
HHSI	Heuristic HSI

1 Introduction

Computer-based systems pervade nearly all contexts, from cell phones to giant banking systems. It is important that these systems work correctly and provide a high level of reliability. Thus, there has been an increasing demand for formal and systematic testing, such as *model-based testing* (MBT). MBT is an approach that aims at automatic test case generation using rigorous test models created by testers. According to Hierons et al. (2009), the adoption of formal models and specifications increases the efficiency and effectiveness in the testing process. Moreover, the literature reports various benefits, such as high fault detection capability, requirement evolution, reduced cost and time for testing, and traceability (Utting and Legeard 2006).

To exploit such benefits, MBT should be applied carefully and correctly in a project. This involves issues regarding the adoption of an MBT approach in a project requires making some critical decisions on the selection of test generation methods and modeling techniques based on the characteristics of the system under consideration (SUC). From a testing point of view, the selection of test generation method is especially important, because it determines the testing approach to be employed.

Finite state machines (FSMs) have been studied for more than 50 years, being often used in MBT (Mealy 1955; Hennie 1964; Vasilevskii 1973; Chow 1978; Lee and Yannakakis 1996); they usually represent the SUC by means of states and transitions that consume inputs and produce outputs. In this context, several test generation methods, such as W (Vasilevskii 1973; Chow 1978), Wp (Fujiwara et al. 1991), HSI (Luo et al. 1995), H (Dorofeeva et al. 2005), and SPY (Simao et al. 2009), have been proposed and used in system/software testing, such as testing of protocols and Web-based systems. These methods are *fault domain-based* since they produce test sets aiming at covering all faults in a given fault model or domain. Among the approaches to FSM-based testing, fault domain-based testing is the most prominent topic. In general, such testing approaches rely on defining a certain set of faults that may exist in the SUC and is intended to be revealed, called fault domain. Under certain conditions or assumptions, test sets generated using these approaches guarantee the discovery of all possible faults from the defined domain. Usually, these conditions require that the test models are reduced, i.e., there are no redundant states, and that the maximal number of states in the implementation is known. However, the conditions required to do this may be difficult or even impossible to satisfy in real-life scenarios.

From a practical point of view, it is often hard to obtain properly reduced test models, because testers usually design non-reduced models and the application of traditional FSM minimization algorithms may yield models that are not suitable for test generation. Also, it is difficult for testers to determine the number of states or an upper bound on the number of states in the implementation, since one does not always have access to the internals of the system. From another point of view, even if a fault domain-based test generation method relies on relatively weaker conditions, it may not be practical to use. Thus, in a broader context, the situations in which one or more conditions required by the fault domain-based testing approach to be applied effectively do not hold are called *imperfect situations*.

In this paper, we propose an improvement of the fault domain-based testing to enable its application in real-life scenarios which lead to imperfect situations. Such situations hinder employing some well-known methods. Therefore, we describe the use of heuristic extensions to those methods. To exemplify our approach, we select HSI (harmonized state identifiers), since it is applicable to partial FSMs, it is a simple algorithm that scales well with large models, and it works with an implementation with more states than the specification. This method is revised and named as the *heuristic HSI method* (HHSI) to be applicable in the imperfect situation characterized as follows.

- The FSM test model designed by the tester is not reduced, i.e., some state pairs cannot be distinguished by any input sequence.
- The direct application of traditional FSM minimization to the test model does not yield a properly reduced machine, because infeasible test cases can be generated from the reduced machine.

In comparison with random and coverage-based testing, we focus on answering the following questions to provide practical insight for helping testers to better choose, adapt, and properly use MBT techniques in imperfect situations.

- What are the characteristics of the test sets generated using the *heuristic* HSI method?
- What is the fault detection effectiveness of using the heuristic HSI method for *different fault domains*?
- What are the *costs* associated with each approach from a practical point of view?
- What are the trade-offs while testing in imperfect situations?

The main contributions of the paper are twofold. First, a heuristic version of the fault domain-based HSI method is introduced for the imperfect situation where the FSM specification is not reduced and the direct application of traditional FSM minimization does not yield a properly reduced machine. Second, experiments are performed to analyze characteristics of the proposed method with respect to traditional approaches (random testing and coverage-based testing) in situations where the SUCs and their models have the same and different numbers of states.

The paper is organized as follows. Section 2 discusses related work. Section 3 presents the fault domain-based testing approach used in this paper. Section 4 introduces the heuristic that is used for its adaptation. After deriving the methodology for the HHSI method, we describe in Sect. 5 experiments to compare it to random testing and coverage-based testing and to analyze its characteristics. Finally, Sect. 6 presents the conclusion and discusses future work.

2 Motivation and related work

FSM-based testing has intensively been researched over the past few decades (Mealy 1955; Moore 1956; Gill 1962); comprehensive surveys on this topic can be found in Lee and

Yannakakis (1996), Lai (2002), Hierons et al. (2009), and Dorofeeva et al. (2010). In this area, the problem of selecting a test set that demonstrates the conformance between a specification and an implementation is arguably pivotal. This problem, so-called *conformance testing*, is formally discussed in Sandberg (2005), Krichen (2005), Björklund (2005), and Gargantini (2005). Conformance testing has been investigated in test case generation methods that cover all faults in a given domain. In general, fault domain-based methods require a set of assumptions and/or model properties to be effectively applied. For instance, some methods are applicable only for deterministic models and others require a reduced specification. Assumptions can be made about the fault domain so that test sets can be selected and proved to reveal all intended faults in this domain. Using FSMs, the proofs are based on the relationship between the specification and the implementation. In other words, if all assumptions hold, then the test set is capable of proving the equivalence (or trace inclusion, depending on the chosen conformance relation) between specification (model) and implementation.

Given a specification (which might be a test model in MBT) and an implementation under test (also referred as SUC), FSM-based test generation methods are based on the following assumptions:

- Reliable reset: There exists a reliable operation that brings the implementation to its initial state.
- FSM implementation: The implementation can be represented/abstracted by an FSM model.
- Known upper bound number of implementation states: Prior to generating the tests, the upper bound number of states in the implementation is known by the tester.

Furthermore, the following requirement is also expected to be fulfilled by most of the fault domain-based methods:

- Reduced specification: The model designed by the tester is reduced, which means that all states are pairwise distinguishable (the concept of reduced model and distinguishability of states are formally defined in Sect. 3.1).

The aforementioned assumptions are required by most of methods found in the literature, such as W (Vasilevskii 1973; Chow 1978), Wp (Fujiwara et al. 1991), HSI (Luo et al. 1995), H (Dorofeeva et al. 2005), and SPY (Simao et al. 2009). The W method (Vasilevskii 1973; Chow 1978) is considered the seminal work on these methods. It basically uses a transition cover set to reach states and a specific group of sequences called characterization set (or W set) for state identification. Other methods adopt different ways to identify states such as identification sets (Fujiwara et al. 1991) and separating families (Luo et al. 1995). Section 3.3 provides a detailed description of one of these methods, namely HSI. The fundamental difference among these methods is the size of the generated test sets. Researchers have investigated means to reduce the test set, while keeping the same properties. Experimental results on test set sizes comparing the existing methods can be found in (Dorofeeva et al. 2010; Endo and Simao 2013). These studies show that the recent methods (H and SPY) are able to produce test sets smaller than the traditional ones (W, Wp, and HSI). To do so, recent methods rely on identifying separating sequences that will have less impact on the test size while the test set is built (this strategy is called *on-the-fly*). The proposal of on-the-fly methods are backed up by sufficient conditions that determine whether an arbitrary test set has the required properties or not (Dorofeeva et al. 2005; Hierons and Ural 2006; Simao and Petrenko 2010). It was observed that recent methods rely on fewer and longer sequences to reduce the test set, while traditional methods show many short sequences (Endo and Simao 2013).

Initiatives have been taken to remove one or more of the assumptions, e.g., the reliable reset. Accordingly, the generation of checking sequences has been investigated (Hennie 1964; Gonenc 1970; Sidhu and Leung 1989; Rezaki and Ural 1995; Hierons and Ural 2006), where the test set consists of a single test case. However, most of the checking sequence generation methods impose other restrictions, such as strongly connected FSMs and the existence of a unique separating sequence for all state pairs (i.e., a *distinguishing sequence*—DS) (Hennie 1964; Gonenc 1970; Hierons and Ural 2006). Among them, the work of Gonenc (1970) has inspired other methods that use a DS to construct checking sequences. His method generates checking sequences through the manipulation of two types of sequences: α-sequence and β-sequence. The α-sequence aims to recognize all states identifying states reached after applying a DS, and the β-sequence is defined to test the transitions. The need for a DS was removed in the work of Rezaki and Ural (1995) by using W sets instead. As W sets exist for all reduced FSMs (Gill 1962), the Rezaki and Ural's method is more general. However, the checking sequence length grows exponentially with the number of sequences in the W set.

Most of discussed papers have focused on the use of deterministic FSMs. Nevertheless, there has been interest in investigating the test from non-deterministic machines (Zhang and Cheung 2003; Hierons 2004; Petrenko and Yevtushenko 2014). Non-determinism may occur in two ways: (i) a state can have different reactions (outputs and/or next states) for the same input, and (ii) internal transitions may exist and move the machine to a different state without producing any output (Zhang and Cheung 2003). As non-determinism usually prevents to decide which the next state is, input sequences used for deterministic FSMs (described in Sect. 3.1) are replaced by testing strategies, frequently represented as trees whose transitions indicate which inputs can be applied after a specific output is observed. Zhang and Cheung (2003) investigate three optimization problems related to the transfer tree (TT), which is used to reach a given state, and diagnosis tree (DT), which is used to identify a given state. Hierons (2004) introduces an adaptive algorithm that aim at reducing the test suite size in non-deterministic FSMs. The algorithm basically produces, at each state, an input sequence or an adaptive test case on basis of the input/output sequences already observed. Similar to the TT and DT, the adaptive test case is also a tree. Petrenko and Yevtushenko (2014) propose a method to generate adaptive test cases from non-deterministic specifications, allowing the implementation to behave non-deterministically. As in methods for deterministic FSMs, the authors define assumptions to be considered when assuring the coverage of all faults in a given domain.

