Checking Experiments for Finite State Machines with Symbolic Inputs

Alexandre Petrenko, Adenilso Simao

Checking Experiments for Finite State Machines with Symbolic Inputs

Alexandre Petrenko

CRIM, Centre de recherche informatique de Montréal
405 Ogilvy Avenue, Suite 101, Montréal (Québec) H3N 1M3, Canada
petrenko@crim.ca

Adenilso Simao

Instituto de Ciencias Matematicas e de Computacao, Universidade de Sao Paulo
Sao Carlos/Sao Paulo, Brazil
adenilso@icmc.usp.br

Abstract. There exists a significant body of work in the theory of checking experiments devoted to test generation from FSM which guarantees complete fault coverage for a given fault model. Practical applications require nevertheless methods for fault-model driven test generation from Extended FSMs (EFSM). Traditional approaches for EFSM focus on model coverage, which provides no characterization of faults that can be detected by the generated tests. Only few approaches use fault models, and we are not aware of any result in the theory of checking experiments for extended FSMs. In this paper, we lift the theory of checking experiments to EFSMs, which are Mealy machines with predicates defined over input variables treated as symbolic inputs. Considering this kind of EFSM, we propose a test generation method that produces a symbolic checking experiment, adapting the well-known HSI method. We then present conditions under which arbitrary instances of a symbolic checking experiment can be used for testing black-box implementations, while guaranteeing complete fault coverage.

Keywords. Finite State Machines; Extended Finite State Machines; Symbolic Automata; Conformance testing; Checking experiments; Fault model based test generation

1 Introduction

Research in Model Based Testing (MBT) is currently advancing rapidly trying to match the growing demand from industry for more effective and better scalable test development technologies. Since the cost for leaving undetected faults in software grows with its complexity, code and model coverage by tests is often considered insufficient and the guaranteed fault detection becomes the ultimate goal. Accordingly, research in MBT has been addressing fault modeling and fault model driven test gen-

eration problems, see, e.g., [2], [38]. Fault models usually refer to test models which formalize reference specifications and/or requirements. State-oriented test models seem to be most popular models among test engineers. Finite state machines (FSM) and input output transition systems (IOTS) are state-oriented models; test generation methods have traditionally been developed separately for these models, even though, as has already been demonstrated, many ideas, especially for fault model based test generation, developed for testing from FSM can successfully be used for testing from IOTS [31].

There exists a significant body of work devoted to the development of methods for test generation from a given FSM to guarantee the complete fault coverage, once a fault model is defined. The pioneering work of Moore [21] and Hennie [13] led to the development of the theory of checking experiments, where faults are modeled by a universe of FSMs with a given number of states, see, e.g., [4], [6], [10], [33]. Checking experiments have already been lifted to FSMs more general than the classical (completely specified and deterministic) Mealy machine, such as partially defined and nondeterministic state machines, see, e.g., [27]. However, practical applications require more extensions to the classical FSM model. These are commonly known as the Extended Finite State Machine (EFSM) models. Various flavors of EFSMs are used in Harel's statecharts [12], SysML/UML [9], Simulink/Stateflow [32], SDL [11] and other modelling languages. Extensions are often suggested without a formal semantics; this creates a big hurdle for fault-model based test generation. Whenever the semantics of a particular specification language is defined by the tool which supports it, fault models become specific to the tool provider and may not be adequate for implementations coming from other suppliers. General testing approaches usually rely on formally defined extensions of Mealy machine, see, e.g., [26], [36].

Most of the existing work on test generation from EFSM concentrates on the model coverage, see, e.g., [3], [14], [18], [28], which provides no characterization of faults that can be detected by the generated tests. There are some techniques for test generation from EFSM which use certain fault models [36], [26] and limited state/configuration identification sequences [5], [19], [26]. The work of [16] uses checking experiment methods, but requires first to determine input/output equivalence classes from a given specification EFSM and choose concrete inputs. To the best of our knowledge, there is no result in lifting the theory of checking experiments to extended FSMs. This observation is one of the main motivations of this work.