Almost all generation methods (for both test sets and checking sequence) work with the assumption that the test model is a reduced specification. Motivated by the adoption of test generation methods and by the manipulation of simpler models with redundant states, several algorithms have been proposed to remove redundancy in FSMs (Grasselli and Luccio 1965; Pena and Oliveira 1998). The so-called minimization algorithms [as in (Grasselli and Luccio 1965; Pena and Oliveira 1998)] identify redundant states and try to remove them, while keeping the equivalence between the original and modified machine. At the end, a reduced specification is produced.

Among the required conditions, obtaining a reduced model is one of the bothersome tasks for testers. The mentioned minimization algorithms assume that undefined inputs in a state are "don't care" types (Luo et al. 1995). This assumption does not hold in many real-life systems (for example, for systems with GUIs). Undefined inputs usually mean that an input (event) is not enabled, that is, there is no way to fire the event in that state.

Consequently, the application of traditional FSM minimization and the use of the reduced model in test generation cause some test cases to be infeasible; i.e., they are not executable in the real SUC.

In such situations, depending on the SUC and the selected fault domain-based method, one may be allowed to perform additional operations on the model so that the conditions required by the fault domain-based method are satisfied. However, it is possible that such operations serve only as some heuristics to enable the application of the favored method. Thus, in the end, the generated test sets may have a reduced fault detection power, being unable to detect all the faults in the defined fault domain. Another point with a large effect on the fault detection power of the testing method is whether the upper bound number of states in the implementation is known or not. Some methods use this information in test generation to cover additional faults that originate from the difference between the number of states in the test model and in the implementation (Simao et al. 2009).

To our knowledge, state-counting (SC) (Petrenko and Yevtushenko 2005) is the only method in the literature that is applicable to both reduced and non-reduced FSMs and has the same theoretical guarantees of other classical methods (W, Wp, HSI, and so on). When applied on reduced specifications, it has the same behavior of traditional methods (e.g., HSI). What differs it from other methods is that SC is directly applicable to non-reduced FSMs (Petrenko and Yevtushenko 2005). In this scenario, the SC method basically counts how many times the tests passed by a given state and use this information to identify a state and distinguish it from other states. However, SC has some scalability problems since the strategy employed to distinguish states produces test sets that rapidly grow in function of the number of states. As a consequence, impractically big test sets can be derived from medium/large FSMs.

In this paper, we use the HSI method for fault domain-based test generation. However, since our models contain states that are not distinguishable (since not all inputs are defined for all states), we insert additional transitions to the model so that we are able to identify the states that can be merged, perform a proper minimization, and run the HSI method. Later, we remove the events related to these additional transitions from the sequences yielded by the HSI method and obtain our test sequences, which are not guaranteed to detect all the intended faults. We call this method heuristic HSI (HHSI) and analyze its characteristics by comparing it to random testing and coverage-based testing in both situations where (i) the implementation and the test models have the same number of states and (ii) they do not.

3 Fault domain-based testing

3.1 Model definition

In this paper, we consider a deterministic Mealy machine model that is composed of states and transitions. For each transition, an input symbol is consumed and an output symbol is produced. An FSM can be represented by a state diagram, which is a directed graph so that nodes are states and edges are transitions. The edges are annotated with inputs and outputs associated with the transition. Figure 1 exemplifies a state-transition diagram of an FSM.

Definition 1 A finite state machine (FSM) M is a 7-tuple $(S, s_0, I, O, D, \delta, \lambda)$, where:

- S is a finite set of *states* with initial state s_0,
- I is a finite set of *inputs*,

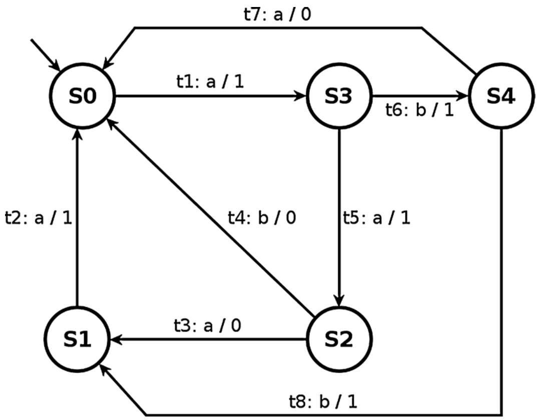

Fig. 1 An example FSM (which is also a reduced FSM)

- O is a finite set of *outputs*,
- $D \subseteq S \times I$ is a *specification domain*,
- δ*:* $D \rightarrow S$ is a *transition function*, and
- λ*:* $D \rightarrow O$ is an *output function*.

Tuple $(s, x) \in D$ is defined as a transition in state s that consumes input symbol x. A transition can be represented using the form $(s_i, x/y, s_j)$, which means there is a transition t from head state s_i to tail state s_j that consumes input symbol x and produces output symbol y. We say that t is an outgoing transition of s_i and an incoming transition of s_j. An FSM which has defined transitions for each input symbol in all states, i.e., $D = S \times I$, is *complete*; otherwise, it is *partial*. A sequence $\alpha = x_1 \ldots x_k$ ($\alpha \in I^*$) is defined as an input sequence for state $s \in S$, if there exist states $s_1, \ldots, s_{k+1}$ such that $s = s_1$ and $\delta(s_i, x_i) = s_{i+1}$ for all $1 \leq i \leq k$. Also, $\Omega(s)$ is used to denote all input sequences defined for state s and Ω_M is an abbreviation for $\Omega(s_0)$. Therefore, Ω_M represents all defined sequences for FSM M. The empty sequence is denoted by symbol ε.

Notation $\alpha\omega$ is used to denote the concatenation of the two sequences, α and ω. Sequence α is a prefix of sequence β, denoted by $\alpha \leq \beta$, if $\beta = \alpha\omega$, for some sequence ω. Sequence α is a proper prefix of β, denoted by $\alpha < \beta$, if $\beta = \alpha\omega$ for some sequence $\omega \neq \varepsilon$. Given two sets of sequences $D1$ and $D2$, $D1.D2$ is the set of sequences obtained by concatenating all sequences in $D1$ with all sequences in $D2$, that is, $D1.D2 = \{\alpha\beta | \alpha \in D1$ and $\beta \in D2\}$. Furthermore, $D^0 = \{\varepsilon\}$ and $D^{i+1} = D.D^i$, for $i \geq 0$.

The transition and output functions are extended for defined input sequences, including the empty sequence ε, as follows. For a state $s \in S$, $\delta(s, \varepsilon) = s$ and $\lambda(s, \varepsilon) = \varepsilon$; given an input sequence $\alpha x \in \Omega(s)$, we have $\delta(s, \alpha x) = \delta(\delta(s, \alpha), x)$ and $\lambda(s, \alpha x) = \lambda(s, \alpha)\lambda(\delta(s,\alpha), x)$.

Two states $s_i, s_j \in S$ are *distinguishable* if there exists a *separating sequence* $\gamma \in \Omega(s_i) \cap \Omega(s_j)$, such that $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$; otherwise they are not distinguishable. An FSM M is reduced if all states are pairwise distinguishable; otherwise it is non-reduced. Given a different FSM $N = (S', q_0, I, O, \Delta, \Lambda)$, we say that two machines M and N are distinguishable if there exists a sequence $\gamma \in \Omega_M \cap \Omega_N$, such that $\lambda(s_0, \gamma) \neq \Lambda(q_0, \gamma)$.

A test case of M is an input sequence $\alpha \in \Omega_M$. A test set of M is a finite set of test cases of M, such that there are no two test cases α and β, such that $\alpha < \beta$. In fact, if a test case α is a proper prefix of a test case β, the execution of β will always imply the execution of α. Thus, α can be removed without altering the test result.

Definition 2 Set Q of input sequences is a *state cover* of M if, for each state $s_i \in S$, there exists a sequence $\alpha_i \in Q$ that transfers the FSM from the initial state to s_i. This set includes sequence ε to reach the initial state.

Definition 3 Set P of input sequences is a *transition cover* of M if for each transition $(s, x) \in D$ there exist the sequences $\alpha, \alpha x \in P$ such that $\delta(s_0, \alpha) = s$. Set P also includes sequence ε.

For the FSM in Fig. 1, we have that $Q = \{\varepsilon, a, aa, ab, aaa\}$ and $P = \{\varepsilon, a, aa, ab, aaa, aab, aba, abb, aaaa\}$. After removing the prefixes properly, both sets can be used as test sets for state and transition coverage.

3.2 Fault domain

We now discuss the rationale behind the fault domain-based testing by defining the fault domain and its classes, as well as describing the complete test sets.

Set $\Im$ represents all deterministic FSMs with the same input alphabet as M for which all sequences in Ω_M are defined, that is, for each $N \in \Im$, $\Omega_M \subseteq \Omega_N$. Let $m \geq 1$ be an integer, $\Im_m$ denotes the set of all FSMs with at most m states such that $\Im_m \subseteq \Im$.

Definition 4 (*m-complete test sets*) Given a specification M with n states, a test set T in Ω_M is *m-complete* if, for each $N \in \Im_m$ distinguishable from M, there exists a test case in T that distinguishes M from N. An m-complete test set has *full fault coverage* for the defined domain, being able to detect all faults in any implementation with at most m states. A test set is called n-complete in the case that $m = n$.

The methods that generate m-complete test sets are based on the assumption that the implementation itself can be represented by an FSM. Thus, $\Im_m$ represents the domain of possible faulty implementations with at most m states. Under this assumption, these methods are formally proved with respect to the ability to produce complete test sets.

3.3 The HSI method

In this paper, we extend the HSI method, which is able to generate m-complete test sets for partial and reduced FSMs (Luo et al. 1995). We select HSI because it is applicable to partial FSMs and it is a simple algorithm that scales well with large models. Moreover, it works with an implementation with more states than the specification ($m \geq n$).