Another motivation comes from research on symbolic automata and transducers, which is driven by several practical problems. The work of [34] mentions applications ranging from modern regex analysis to advanced web security analysis where the so-called sanitizers, string transformation routines are extensively used as the first line of defense against cross site scripting attacks. A large class of sanitizers can be described and analyzed by using symbolic finite state transducers. Symbolic finite automata are introduced as an extension of classical finite state automata that allows transitions to be labeled with predicates. Automata with predicates instead of concrete symbols are also used in [37] and discussed in [23] in the context of natural language processing. The work on learning symbolic automata [20] has also to be mentioned here, since the automata learning shares certain aspects with the testing problem in the following

sense. If a black box passes a checking experiment, then under well-defined conditions it is recognized as some automaton. Hence it is important to investigate checking experiments for symbolic automata.

The community focusing on testing from IOTS has also considered extensions to symbolic representation of transition systems which avoid enumerations of its components, see, e.g. [8], [29], but these approaches are not fault model driven, they use one or another test purpose. More references on symbolic approaches in testing could be found, e.g., in [1] and [17].

In this paper, we attempt to lift the theory of checking experiments to a special type of EFSM, which extends the deterministic Mealy machine with predicates defined over input variables, considered as its symbolic inputs. We propose a test generation method that produces a symbolic checking experiment, adapting the well-known HSI method [39]. We then investigate under which conditions instances of a symbolic checking experiment can be used for testing black-box implementations, guaranteeing the full fault coverage.

The paper is organized as follows. In Section 2, we define the model of FSM with symbolic inputs. In Section 3, we study the relations between SIFSMs. Symbolic and concrete checking experiments are introduced in Section 4, where we also investigate fault detection capability of concrete tests obtained from symbolic checking experiments. Section 5 summarizes our contributions and presents future work.

2 Definitions and Notations

2.1 Preliminaries

We define an (input) alphabet as a set of guards over variables of well-defined types. Let G denote the universe of guards that are predicates over variables in a fixed set V for which a decision theory, e.g., an SMT solver, exists, excluding the predicates that are always false. G^* will denote the universe of input sequences.

Let D_V denote the set of all the valuations $\boldsymbol{v}$ of the input variables in the set V, called *concrete* inputs. A set of concrete inputs is called a *symbolic* input; both, concrete and symbolic, inputs are represented by guards in G. Henceforth, we use set-theoretical operations on symbolic inputs. In particular, we write $\boldsymbol{v} \in g$, when concrete input $\boldsymbol{v}$ satisfies g. We define some relations between input sequences in G^*.

Definition 1. Given two input sequences $\alpha, \beta \in G^*$ of the same length k, $\alpha = g_1...g_k$, $\beta = g'_1...g'_k$, we let $\alpha \cap \beta = g_1 \cap g'_1 \ldots g_k \cap g'_k$ denote the sequence of intersections of inputs in sequences α and β; α and β are *compatible*, if for all $i = 1, \ldots, k$, $g_i \cap g'_i \neq \varnothing$. We say that α is a *reduction* of β, denoted $\alpha \subseteq \beta$, if $\alpha = \alpha \cap \beta$. If α is a sequence of concrete inputs as well as a reduction of β then it is called an *instance* of β; given a finite set of input sequences $E \subseteq G^*$, a set of concrete input sequences I is called an *instance* of the set E, if I contains at least one instance for each input sequence in E.

2.2 Symbolic Input FSM

We define a model, called a symbolic input finite state machine (SIFSM), which operates in discrete time as a synchronous machine reading values of input variables and setting up the values of output variables. Output variables are assumed to have a finite number of valuations and form a finite output alphabet. On the other hand, there may exist an infinite set of input valuations. SIFSM uses guards on transitions which are executed one at a time.

Definition 2. A *symbolic input* finite state machine $\mathcal{S}$ (or machine, for short) is a 7-tuple $(S, s_0, V, O, F, \delta, \lambda)$, where

- S is a finite set of states with the initial state s_0,
- V is a finite set of input variables over which guards in G are defined,
- O is a finite set of outputs,
- $F \subseteq S \times G$ is a finite specification domain,
- $\delta : F \to S$ is a transition function, and
- $\lambda : F \to O$ is an output function.