In the HSI method, separating families are used to identify states. A *separating family* is a set of input sequences $H_i \subseteq \Omega(s_i)$ for a state $s_i \in S$ that satisfies the following condition (Luo et al. 1995): For any two distinct states s_i, s_j, there exist sequences $\beta \in H_i$, $\gamma \in H_j$ and α, such that $\alpha \leq \beta$, $\alpha \leq \gamma$ and $\lambda(s_i, \alpha) \neq \lambda(s_j, \alpha)$. The separating families for the FSM in Fig. 1 are $H_0 = \{ab, aab\}$, $H_1 = \{aab\}$, $H_2 = \{a, b\}$, $H_3 = \{b, aa, ab\}$, and $H_4 = \{a, b\}$.

The number of states in the specification is n and the number of states in the implementation is assumed to be m, $m \geq n$. The application of HSI method can be divided into two parts:

(i) Construction of Z: Z is defined by concatenating the transition cover P with all defined sequences of up to a certain length that is defined by the number of extra states, that is, sequences with length m–n (Simao et al. 2009). Formally, $Z = (P \cup P.I \cup \ldots P.I^{m-n}) \cap \Omega_M$.

(ii) Concatenation of Z and separating families: The generation of m-complete test sets is performed by concatenating the sequences in Z with their separating families and removing the proper prefixes. In other words, $TS_{HSI} = \{\alpha\ H_i | \ \alpha \in Z$ and $\delta(s_0, \alpha) = s_i\}$.

When the specification and implementation have the same number of states $n = m$, $Z = P.\{\varepsilon\}$. Therefore, for the FSM in Fig. 1, when Z is concatenated with H_i, we have $TS_{HSI(n)} = \{$*aaaaab, aabaab, aabab, abaaab, abaab, abbaab, aaaaaab*$\}$.

When the implementation has one or two more states than the specification, that is, $m = n+1$ or $m = n+2$, respectively, $Z_{n+1} = P.\{\varepsilon\} \cup P.I$ and $Z_{n+2} = P.\{\varepsilon\} \cup P.I \cup P.I^2$, respectively, such that $Z_{n+1} \subseteq \Omega_M$ and $Z_{n+2} \subseteq \Omega_M$. Thus, for the FSM in Fig. 1, we have $TS_{HSI(n+1)} = \{$*aaaaab, aabaab, aabab, abaaab, abaab, abbaab, aaaaaab, aabaaa, abaaaa, abbaaab, aaaaaaa*$\}$ and $TS_{HSI(n+2)} = \{$*aabaab, abaaab, abbaab, aaaaaab, aabaaa, abaaaa, abbaaab, aaaaaaa, aababb, aababa, abaabb, abaaba, abbaaaa, aaaaabb, aaaaaba*$\}$.

4 Heuristic HSI method (HHSI)

In many event-based systems, such as GUI systems, when an input event is not defined in a given state, it means that this event is not enabled or cannot be fired. For instance, when a "deletion" event occurs, a modal dialog window is presented (reaching some state *s*) and only two events are enabled, "OK" and "Cancel." When modeling such a case, a partial machine needs to be used. However, it is likely that state *s* will not be distinguishable from the others since "OK" and "Cancel" may not be defined in other states. This fact leads to an imperfect situation, because it hinders the direct use of traditional FSM minimization algorithms, since they tend to merge states with disjoint sets of defined inputs. As a consequence, some test sequences derived from such reduced model cannot be applied to the SUC and, therefore, they are useless. Consequently, the imperfect situations considered in this paper are defined as follows.

Definition 5 (*Imperfect situation*) Let M be a non-reduced FSM and N be an FSM that is reduced from M (using traditional FSM minimization). An *imperfect situation* occurs if $\Omega_N \neq \Omega_M$.

For instance, when the FSM in Fig. 2 is reduced, some states are merged and we have a smaller machine with three states, as demonstrated in the FSM in Fig. 3. Test sequence *adbc* obtained from this reduced machine is not applicable to the SUC. In other words, sequence *adbc* is not defined in the original FSM (Fig. 2) and, as a consequence, cannot be executed on the SUC. Thus, the machine in Fig. 3 is not a properly reduced FSM for our case, causing an imperfect situation.

To tackle this issue, we present a heuristic solution we called heuristic HSI method (HHSI). This approach is divided into three parts that are described as follows.

Part 1: Construct a modified machine: First, we assume that undefined inputs in a given state of an FSM M are not enabled for that state, that is, they cannot be fired or provoked. For example, in Fig. 2, input *a* can only be applied to states *s1, s4,* and *s6*, while no input can be fired in state *s5*. This assumption holds for the models used in the case studies (Sect. 5).

The heuristic solution is based on the idea of minimizing the FSM while keeping the same defined input sequences. This is performed by merging only the states with the same defined input sequences. To do so, we add a self-transition for each state that consumes a special input *f* and produces an output which is the set of enabled inputs in that state. These

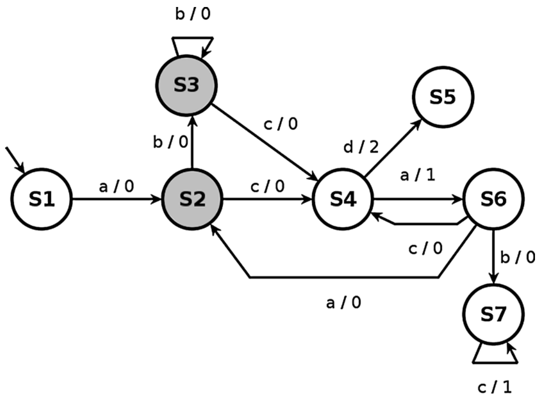

Fig. 2 An example of a non-reduced FSM

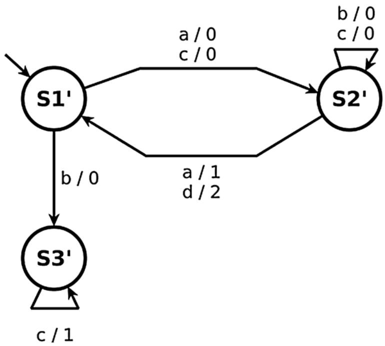

Fig. 3 Directly reduced FSM

transitions are labeled as *f*-transitions. For the FSM in Fig. 2, a modified FSM with the *f*-transitions is illustrated in Fig. 4.

Part 2: Minimize the machine: Using this modified machine, a traditional minimization algorithm (Grasselli and Luccio 1965; Pena and Oliveira 1998) is applied to obtain a reduced FSM. In this example, states *s2* and *s3* are not distinguishable and then merged in one state *s2–3*, as shown in Fig. 5.

When compared with the FSM in Fig. 3 which is reduced from the original model by a direct application of FSM minimization, the FSM in Fig. 5 correctly identifies the compatible states to prevent generation of infeasible sequences. Thus, it is a properly reduced version of the original FSM in our case, whereas the FSM in Fig. 3 is not.

Part 3: Generate the test set: When the heuristic HSI method is applied to the reduced machine in Fig. 5, separating families are built prioritizing sequences without *f*, that is, sequences with *f* are used only if no other sequence exists to distinguish a pair of states. The separating families for the FSM in Fig. 5 are $H_1 = \{a, f\}$, $H_{2\text{–}3} = \{c, bc, f\}$, $H_4 = \{a, f\}$, $H_5 = \{f\}$, $H_6 = \{a, c, bc, f\}$, and $H_7 = \{c, f\}$. Notice that especial input *f* is enough to distinguish states *s2–3* and *s6*. Although *f* would be shorter, we prioritize sequence *bc* which is included instead.

Next, the HSI method is applied in the FSM in Fig. 5 as explained in Sect. 3.3. Given that $P = \{\varepsilon, a, f, ab, ac, af, acd, aca, acf, acdf, acaa, acab, acac, acaf, acabc,$

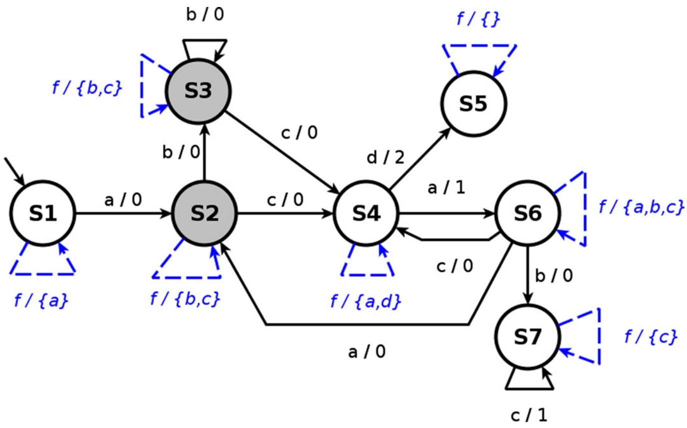

Fig. 4 An FSM with *f*-transitions

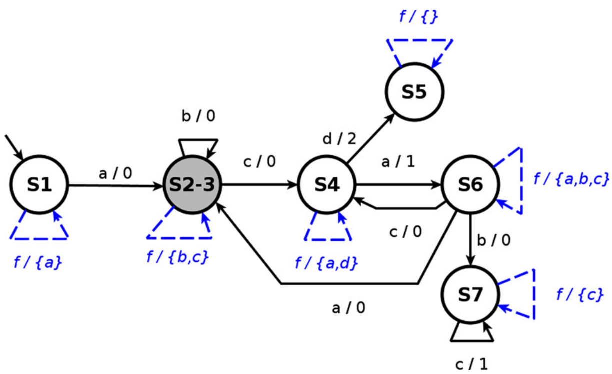

Fig. 5 Reduced FSM with *f*-transitions

acabf} and the specification and implementation have the same number of states $n = m$, we have $TS_{HHSI(n)} = \{$*abc, ff, fa, abf, abbc, aff, afc, afbc, acff, acfa, acdff, acaaf, acaac, acaabc, acacf, acaca, acaff, acafc, acafa, acafbc, acabcf, acabcc, acabff, acabfc*$\}$. Notice that event *f* occurs in both the transition cover *P* and the produced test suite $TS_{HHSI(n)}$.

Later, we remove the occurrences of event *f* from the generated test sequences, because event *f* does not really exist in the system. At the end, for the FSM in Fig. 2 (initially used and passed by intermediate steps illustrated in Figs. 4, 5), we have $TS_{HHSI(n)} = \{$*abc, abbc, acd, acaac, acaabc, acaca, acabcc*$\}$. These steps are the same for the case when the implementation has more states than the specification $m > n$.

As a consequence of the heuristic use of event *f*, the resulting test set may detect fewer faults, though states and transitions are still verified when possible. This procedure also assures that the generated test cases (from FSM in Fig. 5) are also defined in the original FSM presented in Fig. 2. Algorithm 1 outlines this process that is divided into three parts.

It is important to emphasize that all produced test sequences in TS_{HHSI} are defined in M, that is, $TS_{HHSI} \subseteq \Omega_M$.

Algorithm 1 Heuristic application of the HSI method

Input: $M = (S, s_0, I, O, D, \delta, \lambda)$ – an FSM
Output: TS_{HSI} – the test set derived out of M

```
//------------ Part 1: construct the modified M' --------//
M' = M
I' = I' ∪ {f}
for each s ∈ S' do
    out = {x | (s, x) ∈ D'}
    t = (s, f /out, s)
    D' = D' ∪ {(s, f)}
    O' = O' ∪ {out}
    update(δ', t)
    update(λ', t)
endfor

//------------ Part 2: minimize M' --------//
M' = minimize(M')     //apply minimization algorithm

//------------ Part 3: Generate the test set --------//
Let m be the estimated number of states in the implementation
TS_HHSI = apply-HSI-Method(M', m)     //as previously described
TS_HHSI = removeFs(TS_HHSI)     //remove input f from test cases
```

Formally, any FSM can be reduced using direct application of FSM minimization (Grasselli and Luccio 1965; Pena and Oliveira 1998). However, as discussed, this causes the states that do not have the same set of enabled inputs, that is, incompatible states, to be merged in the reduced machine. Therefore, test cases generation from such a machine results in infeasible test cases. The introduction of *f*-transitions does not violate the definition of an FSM; thus, it does not prevent the machine from being reduced. The *f*-transitions only help to identify the compatible states properly so that the incompatible states are not merged during the minimization. As a consequence, the reduced FSM keeps the same set of defined sequences as the original and, therefore, it can be used for test generation properly.

Notice that, from an FSM minimization perspective, the inclusion of *f*-transitions has an effect similar to completing the FSM with loopback transitions for unspecified inputs and producing null outputs (Sidhu and Leung 1989). However, this completeness strategy will introduce much more transitions (proportional to $n.|I|$) and, as a consequence, the performance of minimization and test generation algorithms will decline significantly. Moreover, greater number of non-executable test cases will need to be removed in the following steps. In addition, without the proper treatment during the selection of separating families, the direct application of traditional methods will always choose the shortest sequences which usually include the artificially added transitions. As we discuss above, we treat all these issues in the HHSI method.

5 Case studies

This section describes the case studies conducted to compare the fault domain-based testing approach proposed in Sect. 4, that is, the heuristic HSI method, to random testing and simple coverage-based testing. We aim to evaluate the performance of the HHSI method with respect to cost (through test set characteristics) and effectiveness (through the fault detection ratio).

First, we present three SUCs, which represent important facilities provided by a large commercial Web-based system. Later, the experiment details are outlined. After performing the experiments, we analyze the results of (model-based) mutation analysis to evaluate fault detection and simple cost-effectiveness of test sets generated by the test generation methods (namely HHSI, random, and state/transition coverage). Finally, we discuss the results of the experiments and limitations of the approach.

5.1 Systems under consideration

ISELTA (Isik's System for Enterprise-Level Web Centric Tourist Applications—http://www.iselta.de) is a Web portal for marketing tourist services. It enables travel and tourist enterprises, such as hotel owners and agencies, to create their own individual search and service offering masks. These masks can be embedded in the existing homepage of the hotels as an interface between customers and system. Potential customers can then use those masks to select and book hotel rooms and benefit from various different facilities. We use three non-trivial facilities of ISELTA as SUCs. Therefore, the test models are built using the following three facilities available in ISELTA: Specials, Additionals, and Prices.

Through Specials, a hotel owner or a travel agent is able to add special prices to the marketed hotel. To add a special, at least the room type, number of rooms of this type, basic price, and time period information should be provided, together with a unique name for the special. One can also upload a photo or write additional descriptions. Using this facility, the existing specials can also be edited or deleted.

Additionals provides functionalities to manage offerings of additional facilities, such as extra beds or extra rooms in specified periods and service days. To add an additional service, at least the period, service days, room type, amount per day, price, and a unique name should be provided. Optionally, descriptions and photos can be included, and existing additional services can also be edited or deleted using this facility.

In addition, using Prices, hotel owners or travel agencies can define reduced or additional prices per person based on, for example, number of children, number of persons, duration of the stay, and/or some specific dates. To define a price, at least a unique name and the price should be entered. In addition, if discount is selected, some additional data like age group and number of persons have to be entered. Also, existing prices can be edited or deleted using this facility.

For more information on the SUCs, reader may refer to Appendix A of ESM.

5.2 Experimental configuration

For each SUC, inputs are identified by listing relevant user actions and outputs are identified by considering the observed Web pages and certain relevant elements in these Web pages. More precisely, inputs are identified using the events that can be performed by the user in different phases of system activity; for example, different types of data entering and canceling events are distinguished from each other by using slightly different labels.

Outputs are characterized by considering certain properties of the list elements and the form elements in the pages; the properties of list elements considered are the number of current list elements, the change in this number, and the presence of a locked element in the list; and the properties of form elements considered are the page type, the completeness of field elements, and the presence of warning messages, warning pop-ups and delete pop-ups. Later, states are identified carefully by using the possible input–output combinations to build a correct transition function.

Based on these artifacts, FSM models are derived to employ the fault domain-based approach developed in Sect. 4. After the construction of the initial, redundant FSM models, reduced machines are computed automatically for test generation. For the sake of saving space, we include the FSM models in Appendix B.

Using the heuristic HSI method (Sect. 4) on FSM models, n-complete, $(n + 1)$-complete, and $(n + 2)$-complete test sets are generated. We refer to these three test sets as HHSI(n), HHSI($n + 1$), and HHSI($n + 2$), respectively.

After generating HHSI(n), HHSI($n + 1$), and HHSI($n + 2$), considering the size of these test sets, we generate three random test sets which have approximately the same sizes as the HHSI test sets, namely Random(n), Random($n + 1$), and Random($n + 2$), respectively. To assure that the corresponding fault domain-based and random test sets have approximately the same size, we generate random test sets to satisfy the following properties. For each Random(i) where $i = n, n + 1, n + 2$:

- Each test case in Random(i) has length X, where X is the smallest integer larger than or equal to the average length of the test cases in HHSI(i).
- Random(i) contains Y test cases, where Y is the smallest integer larger than or equal to ||HHSI(i)||/X, where ||HHSI(i)|| is the sum of all test case lengths in HHSI(i).

In this way, the size of Random(i), $X.Y$, is very close to the size of HHSI(i), for $i = n, n + 1, n + 2$.

We also generate two additional test sets using conventional FSM-based coverage criteria. More precisely, two test sets are additionally generated by covering states and transitions [using the testing tree in (Chow 1978)], referred to as state (cover) and transition (cover), respectively. Furthermore, random test sets having approximately the same size as these test sets are generated following the methodology similar to the above. These random test sets are referred to as Random(state) and Random(transition), respectively.

To obtain average trends for random testing approach, in each case study, we generate and use 10 random test sets for each Random(i) where $i = n, n + 1, n + 2$, *state*, and *transition*. Figure 6 summarizes the process performed for each SUC to obtain the test model and generate the test sets.

The effectiveness of the test sets, generated as shown in Fig. 6, is compared in two steps using mutation analysis. Figure 7 illustrates the mutation analysis process adopted in the case studies. For each SUC, mutants are generated using two different types of FSM models:

- Reduced FSM models, which are also used for generating HHSI test sets: This induces a situation where the models and the SUCs have the same number of states, which may not always be the case in practice.
- Non-reduced FSM models: This induces a situation where the models and the SUCs have different number of states and the difference is unknown. For instance, the tester does not have access to the internals of the system and/or is incapable of determining the exact number of states in the system.