Examples of SIFSM are given in Figure 1. Examples of realistic systems which can be specified as SIFSM could be found in [15] and in [28]. In the first work, the Ceiling speed monitoring following the public ETCS system specification [7] is modelled, it has two input and two output variables. In the second work, an HVAC controller specified in Simulink/Stateflow is considered, it has nine input variables, Boolean and naturals, the most complex transition guard comprises 13 terms.

The semantics of SIFSM is defined by a Mealy state machine with a possibly infinite input set, where the state and output sets remain finite.

Given $(s, g) \in F$, we say that input g is *defined* in state $s \in S$. Then, $G(s) = \{g \in G \mid (s, g) \in F\}$ contains all inputs defined at s. The machine $\mathcal{S}$ is *deterministic*, if for any $(s, g), (s, g') \in F$, it holds that $g \cap g' = \varnothing$. State s of the machine $\mathcal{S}$ is *input-complete*, if for each input valuation $\boldsymbol{v}$, at least one of its guards evaluates to True, i.e., $\{\boldsymbol{v} \in g \mid g \in G(s)\} = D_V$. The machine $\mathcal{S}$ is *input-complete*, if each state is input-complete. The machine $\mathcal{S}$ is *normalized*, if for all $(s, g), (s, g') \in F$, $\delta(s, g) = \delta(s, g')$ implies that $\lambda(s, g) \neq \lambda(s, g')$; in other words, the machine has at most one transition with a given output for each ordered pair of states. Any machine that is not normalized can always be converted into a normalized one by merging transitions with the same start and end states as well as the same output and forming the disjunction of their guards. This is a unique compact form of a SIFSM. We will consider only normalized deterministic input-complete specification machines.

An input sequence $\alpha \in G^*$, $\alpha = g_1 \ldots g_k$, is *defined* in state $s \in S$, if each input in α is defined in a corresponding state, i.e., if there exist states $s_1, \ldots, s_k, s_{k+1}$, where $s_1 = s$, such that $(s_i, g_i) \in F$ and $\delta(s_i, g_i) = s_{i+1}$ for each $1 \leq i \leq k$. Let $\Psi_s(s)$ denote the set of input sequences defined in state s, and Ψ_s denote sequences defined in the initial state of $\mathcal{S}$. Moreover, $\Omega_s(s)$ denotes the set $\Psi_s(s)$ closed under the reduction relation, called the set of input sequences *admissible* in state s, i.e., $\Omega_s(s) = \{ \alpha \in G^* \mid \beta \in \Psi_s(s), \alpha \subseteq \beta\}$, and Ω_s denotes sequences admissible in the initial state of $\mathcal{S}$. Notice that for an

input-complete machine $\mathcal{S}$ any concrete input sequence is admissible in every state, i.e., $D_V^* = \Omega_s(s)$, for each $s \in S$. We lift the transition and output functions from inputs to admissible input sequences, including the empty sequence ε, as usual: for $s \in S$, $\delta(s, \varepsilon) = s$ and $\lambda(s, \varepsilon) = \varepsilon$; and for input sequence $\alpha \in \Omega_s(s)$ and input $g \in \Omega_s(\delta(s, \alpha))$, $\delta(s, \alpha g) = \delta(\delta(s, \alpha), g')$ and $\lambda(s, \alpha g) = \lambda(s, \alpha)\lambda(\delta(s, \alpha), g')$, if $g' \in G_s((\delta(s, \alpha))$ and $g \subseteq g'$.