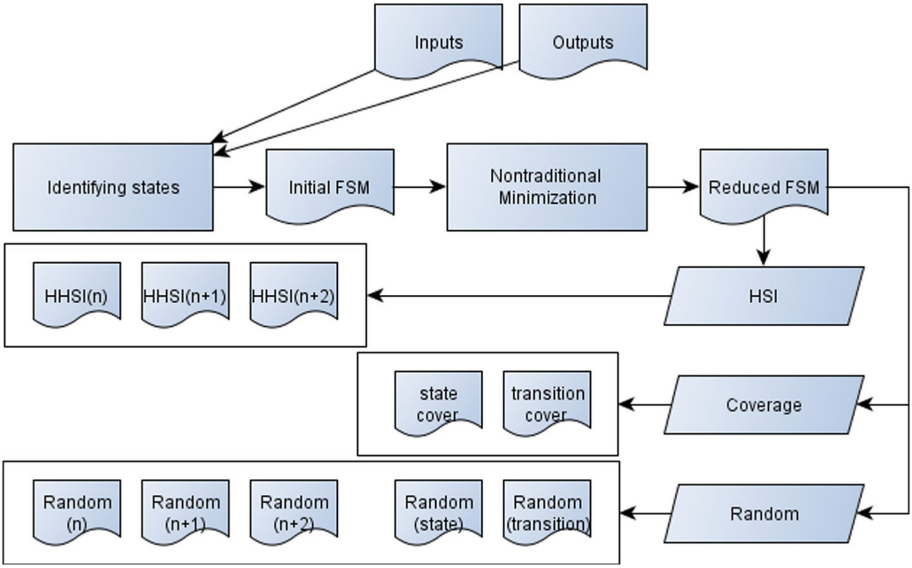

Fig. 6 Modeling and test set generation

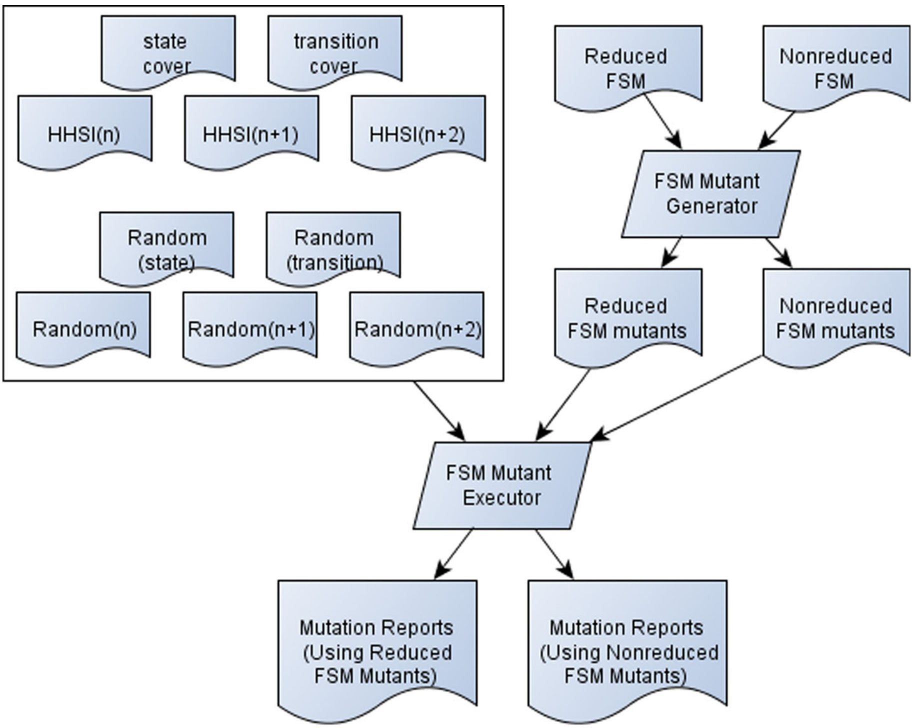

Fig. 7 Mutation analysis

Note that it is important whether corresponding models and SUCs have the same number of states or not, because it directly affects the size of the fault domain and the power of the generated test cases. In the end, for each generated test set, two sets of mutation scores are calculated.

5.3 Mutation operators

Mutation operators introduced in this subsection are used to seed faults into FSM specifications. The following operators are defined in (Fabbri et al. 1994; Simao et al. 2008) for FSMs:

- *Change Initial State* (CIS): The initial state of the FSM is changed to a different state.
- *Change Input* (CI): Input of a transition is changed to a different one. The operator is not applied if the change produces a non-deterministic machine (mutant).
- *Change Output* (CO): Output of a transition is changed to a different one.
- *Missing Transition* (MT): A given transition is removed from the FSM.
- *Tail State Exchange* (TSE): Tail state of a transition is changed to a different one.
- *Head State Exchange* (HSE): Head state of a transition is changed to a different one. The operator is not applied if the change produces a non-deterministic machine (mutant).

In mutation analysis, these operators are applied one by one to the original specification and a set of first-order mutants are generated. During test execution, outputs obtained from a mutant are compared to the outputs obtained from the original FSM. If a test case reveals a mismatch between these outputs, the mutant is *killed*; otherwise, it is *alive*. Thus, mutation score *ms* is calculated as

$$ms = \#km/(\#tm - \#em)$$

where *#tm* is the total number of mutants, *#km* is the number of killed mutants, and *#em* is the number of equivalent mutants. The equivalent mutants are identified automatically by a polynomial time algorithm that searches for a separating sequence between the initial states of the original and mutant FSMs (Lee and Yannakakis 1996).

As mentioned, mutation analysis is performed using both reduced and non-reduced FSM models for the test sets generated using fault domain-based, coverage-based, and random testing approaches (See Sect. 5.2). Non-reduced models naturally contain extra states; therefore, we opted not to use state adding mutations to include states artificially.

5.4 Test models and test sets

This subsection presents the data about the adopted test models and generated test sets. Table 1 presents general information about reduced and non-reduced FSM models for each SUC. FSM models of Specials have the smallest size, whereas the models of Prices are the largest ones. Also, for each SUC, a non-reduced FSM model represents the model which is constructed initially from the system specification. It contains redundancy; therefore, its size is bigger.

Besides the number of states, transitions, inputs, and outputs, Table 1 also shows the averages (and the standard deviations) of the numbers of incoming transitions and of outgoing transitions. The number of outgoing transitions shows less variation (in all models, it varies from 0 to 8 transitions). The number of incoming transitions has more variation, as can be seen by the standard deviation values given in the parentheses. In

Table 1 Data related to models

Model	Model element	Specials	Additionals	Prices
FSM (reduced)	States	71	75	103
	Transitions	305	377	533
	Inputs	13	14	14
	Outputs	54	57	43
	Incoming transitions	4.3 (4.3)	5.0 (5.2)	5.2 (6.1)
	Outgoing transitions	4.3 (1.9)	5.0 (2.3)	5.2 (2.2)
FSM (non-reduced)	States	91	94	148
	Transitions	428	512	828
	Inputs	13	14	14
	Outputs	54	57	43
	Incoming transitions	4.7 (4.5)	5.4 (5.8)	5.6 (7.2)
	Outgoing transitions	4.7 (1.9)	5.4 (2.2)	5.6 (2.2)

addition, we have observed that most of the states have from 1 to 10 incoming transitions (around 90 %), while a relatively small percentage of states have from 11 to 49 incoming transitions (around 10 %).

Table 2 presents the information about the test sets generated using different methods for each SUC, as already explained in Sect. 5.2. The size of each test set is measured by its total length, which is the sum of the length of the test cases in the test set. This can be used as a rough estimate for testing cost, because it gives an idea of how many events need to be executed in total since all events have approximately the same execution cost/time. The size order given in Fig. 8 is observed for the test sets generated for the three SUCs.

The differences between HHSI(n), HHSI($n + 1$), and HHSI($n + 2$) (and their corresponding random test sets) are expected, since there is an subsumption relation (Zhu et al. 1997) between HHSI(n), HHSI($n + 1$), and HHSI($n + 2$) (and the random test sets are generated correspondingly to have approximately the same sizes). Table 2 also shows that, except for random test sets, the shortest test case lengths are the same for all models and test sets and the longest test case lengths are very close. The length of the longest and the average test case grow approximately by one for HHSI test sets, and the standard deviation is also low. In addition, for each random test set, the shortest, the longest, and the average test case length values are all equal, because random test cases have fixed length.

For the three SUCs (Specials, Additionals, Prices), all separating sequences (which compose the separating families as defined in Sect. 3.1) have length 1, i.e., one input event is sufficient to distinguish state pairs in these models, probably due to the fact that the outputs of the SUCs are in the form of web pages and, thus, they are rich of information, which makes it easier to distinguish the states without requiring longer separating sequences. Furthermore, the FSMs of the three SUCs used in the experiments have the same diameter, that is, the minimum number of inputs required to reach the state farthest from the initial state. The diameter of each FSM is equal to 12.

5.5 Mutation analysis

We herein present the results of mutation analysis to evaluate the fault detection power of the generated test sets using reduced and non-reduced FSM models. Table 3 presents the

Table 2 Data related to test sets

Test set	Total length	Number of test cases	Shortest test case length	Longest test case length	Average test case length/ standard deviation
Specials					
State	260	35	3	12	7.43/2.2
Transition	1,992	236	3	13	8.44/2.1
HHSI (*n*)	6,755	714	3	14	9.46/2.0
HHSI (*n* + 1)	39,699	3,725	3	15	10.66/2.0
HHSI (*n* + 2)	223,102	18,887	3	16	11.81/1.9
Random(state)	264	33	8	8	8/0
Random(transition)	1,998	222	9	9	9/0
Random (*n*)	6,760	676	10	10	10/0
Random (*n* + 1)	39,699	3,609	11	11	11/0
Random (*n* + 2)	223,104	18,592	12	12	12/0
Additionals					
State	276	38	3	12	7.26/2.3
Transition	2,359	282	3	13	8.37/2.1
HHSI (*n*)	8,549	918	3	14	9.31/2.1
HHSI (*n* + 1)	57,003	5,435	3	15	10.49/2.0
HHSI (*n* + 2)	363,262	31,270	3	16	11.62/2.0
Random(state)	280	35	8	8	8/0
Random(transition)	2,367	263	9	9	9/0
Random (*n*)	8,550	855	10	10	10/0
Random (*n* + 1)	57,013	5,183	11	11	11/0
Random (*n* + 2)	363,264	30,272	12	12	12/0
Prices					
State	368	50	3	12	7.36/2.0
Transition	3,522	432	3	13	8.15/2.0
HHSI (*n*)	15,086	1,656	3	14	9.11/1.9
HHSI (*n* + 1)	101,353	9,920	3	15	10.22/1.9
HHSI (*n* + 2)	667,050	58,970	3	16	11.31/1.9
Random(state)	368	46	8	8	8/0
Random(transition)	3,528	392	9	9	9/0
Random (*n*)	15,090	1,509	10	10	10/0
Random (*n* + 1)	101,354	9,214	11	11	11/0
Random (*n* + 2)	667,056	55,588	12	12	12/0

Fig. 8 Test set sizes

state ≈ Random(state) <
transition ≈ Random(transition) <
HHSI(n) ≈ Random(n) <
HHSI(n+1) ≈ Random(n+1) <
HHSI(n+2) ≈ Random(n+2)

mutation scores obtained using the reduced FSM models as implementation models to generate the mutants [for each system under consideration (SUC)]. Note that, since reduced FSM models are also used for generation of HHSI(n), HHSI($n + 1$), and HHSI($n + 2$) test sets, an original model and each mutant generated from this model have the same number of states.

During mutation analyses, all CIS, CI, CO, MT, and HSE mutants are killed by HHSI test sets. TSE mutants turned out to be the hardest-to-kill mutants, presenting live mutants for all the test sets. Also, no equivalent mutants were observed.