Considering the input alphabet G, we further extend the transition and output functions to the set of all possible input sequences in G^*. The extended transition function describes the set of all possible states which a deterministic machine from a given state can reach in response to input sequence and the extended output reaction function gives the set of all possible corresponding output sequences; these sets are singletons if the input sequence is admissible for the starting state. We define the function $\Delta : S \times G^* \to 2^S$ as follows. Given $s \in S$ and $\alpha \in G^*$, we let $\Delta(s, \alpha)$ be $\{\delta(s, \beta) \mid \beta \subseteq \alpha, \beta \in \Omega_s(s)\}$. Obviously, for any $\alpha \in \Omega_s(s)$, $\Delta(s, \alpha) = \{\delta(s, \alpha)\}$. Similarly, we define the function $\Lambda : S \times G^* \to 2^{O^*}$. For $s \in S$ and $\alpha \in G^*$, we define $\Lambda(s, \alpha) = \{ \lambda(s, \beta) \mid \beta \subseteq \alpha, \beta \in \Omega_s(s)\}$. We call the functions Δ and Λ the *extended* transition and output functions. For any $\alpha \in \Omega_s(s)$, $\Lambda(s, \alpha) = \{\lambda(s, \alpha)\}$. Given a set of symbolic input sequences $\Phi \subseteq G^*$, the SIFSM $\mathcal{S}$ is said to be a Φ-*converter* if for each $\alpha \in \Phi$, $|\Lambda(s_0, \alpha)| = 1$.

Given input sequence α, we use *pref*(α) to denote the set of all prefixes of α. Similar, *pref*(A) denotes the set of prefixes of sequences in A. The set A is prefix-closed if *pref*(A) = A.

3 Relations Between SIFSMs

In this section, we extend the classical equivalence and distinguishability relations to SIFSMs and introduce new types of distinguishability which have no counterparts in the classical deterministic Mealy machine. We define a designated symbolic machine which can be used to check the distinguishability of symbolic input finite state machines.

In this paper, we focus our attention on deterministic systems, in which different output sequences produced by two states in response to the same symbolic input sequence indicate that the two states can be distinguished by the input sequence.

Definition 3. Given states $s, s' \in S$ of $\mathcal{S}$, states $s, s' \in S$ are *distinguishable*, denoted $s \not\simeq s'$, if there exist compatible input sequences $\alpha \in \Psi_s(s)$ and $\beta \in \Psi_s(s')$, such that $\lambda(s, (\alpha \cap \beta)) \neq \lambda(s', (\alpha \cap \beta))$; the sequence $\alpha \cap \beta$ is called a *separating* sequence for distinguishable states s and s'.

Since the machine is deterministic, its reacts to a given admissible input sequence as it does to any of its reduction. We have therefore the following corollary.

Corollary 1. Any instance of a separating sequence is a separating sequence.

The importance of this property of separating sequences becomes evident in the context of testing, as we discussed later.

Given a prefix-closed set of input sequences $E \subseteq G^*$, we let $s \not\simeq_E s'$ to denote the fact that the set E contains a separating sequence. If E contains no separating sequence then states s and s' are said to be *E-equivalent*, denoted $s \simeq_E s'$, and if $E = G^*$, then the states are *equivalent*, denoted $s \simeq s'$. The machine is *reduced* if it has no equivalent states. We further assume that the specification machine $\mathcal{S}$ is reduced.

As usual, we define equivalence and distinguishability of machines as the corresponding relation between their initial states.

To decide distinguishability we define a designated machine, where, instead of composing transitions caused by the same input as in the case of FSMs [27], we compose transitions with compatible inputs. The machine has the common behavior of the given machines, as the classical automata product (even lifted to symbolic automata [34]), but in addition it signals when they disagree on output and enters a sink state.

Definition 4. Given two SIFSMs $\mathcal{S} = (S, s_0, V, O, F_s, \delta_s, \lambda_s)$ and $\mathcal{P} = (P, p_0, V, O, F_p, \delta_p, \lambda_p)$ over the same set of input variables V, a SIFSM $\mathcal{C} = (C \cup \{\nabla\}, c_0, V, O \cup \{\perp\}, F_c, \delta_c, \lambda_c)$, where ∇ is a designated sink state, $\perp$ is a designated output, is the *distinguishing machine* for $\mathcal{S}$ and $\mathcal{P}$ denoted $\mathcal{S} \oplus \mathcal{P}$, if