Overall, 51,782 mutants were generated for Specials, 66,406 mutants for Additionals, and 108,104 mutants for Prices, and none of the test sets managed to achieve the perfect mutation score for any of the SUCs.

Random test sets achieve lower mutation scores than their non-random counterparts. The lowest mutation scores are achieved by Random(state) with 11.99 to 14.15 %. They are followed by state covers (14.82 to 18.37 %), Random(transition) (24.50 to 27.22 %), Random(n) (35.88 to 40.41 %), Random($n + 1$) (53.49 to 59.83 %), transition cover (59.20 to 68.06 %), and Random($n + 2$) (67.95 to 80.46 %).

HSI test sets have significantly superior mutation scores when compared with the other test sets with HHSI($n + 1$) and HHSI($n + 2$) achieving exactly the same and the highest scores (99.29 to 99.41 %), and HHSI(n) is following them with the scores between 99.12 and 99.33 % by killing 36, 113, and 87 fewer mutants for Specials, Additionals, and Prices, respectively.

Thus, we can order the test sets using the averages of the mutation scores achieved over different SUCs as shown in Fig. 9.

Table 4 presents the mutation scores obtained using the non-reduced FSM models as implementation models to generate the mutants (for each SUC). In this case, an original model and each mutant obtained from this model have different number of states. This represents a situation which decreases the power of the heuristic HSI method. However, the results show a similar trend with those obtained using reduced FSM models as explained above.

During mutation analyses, all CI, CO, MT, and HSE mutants were detected by HHSI($n + 1$) and HHSI($n + 2$) test sets for Specials and Additionals. Furthermore, this time, equivalent mutants were observed but only for TSE operator, which turned out to generate the hardest-to-kill mutants once again.

Overall, 82,736 mutants were generated for Specials, 101,287 mutants for Additionals, and 216,181 mutants for Prices, representing a larger fault domain with almost twice the number of mutants obtained from reduced FSMs. None of the test sets managed to achieve the perfect mutation score for any of the SUCs.

Once again non-random test sets achieve higher mutation scores than their random counterparts. The lowest mutation scores are achieved by Random(state), 9.01 to 10.86 %. They are followed by state covers (9.31 to 12.82 %), Random(transition) (19.13 to 22.39 %), Random(n) (31.12 to 34.79 %), Random($n + 1$) (45.76 to 52.38 %), transition covers (35.47 to 46.46 %), and Random($n + 2$) (66.14 to 71.88 %).

HHSI($n + 2$) test sets achieve the highest mutation scores in range 92.09–99.04 % where HHSI($n + 1$) test sets manage to achieve second to best mutation scores with 88.45–99.04 %. HHSI(n) test sets attain mutation scores in the interval from 70.01 to 83.21 %, falling significantly behind of HHSI($n + 1$) and HHSI($n + 2$).

When we order the test sets using the averages of the mutation scores achieved over different SUCs (Fig. 10), we have an ordering similar to that obtained using reduced FSM

Table 3 Mutation scores over reduced FSM models

Test set	CIS	CI	CO	MT	TSE	HSE	Alive/killed	Total score
Specials								
State	1.00000	0.22644	0.22951	0.22951	0.11363	0.23412	42,272/9,510	0.18366
Transition	1.00000	1.00000	1.00000	1.00000	0.22529	1.00000	16,540/35,242	0.68058
HHSI (n)	1.00000	1.00000	1.00000	1.00000	0.98319	1.00000	359/51,423	0.99307
HHSI ($n + 1$)	1.00000	1.00000	1.00000	1.00000	0.98487	1.00000	323/51,459	0.99376
HHSI ($n + 2$)	1.00000	1.00000	1.00000	1.00000	0.98487	1.00000	323/51,459	0.99376
Random(state)	1.00000	0.16939	0.15279	0.15279	0.12809	0.13928	44,453.8/7,328.2	0.14152
Random (transition)	1.00000	0.31489	0.29541	0.29541	0.24285	0.28024	37,685.6/14,096.4	0.27223
Random (n)	1.00000	0.44103	0.41738	0.41738	0.35295	0.40734	31,562.8/20,219.2	0.39047
Random ($n + 1$)	1.00000	0.63666	0.61574	0.61574	0.53187	0.60597	21,723.6/30,058.4	0.58048
Random ($n + 2$)	1.00000	0.79750	0.77869	0.77869	0.71594	0.77046	12,833.5/38,948.5	0.75216
Additionals								
State	1.00000	0.19727	0.19629	0.19629	0.09460	0.20869	55,973/10,433	0.15711
Transition	1.00000	0.94302	0.94165	0.94165	0.19252	0.93373	24,876/41,530	0.62540
HHSI (n)	1.00000	1.00000	1.00000	1.00000	0.97928	1.00000	578/65,828	0.99130
HHSI ($n + 1$)	1.00000	1.00000	1.00000	1.00000	0.98333	1.00000	465/65,941	0.99300
HHSI ($n + 2$)	1.00000	1.00000	1.00000	1.00000	0.98333	1.00000	465/65,941	0.99300
Random (state)	1.00000	0.15242	0.13316	0.13316	0.11598	0.12045	58,098.2/8,307.8	0.12511
Random (transition)	1.00000	0.28431	0.26340	0.26340	0.22451	0.24516	50,136.7/16,269.3	0.24500
Random (n)	1.00000	0.40733	0.38196	0.38196	0.33001	0.36669	42,581.8/23,824.2	0.35877
Random ($n + 1$)	1.00000	0.59487	0.56923	0.56923	0.49069	0.55513	30,884.7/35,521.3	0.53491
Random ($n + 2$)	1.00000	0.74485	0.71797	0.71797	0.64033	0.70099	32,366.2/68,607.8	0.67946
Prices								
State	1.00000	0.19562	0.19137	0.19137	0.09657	0.20603	92,081/16,023	0.14821
Transition	1.00000	1.00000	1.00000	1.00000	0.18865	1.00000	44,110/63,994	0.59197
HHSI (n)	1.00000	1.00000	1.00000	1.00000	0.98657	1.00000	730/107,374	0.99325

Table 3 continued

Test set	CIS	CI	CO	MT	TSE	HSE	Alive/killed	Total score
HHSI ($n + 1$)	1.00000	1.00000	1.00000	1.00000	0.98817	1.00000	643/107,461	0.99405
HHSI ($n + 2$)	1.00000	1.00000	1.00000	1.00000	0.98817	1.00000	643/107,461	0.99405
Random (state)	1.00000	0.14613	0.13058	0.13058	0.11093	0.12168	95,138.4/12,965.6	0.11994
Random (transition)	1.00000	0.30136	0.28124	0.28124	0.23491	0.26971	80,367.6/27,736.4	0.25657
Random (n)	1.00000	0.45742	0.43490	0.43490	0.37587	0.42459	64,420.2/43,683.8	0.40409
Random ($n + 1$)	1.00000	0.65883	0.63884	0.63884	0.56009	0.63052	43,423.9/64,680.1	0.59831
Random ($n + 2$)	1.00000	0.84649	0.83190	0.83190	0.77794	0.82832	21,122.8/86,981.2	0.80461

Fig. 9 Mutation scores over reduced FSM models

Random(state) < state < Random(transition) < Random(n) < Random(n+1) < transition < Random(n+2) < HHSI(n) < HHSI(n+1) = HHSI(n+2)

models. The only difference is that HHSI($n + 1$) test sets achieve smaller mutation scores than HHSI($n + 2$) for each SUC.

As already mentioned, using non-reduced FSMs causes the fault domain to grow, and, in general, all test sets experience a decrease in mutation scores. If we express the average decrease for each test set relative to its average mutation score obtained using non-reduced FSM models, we have the followings: State and transitions covers suffer from relatively greater decreases where, on the average, transition covers experience a relative decrease of $\sim$33.62 % and state covers a relative decrease of $\sim$31.51 %. These test sets are followed by Random(state), HHSI(n), Random(transition), Random(n), Random($n + 1$), and Random($n + 2$) with relative decreases of $\sim$23.04, $\sim$22.02, $\sim$19.94, $\sim$15.20, $\sim$14.30, and $\sim$7.90 %, respectively. HHSI($n + 1$) and HHSI($n + 2$) are the test sets which have the smallest relative decreases in mutation scores with $\sim$3.97 and $\sim$2.20 %, respectively. This shows that HHSI($n + 1$) and HHSI($n + 2$) are relatively more tolerant to changes in fault domain which is caused by the use of redundant models.

Furthermore, let us consider the test set size as a rough estimate for the testing costs and calculate the number of mutants revealed per event by using the ratio of average mutation scores to test set sizes for each test set and each SUC. If we use the average of these ratios for the corresponding random and non-random test sets, we obtain the inequalities in Fig. 11 for both of the cases where reduced and non-reduced FSMs are used. This order suggests that non-random test sets are likely to attain better performance with respect to their random counterparts in testing.

5.6 Analysis of the results

In the light of the data presented and discussed in Sects. 5.4 and 5.5, we can briefly state the following results.

The adaptation of the HSI method is successful in the sense that the corresponding test sets achieve higher mutation scores and manage to detect more faults than random test sets and coverage-based test sets in mutation analyses. However, due to the nature of fault domain-based testing methods, the test sets generated using the heuristic HSI method have greater sizes than coverage-based test sets. Furthermore, although we selected and used random test sets having approximately the same sizes, test execution costs associated with HHSI test sets turn out to be slightly greater.

The change in the fault domain (caused by the difference between the number of states in the SUC and the number of states in the model of the SUC) has a non-negligible negative effect on fault detection performance of the generated test sets. HHSI test sets suffer less from this negative effect with respect to their random counterparts, for sufficiently large parameter values, because the effect becomes less apparent when the value of method parameter m is increased.

Although state and transition covers are quite commonly used in MBT with transition-based models (Utting and Legeard 2006), our experiments show that they tend to miss more faults. Using these methods, the tester can be risking leaving out 50 % of the faults in the domains we defined.