- $c_0 = (s_0, p_0)$
- $F_c \subseteq C \times G$ such that for $(s, p) \in C$, $g \cap g' \in G_c(s, p)$, if $g \in G_s(s)$, $g' \in G_p(p)$, and $g \cap g' \neq \varnothing$
- For $(s, p) \in S \times P$ and $g \cap g' \in G_c(s, p)$, $\delta_c((s, p), g \cap g') = (\delta_s(s, g), \delta_p(p, g'))$ and $\lambda_c((s, p), g \cap g') = \lambda_s(\delta(s, g))$, $(\delta_s(s, g), \delta_p(p, g')) \in S \times P$ if $\lambda_s(s, g) = \lambda_p(p, g')$ otherwise, i.e., if $\lambda_s(s, g) \neq \lambda_p(p, g')$, then $\delta_c((s, p), g \cap g') = \nabla$, and $\lambda_c((s, p), g \cap g') = \perp$.

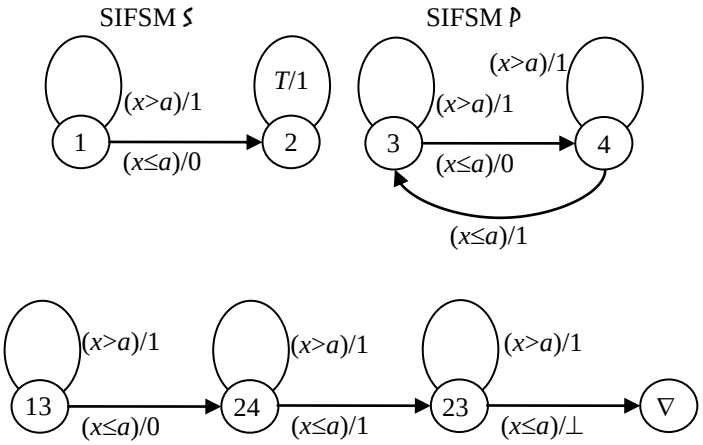

Fig. 1. SIFSMs $\mathcal{S}$, $\mathcal{P}$, and the distinguishing machine $\mathcal{S} \oplus \mathcal{P}$.

We further assume that the distinguishing machine is normalized by merging, if needed, transitions with the designated output $\perp$ from the same state. By the definition, any input sequence reaching the sink state of the distinguishing machine is a separating sequence for the given machines; the distinguishing machine could be used

to decide the equivalence of two distinct machines as well as states in the same machine. To illustrate the above we consider the SIFSMs in Figure 1, where x is an input variable, a is a constant, 0, 1 and 2 are outputs and T stands for True.

The machines $\mathcal{S}$ and $\mathcal{P}$ are distinguishable, as the distinguishing machine $\mathcal{S} \oplus \mathcal{P}$ shows. The shortest separating sequence is $(x \le a)(x \le a)(x \le a)$. Indeed, in response to it $\mathcal{S}$ produces 011, while $\mathcal{P}$ produces 010.

In this example, we have that the separating sequence is admissible in both machines, as required; however, it is defined only in one of them, namely, $(x \le a)(x \le a)(x \le a) \in \Psi_{\mathcal{P}}$, though $(x \le a)(x \le a)(x \le a) \notin \Psi_{\mathcal{S}}$. This sequence is a reduction of the symbolic sequence $(x \le a)(T)(T)$ defined in the machine $\mathcal{S}$. Considering the relations between separating and defined sequences we further refine the distinguishability relation.

Definition 5. Given distinguishable states s, $s' \in S$ of $\mathcal{S}$, s' is *strongly*-distinguishable from s if there exists a separating sequence defined in state s, i.e., $\alpha \in \Psi_{\mathcal{S}}(s)$; if state s' is not strongly-distinguishable from state s, then s' is said to be *weakly*-distinguishable from state s.

For two arbitrary states, they could be equivalent, one can be strongly-distinguishable from another, both can be strongly-distinguishable from each other, or both can be weakly-distinguishable from each other. In the last case, they are just distinguishable. Notice that the strongly-distinguishability relationship is not symmetric.

In our example, the machine $\mathcal{S}$ is strongly-distinguishable from $\mathcal{P}$, because the separating sequence $(x \le a)(x \le a)(x \le a)$ is defined in $\mathcal{P}$. The machine $\mathcal{P}$, in turn, is weakly-distinguishable from $\mathcal{S}$, since all the separating sequences in the distinguishing machine are not defined in $\mathcal{S}$.