Table 4 Mutation scores over non-reduced FSM models

Test set	CIS	CI	CO	MT	TSE	HSE	Alive/Killed	Total score
Specials								
State	1.00000	0.16811	0.16355	0.16355	0.08134	0.17117	71,837/10,563	0.12819
Transition	1.00000	0.74242	0.71262	0.71262	0.16138	0.74027	44,115/38,285	0.46462
HHSI (n)	1.00000	0.86966	0.85280	0.85280	0.69841	0.85948	13,838/64,562	0.83206
HHSI ($n + 1$)	1.00000	1.00000	1.00000	1.00000	0.97913	1.00000	797/81,603	0.99033
HHSI ($n + 2$)	1.00000	1.00000	1.00000	1.00000	0.97918	1.00000	795/81,605	0.99035
Random (state)	1.00000	0.13752	0.11893	0.11893	0.09937	0.10521	73,451.6/8,948.4	0.10860
Random (transition)	1.00000	0.27319	0.24953	0.24953	0.19840	0.23256	63,949/1,8451	0.22392
Random (n)	1.00000	0.40492	0.37710	0.37710	0.31733	0.36192	53,733.2/28,666.8	0.34790
Random ($n + 1$)	1.00000	0.59133	0.56449	0.56449	0.48052	0.54910	39,240.5/43,159.5	0.52378
Random ($n + 2$)	1.00000	0.77836	0.75724	0.75724	0.67977	0.74041	23,170.7/59,229.3	0.71880
Additionals								
State	1.00000	0.15094	0.14453	0.14453	0.06985	0.15939	89,504/11,470	0.11359
Transition	1.00000	0.72157	0.69336	0.69336	0.14206	0.71351	56,489/44,485	0.44056
HHSI (n)	1.00000	0.86538	0.84766	0.84766	0.71655	0.86059	21,236/79,738	0.78969
HHSI ($n + 1$)	1.00000	1.00000	1.00000	1.00000	0.97377	1.00000	1,241/99,733	0.98771
HHSI ($n + 2$)	1.00000	1.00000	1.00000	1.00000	0.97408	1.00000	1,226/99,748	0.98786
Random (state)	1.00000	0.12644	0.10645	0.10645	0.09223	0.09370	90,997.5/9,976.5	0.09880
Random (transition)	1.00000	0.25054	0.22637	0.22637	0.18412	0.20687	80,347.7/20,626.3	0.20427
Random (n)	1.00000	0.37481	0.34629	0.34629	0.29226	0.32774	68,770.5/32,203.5	0.31893
Random ($n + 1$)	1.00000	0.55467	0.71797	0.71797	0.64033	0.70099	51,765.2/49,208.8	0.48734
Random ($n + 2$)	1.00000	0.74485	0.71797	0.71797	0.64033	0.70099	32,366.2/68,607.8	0.67946
Prices								
State	1.00000	0.13195	0.12319	0.12319	0.06243	0.13635	195,288/20,057	0.09314
Transition	1.00000	0.67454	0.64372	0.64372	0.12184	0.65505	138,969/76,376	0.35467
HHSI (n)	1.00000	0.80408	0.78382	0.78382	0.63438	0.78178	64,567/150,778	0.70017

Table 4 continued

Test set	CIS	CI	CO	MT	TSE	HSE	Alive/Killed	Total score
HHSI ($n + 1$)	1.00000	0.91144	0.90217	0.90217	0.87218	0.89745	24,868/190,477	0.88452
HHSI ($n + 2$)	1.00000	0.95082	0.94444	0.94444	0.90445	0.93906	17,033/19,8312	0.92090
Random (state)	1.00000	0.11628	0.09891	0.09891	0.08425	0.09188	195,946.2/19,398.8	0.09008
Random (transition)	1.00000	0.23724	0.21184	0.21184	0.17705	0.20242	174,153/41,192	0.19128
Random (n)	1.00000	0.36786	0.33768	0.33768	0.29218	0.32834	148,332.7/67,012.3	0.31119
Random ($n + 1$)	1.00000	0.52306	0.49046	0.49046	0.43293	0.48286	116,802.3/98,542.7	0.45760
Random ($n + 2$)	1.00000	0.72604	0.69940	0.69940	0.63179	0.69512	72,920.5/142,424.5	0.66138

Fig. 10 Mutation scores over non-reduced FSM models

Random(state) < state < Random(transition) <
Random(n) < Random(n+1) < transition < Random(n+2) <
HHSI(n) < HHSI(n+1) < HHSI(n+2)

Fig. 11 Approximate testing costs based on test set sizes

Random(state) < state,
Random(transition) < transition,
Random(n) < HHSI(n),
Random(n+1) < HHSI(n+1), and
Random(n+2) < HHSI(n+2)

5.7 Threats to validity and discussion

As for every study with an experimental component, we also suffer from certain issues explained as below.

First, we selected and used three SUCs to perform the experiments and obtain results. Some results may have been affected by unique inherent properties of these SUCs, which we are not aware of. However, we tried to reduce this threat by discussing our results based on averages and/or minimum and maximum values; we mainly demonstrated common trends, relations, and relative changes, instead of focusing on specific values.

The ISELTA system was developed in collaboration of Isik Touristic (www.isik.de) with the Software Engineering Department of the University of Paderborn. We recognize a possible threat here given the fact that two of the authors are from this department. This threat was mitigated because (i) the tester who designed the models is neither a developer nor one of the authors and (ii) the researchers that performed the experiments were not involved in the development either.

For each SUC, we used a single FSM model, which is the initial FSM model constructed from the system specifications, to create a redundant system model to represent a situation where the SUC and its model have different number of states. However, a deterministic FSM model may have infinitely many equivalent but redundant deterministic FSM models. Unfortunately, to our knowledge, there is no systematic way to generate a sufficiently large finite subset of these redundant models by controlling the number of states. Also, there is no prior work that studies which of these models are likely to occur in practice, and, thus, realistic to use. Still, we believe that our selection of redundant FSM models makes sense, at least, to gain some insights into such situations.

In our analyses, all input events have approximately the same cost of execution. This means that when the test set is considered as a whole, each event has similar execution time on average. In practice, each system has its characteristics and some events can run faster or slower. To handle this challenge, the test methods have to take into account input events with weights, which is out of the scope of this paper.

We generated random test cases with fixed length to have test sets whose sizes are similar to the non-random test sets used in the case studies. Thus, the size served as a common criterion to approximately equalize the test execution efforts. Another and probably more sensible common criterion would be to generate m-complete test sets randomly. However, to our knowledge, there is no such random test generation method. Therefore, our results hold for a specific type of random test sets, which still belong to a substantially large pool of random test cases.

One may argue that HHSI should be compared with test case generation methods "stronger" than state/transition coverage and random testing. There are three main reasons for the comparisons we conducted. First, HHSI test sets cover all states and transitions by definition. As a consequence, we can reason that HHSI test sets have, at least, the same fault detection capability of state/transition cover test sets. However, this comparison is still valid since as these coverage criteria are widely used in practice (Utting and Legeard 2006), practitioners may be interested in the gains of adopting a new method like HHSI with respect to these widely used criteria. Second, random testing is frequently used as a reference in software testing (Juristo et al. 2004). The presented results have provided evidences that test sets generated by HHSI outperform random test sets with similar sizes. This eliminates the argument that more faults were detected just because the test sets are arbitrarily greater. Finally, there are stronger methods as already discussed, like W, Wp, HSI, H, and SPY. These methods are able to generate m-complete test sets but it is mandatory that the FSM model is reduced. As HHSI is intended to work with non-reduced machines, a direct comparison is not possible. When HHSI is applied to reduced FSMs, the method performs just like the original HSI and f-transitions (and other steps in Sect. 4) will have no effect. Detailed comparisons involving HSI and other methods can be found in (Dorofeeva et al. 2010; Endo and Simao 2013).

Another method which is similar to the method we propose is the SC method (Petrenko and Yevtushenko 2005) (see also Sect. 2). The SC method also works with non-reduced FSMs and yet it is able to generate m-complete test sets. As a drawback, it can be unfeasible when applied to FSMs with several equivalent states. Formally, equivalent states are distinguished by using traversal sets that are expanded until states are covered m times in each branch; this step can lead to sequences with up to $\sim m.k$ symbols, where k is the maximum number of states which are pairwise equivalent ($k = 1$ if all states are distinguishable). As for each prefix of such sequences, there may exist l distinct inputs, where l is the number of inputs. Thus, in the worst case, a test set of size $O(l^{m.k})$ inputs is generated. For example, an n-complete test set with the total length of 1,892 inputs is generated when the SC method is applied to the machine in Fig. 2. The size of this test set is very large when compared with the n-complete test set generated using the HHSI method from the same machine, which has the total length of only 32 inputs (see Sect. 4—Part 3). Furthermore, our implementation of SC was not able to produce a test set even for a small portion of Specials (an FSM with 12 states). These results prevented us from including the SC method in our comparisons since the method would not scale for the models used in the experiments, which have 91, 94, and 148 states. In future, we intend to improve our SC implementation so that the scalability can be increased, enabling comparisons between HHSI and SC.

6 Conclusion and future work

In this paper, a heuristic application of the fault domain-based HSI method has been presented. We consider that the conditions required by fault domain-based methods may not be feasible to be satisfied in practice and, therefore, certain workarounds may be needed to apply these methods to imperfect situations. The method we have proposed is (i) designed for the systems which are not input enabled, that is, not all the inputs can be performed in every state, and (ii) based on the idea of minimizing the FSM while keeping the same defined input sequences. To do this, an initial FSM model is augmented with auxiliary transitions which prevent the states with disjoint sets of defined inputs from being merged during minimization. After performing the minimization, the HSI method is applied to generate test sets and inputs in these test sets corresponding to the inserted transitions are removed to assure that test cases are executable.