As follows from Corollary 1, if the machines are distinguishable, they are distinguished by any instance of a separating sequence; this is also the case when one machine is strongly-distinguishable from another machine and by the definition the separating sequence is defined in the latter. However, an arbitrary instance of such a sequence may not distinguish a machine that is weakly-distinguishable from another. This difference becomes crucial in conformance testing, when one machine represents a specification and another an implementation under test (IUT). To test the latter, only concrete input sequences would be used, when the IUT is treated as a black box. In the example, assuming that the machine $\mathcal{P}$ is the specification and $\mathcal{S}$ is the IUT, any instance of the separating sequence $(x \le a)(x \le a)(x \le a)$ can be used to detect non-conformance of the IUT $\mathcal{S}$, as it is not equivalent to $\mathcal{P}$, moreover, $\mathcal{S}$ is strongly-distinguishable from $\mathcal{P}$. On the other hand, if the machines swap their roles then since $\mathcal{P}$ is weakly-distinguishable from $\mathcal{S}$, then non-conformance of the IUT $\mathcal{P}$ cannot be detected by an arbitrary instance of the separating sequence $(x \le a)(x \le a)(x \le a)$.

We formulate a condition under which a SIFSM is either equivalent to or strongly-distinguishable from another SIFSM. It is based on the property of one machine being a converter for all symbolic input sequences defined in another machine. Intuitively, the condition $|\Lambda(m_0, \alpha)| = 1$ corresponds to the case when the two machines are

equivalent as well as to the case when they produce different output sequences in response to α.

Theorem 1. Given a (specification) machine $\mathcal{S}$ and an (implementation) machine $\mathcal{M}$, if $\mathcal{M}$ is a Ψ_s-converter then $\mathcal{M}$ is either equivalent to or strongly-distinguishable from $\mathcal{S}$.

Proof. Assume that $\mathcal{M}$ is a Ψ_s-converter, i.e., $|\Lambda(m_0, \alpha)| = 1$ for each $\alpha \in \Psi_s$. As $\mathcal{S}$ is deterministic, we have that $|\Lambda(s_0, \alpha)| = 1$ for each $\alpha \in \Psi_s$. Assume also that $\mathcal{M}$ is not strongly-distinguishable from $\mathcal{S}$. Thus, for each $\alpha \in \Psi_s$, $\Lambda(m_0, \alpha) \supseteq \Lambda(s_0, \alpha)$. It implies that for each $\alpha \in \Psi_s$, $\Lambda(m_0, \alpha) = \Lambda(s_0, \alpha)$, since $|\Lambda(m_0, \alpha)| = |\Lambda(s_0, \alpha)| = 1$. Hence, there is no separating sequence for $\mathcal{S}$ and $\mathcal{M}$, i.e., they are equivalent. ♦

Clearly, the sufficient condition is not a necessary one. Consider the machine $\mathcal{P}$ in Figure 1 as an IUT and assume that both transitions from state 4 have the output 0. The modified machine is strongly-distinguishable from $\mathcal{S}$, but it has $|\Lambda(m_0,(x \le a)(T))| = 2$.

4 Symbolic and Concrete Checking Experiments

In this section, we define symbolic checking experiments following a usual framework for defining complete test suite for a given machine, conformance relation, and fault domain [25]. In this case, we are dealing with specification and implementation SIFSMs in a fault domain containing only normalized deterministic input-complete machines; the conformance relation is the machine equivalence. Complete test suite is considered as checking experiment, which could be symbolic or concrete. In the context of symbolic execution and constraint solving, symbolic experiments are of interest for white box testing, while concrete ones for back box testing, where all test data should be concrete. Another specific feature of testing from SIFSM is that a non-equivalent implementation machine in a fault domain can either be weakly- or strongly-distinguishable, which as we show later has a significant impact on fault detection capability of concrete checking experiments.

Let $\mathfrak{I}(V, m)$ be the universe of SIFSMs over the input variables V with at most m states. A subset of $\mathfrak{I}(V, m)$ is called a *fault domain* for a specification machine $\mathcal{S} = (S, s_0, V, O, F, \delta, \lambda)$; it includes SIFSMs which model all possible implementations of $\mathcal{S}$. A set of input sequences $E \subseteq \Omega_s$ is a *checking* experiment for $\mathcal{S}$ in a fault domain $\Sigma \subseteq \mathfrak{I}(V, m)$ iff $\mathcal{S} \simeq_E \mathcal{M}$ implies $\mathcal{S} \simeq \mathcal{M}$, for each $\mathcal{M} \in \Sigma$.

We now define main ingredients of symbolic checking experiments, following the classical approach of state identification.

A *symbolic state cover* C for the machine $\mathcal{S}$ is a set which contains the empty sequence and for each state $s \in S$ a single defined input sequence $\alpha \in \Psi_s$, such that $\delta(s_0, \alpha) = s$. A *symbolic transition cover* T for the machine $\mathcal{S}$ is a set $\{\alpha g \mid \alpha \in C, g \in G(\delta(s_0, \alpha))\}$, where C is a symbolic state cover.

Given state $s \in S$ of the reduced machine $\mathcal{S}$, a finite set $E \subseteq \Omega_s(s)$ is a *state identifier for* s, denoted $Id(s)$, if $s \not\simeq_E s'$ for each $s' \neq s$. State identifiers in a set $H = \{Id(s) \mid s$

$\in S\}$ are *harmonized* if for each pair of distinguishable states s and s', there exists a separating sequence $\alpha \in pref(Id(s)) \cap pref(Id(s'))$. A straightforward way of constructing HSIs is to determine a distinguishing machine for each pair of states and include the found sequence in the identifiers of the states in the pair.

Given a symbolic state cover C, a symbolic transition cover T and a set of harmonized state identifiers $H = \{Id(s) \mid s \in S\}$, a symbolic *HSI experiment* is $\{\alpha\gamma \mid \alpha \in (C \cup T), \gamma \in Id(\delta(s_0, \alpha))\}$.

As an example, we construct a symbolic checking experiment for $\mathcal{S}$ in Figure 1. The state cover is $\{\varepsilon, (x \le a)\}$, the transition cover is $\{(x > a), (x \le a), (x \le a)(T)\}$. The symbolic input $(x \le a)$ separates states, so $Id(1) = Id(2) = (x \le a)$. Then the HSI experiment becomes $\{(x > a)(x \le a), (x \le a)(x \le a), (x \le a)(T)(x \le a)\}$, which could be simplified to $\{(x > a)(x \le a), (x \le a)(T)(x \le a)\}$.

Recall that we assume that a specification SIFSM $\mathcal{S} = (S, s_0, V, O, F, \delta, \lambda)$ is reduced, normalized, deterministic, and input-complete.

Theorem 2. Let $\mathcal{S}$ be a specification SIFSM, an HSI experiment is a checking experiment for $\mathcal{S}$ in the fault domain $\mathfrak{I}(V, n)$.

Before proving Theorem 2, we demonstrate some auxiliary results.

Lemma 3. Let $\mathcal{S} = (S, s_0, V, O, F, \delta_S, \lambda_S)$ be a specification SIFSM, E be an HSI experiment for $\mathcal{S}$ and $\mathcal{M} = (M, m_0, V, O, F_M, \delta_M, \lambda_M)$ be a SIFSM from the fault domain $\mathfrak{I}(V, n)$. If $\mathcal{M} \simeq_E \mathcal{S}$ then $\mathcal{M}$ has n states.