Since the heuristic HSI method is not guaranteed to detect all the faults in the defined fault domain, we also conducted extensive experiments to compare it to random testing and commonly used coverage-based testing methods. The comparison was performed in two different situations where (i) the test model and the SUC have the same number of states, and (ii) they have a different number of states. In the former, test sets generated using the heuristic HSI method revealed on average 15 to 352 % more faults and achieved on the average 25 to 107 % higher fault detection rates than their corresponding random counterparts. In the latter, test sets generated using the heuristic HSI method revealed on average 22 to 387 % more faults and achieved on average 28 to 90 % higher fault detection rates than their corresponding random counterparts. The results provided evidences that, in both cases, the proposed method managed to detect more faults than other methods and also achieved better fault detection rates than the random testing method when the respective test execution efforts are approximately equal.

As for the future work, the issues pointed out by the threats to validity can be used to improve the results. For example, it would be interesting to devise a method which systematically generates non-reduced FSM models from a given reduced deterministic FSM model to induce other types of imperfect situations. In this way, one can analyze the results considering the difference in the number of states.

Also, a comparison to different types of random testing methods can also be performed to extend the validity of the results. Instead of, or in addition to, approximately equalizing the size of the test sets, different test set metrics, such as coverage or distribution of the length of the test cases, can be used in the generation of random test sets.

Last but not the least, heuristic versions of other fault domain-based testing methods can be developed for adaptations to imperfect situations. Similar analyses can be performed in comparison with test generation methods which are not fault domain-based, such as Belli (2001), and Belli and Beyazit (2010).

References

Belli, F. (2001). Finite-state testing and analysis of graphical user interfaces. In *The 12th international symposium on software reliability engineering (ISSRE 2001)* (pp. 34–43).

Belli, F., & Beyazıt, M. (2010). A formal framework for mutation testing. In *The 4th international conference on secure software integration and reliability improvement (SSIRI 2010)* (pp. 121–130).

Björklund, H. (2005). State verification. In M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, & A. Pretschner (Eds.), *Model-based testing of reactive systems* (pp. 69–86). Lecture Notes in Computer Science, Vol. 3472, 659 p., ISBN 978-3-540-26278-7. New York: Springer.

Chow, T. S. (1978). Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering, 4*(3), 178–187.

Dorofeeva, R., El-Fakih, K., Maag, S., Cavalli, A. R., & Yevtushenko, N. (2010). FSM-based conformance testing methods: A survey annotated with experimental evaluation. *Information and Software Technology, 52*(12), 1286–1297.

Dorofeeva, R., El-Fakih, K., & Yevtushenko, N. (2005). An improved conformance testing method. In *Formal techniques for networked and distributed systems (FORTE 2005)* (pp. 204–218).

Endo, A. T., & Simao, A. S. (2013). Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. *Information and Software Technology, 55*(6), 1045–1062.

Fabbri, S. C. P. F., Delamaro, M. E., Maldonado, J. C., & Masiero, P. C. (1994). Mutation analysis testing for finite state machines. In *The 5th international symposium on software reliability engineering (ISSRE 1994)* (pp. 220–229).

Fujiwara, S., Bochmann, G. V., Khendek, F., Amalou, M., & Ghedamsi, A. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering, 17*(6), 591–603.

Gargantini, A. (2005). Conformance testing. In M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, & A. Pretschner (Eds.), Model-based testing of reactive systems (pp. 87–111). Lecture Notes in Computer Science, Vol. 3472, 659 p., ISBN 978-3-540-26278-7. New York: Springer.

Gill, A. (1962). *Introduction to the theory of finite-state machines*. Hill: McGraw.

Gonenc, G. (1970). A method for design of fault detection experiments. *IEEE Transactions on Computers, 19*(6), 551–558.

Grasselli, A., & Luccio, F. (1965). A method for minimizing the number of internal states in incompletely specified sequential networks. *IEEE Transactions on Electronic Computers, EC, 14*(3), 350–359.

Hennie, F. C. (1964). Fault detecting experiments for sequential circuits. In *The annual IEEE symposium on foundations of computer science* (pp. 95–110).

Hierons, R. M. (2004). Testing from a nondeterministic finite state machine using adaptive state counting. *IEEE Transactions on Software Engineering, 53*(10), 1330–1342.

Hierons, R. M., Bogdanov, K., Bowen, J. P., Cleaveland, R., Derrick, J., Dick, J., et al. (2009). Using formal specifications to support testing. *ACM Computing Surveys, 41*(2), 1–76.

Hierons, R. M., & Ural, H. (2006). Optimizing the length of checking sequences. *IEEE Transactions on Computers, 55*(5), 618–629.

Juristo, N., Moreno, A. M., & Vegas, S. (2004). Reviewing 25 years of testing technique experiments. *Empirical Software Engineering, 9*(1–2), 7–44.

Krichen, M. (2005). State identification. In M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, & A. Pretschner (Eds.), *Model-based testing of reactive systems* (pp. 35–67). Lecture Notes in Computer Science, Vol. 3472, 659 p., ISBN 978-3-540-26278-7. New York: Springer.

Lai, R. (2002). A survey of communication protocol testing. *Journal of Systems and Software, 62*(1), 21–46.

Lee, D., & Yannakakis, M. (1996). Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE, 84*(8), 1090–1123.

Luo, G., Petrenko, A., & Bochmann, G. V. (1995). Selecting test sequences for partially-specified nondeterministic finite state machines. In *The 7th IFIP WG 6.1 international workshop on protocol test systems (IWPTS 94)* (pp. 95–110).

Mealy, G. H. (1955). A method for synthesizing sequential circuits. *Bell System Technical Journal, 34*(5), 1045–1079.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematics Series, 34*, 129–153.

Pena, J. M., & Oliveira, L. (1998). A new algorithm for the reduction of incompletely specified finite state machines. In *The international conference on computer-aided design (ICCAD 1998)* (pp.482-489).

Petrenko, A., & Yevtushenko, N. (2005). Testing from partial deterministic FSM specifications. *IEEE Transactions on Computers, 54*(9), 1154–1165.

Petrenko, A., & Yevtushenko, N. (2014). Adaptive testing of nondeterministic systems with FSM. In *The 15th international symposium on high-assurance systems engineering (HASE)* (pp. 224–228).

Rezaki, A., & Ural, H. (1995). Construction of checking sequences based on characterization sets. *Computer Communications, 18*(12), 911–920.

Sandberg, S. (2005). Homing and synchronizing sequences. In M. Broy, B. Jonsson, J. P. Katoen, M. Leucker, & A. Pretschner (Eds.), *Model-based testing of reactive systems* (pp. 5–33). Lecture Notes in Computer Science, Vol. 3472, 659 p., ISBN 978-3-540-26278-7. New York: Springer.

Sidhu, D. P., & Leung, T. K. (1989). Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering, 15*(4), 413–426.

Simao, A., & Petrenko, A. (2010). Checking completeness of tests for finite state machines. *IEEE Transactions on Computers, 59*(8), 1023–1032.

Simao, A., Petrenko, A., & Maldonado, J. C. (2008). Comparing finite state machine test coverage criteria. *IET Software, 3*(2), 91–105.

Simao, A., Petrenko, A., & Yevtushenko, N. (2009). Generating reduced tests for FSMs with extra states. In *International conference on testing of software and communication systems (TESTCOM)* (pp. 129–145).

Utting, M., & Legeard, B. (2006). *Practical model-based testing: A tools approach*. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Vasilevskii, M. P. (1973). Failure diagnosis of automata. *Cybernetics and Systems Analysis, 9*, 653–665.

Zhang, F., & Cheung, T. (2003). Optimal transfer trees and distinguishing trees for testing observable nondeterministic finite-state machines. *IEEE Transactions on Software Engineering, 29*(1), 1–14.

Zhu, H., Hall, P. A. V., & May, J. H. R. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys, 29*(4), 366–427.

Fevzi Belli is a professor of Software Engineering at the University of Paderborn, Germany. He completed his PhD in formal methods for verifying software systems and self-correction features in formal languages at Berlin Technical University in 1978. He spent several years as a software engineer in Munich, writing programs to test other programs, before he moved to the University of Paderborn in 1989. He has an interest and experience in software reliability/fault tolerance, model-based testing, and test automation.

Mutlu Beyazıt received his MSc degree in Computer Software from İzmir Institute of Technology (Turkey) in 2008. He also worked as a research assistant at İzmir Institute of Technology from 2005 to 2008. In 2009, he joined the Department of Computer Science, Electrical Engineering and Mathematics in the University of Paderborn and got his PhD in 2014 from there. His current interests lie in the area of model-based testing.

Andre Takeshi Endo received a bachelor's degree in Analysis of Systems from the Universidade Federal de Mato Grosso do Sul, Brazil, in 2005. He also holds MSc and PhD degrees in Computer Science from Universidade de Sao Paulo, Brazil, awarded in 2008 and 2013, respectively. He is currently an Assistant Professor at the Universidade Tecnologica Federal do Parana (UTFPR), Cornelio Procopio, Brazil. Since he left his hometown Alvares Machado in 2002, he has pursued his professional passions for computers, programming, and research. His area of interest includes software engineering with emphasis on software testing, model-based testing, service-oriented architectures, and web services.

Aditya Mathur is head of Pillar Information Systems Technology and Design at the Singapore University of Technology and Design and professor of Computer Science at Purdue University, USA. His research spans software testing, software reliability, cyber security, and process control. His work relates to investigations into the effectiveness of testing techniques and their applicability to the testing of sequential and distributed software systems, methods for the estimation of software system reliability, and techniques and tools for managing a collection of Internet-enabled devices. In software process control, Aditya's research group has formalized the feedback control of the software development process using methods from the discipline of automatic control. He is the author of several textbooks including "Introduction to Microprocessors–the first book on this subject in India, and "Foundations of Software Testing" which is now in its second edition. Aditya has consulted with and delivered numerous invited talks at academic and industrial forums.

Adenilso Simao received the BS degree in computer science from the State University of Maringa (UEM), Brazil, in 1998, and the MS and PhD degrees in computer science from the University of Sao Paulo (USP), Brazil, in 2000 and 2004, respectively. Since 2004, he has been a professor of computer science at the Computer System Department of USP. From August 2008 to July 2010, he has been on a sabbatical leave at Centre de Recherche Informatique de Montreal (CRIM), Canada. His research interests include software testing and formal methods. He is a member of the Brazilian Computer Society (SBC).