Proof. Let s and s' be two states of $\mathcal{S}$. There exist $\alpha, \alpha' \in C$, such that $\delta_S(s_0, \alpha) = s$ and $\delta_S(s_0, \alpha') = s'$. There also exists $\gamma \in pref(Id(s)) \cap pref(Id(s'))$, such that $\alpha\gamma, \alpha'\gamma \in E$ and $\Lambda(s, \gamma) \ne \Lambda(s', \gamma)$; thus, $\Lambda(\Delta(s_0, \alpha), \gamma) \ne \Lambda(\Delta(s_0, \alpha'), \gamma)$. As $\mathcal{S} \simeq_E \mathcal{M}$, we have that $\Lambda(s_0, \alpha\gamma) = \Lambda(m_0, \alpha\gamma)$ and $\Lambda(s_0, \alpha'\gamma) = \Lambda(m_0, \alpha'\gamma)$. Hence, $\Lambda(\Delta(s_0, \alpha), \gamma) = \Lambda(\Delta(m_0, \alpha), \gamma)$ and $\Lambda(\Delta(s_0, \alpha'), \gamma) = \Lambda(\Delta(m_0, \alpha'), \gamma)$. Thus, $\Lambda(\Delta(m_0, \alpha), \gamma) \ne \Lambda(\Delta(m_0, \alpha'), \gamma)$ and, therefore, $\Delta(m_0, \alpha) \ne \Delta(m_0, \alpha')$. We conclude that for each pair of states of $\mathcal{S}$, there exists at least a pair of states of $\mathcal{M}$ which are distinct. Therefore, $\mathcal{M}$ has at least n states. As $\mathcal{M} \in \mathfrak{I}(V, n)$, $\mathcal{M}$ has at most n states. Thus, $\mathcal{M}$ has n states. ♦

Lemma 4. Let $\mathcal{S} = (S, s_0, V, O, F_S, \delta_S, \lambda_S)$ be a specification SIFSM, E be an HSI experiment for $\mathcal{S}$ and $\mathcal{M} = (M, m_0, V, O, F_M, \delta_M, \lambda_M)$ be a SIFSM from the fault domain $\mathfrak{I}(V, n)$. If $\mathcal{M} \simeq_E \mathcal{S}$ then there exists a bijection $f : S \leftrightarrow M$, such that for each $\alpha \in (C \cup T)$, $f(\Delta(s_0, \alpha)) = \Delta(m_0, \alpha)$.

Proof. C contains n symbolic input sequences, one for each state of $\mathcal{S}$. By Lemma 3, $\mathcal{M}$ has n states. Thus, we can define a bijection $f : S \leftrightarrow M$, such that for each $\alpha \in C$, $f(\Delta(s_0, \alpha)) = \Delta(m_0, \alpha)$. It thus remains to show that for each $\beta \in T$, we also have that $f(\Delta(s_0, \beta)) = \Delta(m_0, \beta)$. Let $\beta \in T$ and $s = \Delta(s_0, \beta)$. There exists $\alpha \in C$, such that $s = \Delta(s_0, \alpha)$. We have that $f(s) = \Delta(m_0, \alpha)$. Let $\alpha' \in C$, such that $s' = \Delta(s_0, \alpha') \ne s$. Thus, $f(s') = \Delta(m_0, \alpha')$. There also exists $\gamma \in pref(Id(s)) \cap pref(Id(s'))$, such that $\beta\gamma, \alpha'\gamma \in E$ and $\Lambda(s, \gamma) \ne \Lambda(s', \gamma)$; thus, $\Lambda(\Delta(s_0, \beta), \gamma) \ne \Lambda(\Delta(s_0, \alpha'), \gamma)$. As $\mathcal{S} \simeq_E \mathcal{M}$, we have that $\Lambda(s_0, \beta\gamma) = \Lambda(m_0, \beta\gamma)$ and $\Lambda(s_0, \alpha'\gamma) = \Lambda(m_0, \alpha'\gamma)$. Hence, $\Lambda(\Delta(s_0, \beta), \gamma) = \Lambda(\Delta(m_0, \beta), \gamma)$ and $\Lambda(\Delta(s_0, \alpha'), \gamma) = \Lambda(\Delta(m_0, \alpha'), \gamma)$. Thus, $\Lambda(\Delta(m_0, \beta), \gamma) \ne \Lambda(\Delta(m_0, \alpha'), \gamma)$ and,

therefore, $\Delta(m_0, \beta) \neq \Delta(m_0, \alpha') = f(s')$ and $\Delta(m_0, \beta) \neq f(s')$. It follows that $f(s) = \Delta(m_0, \beta)$ and, thus, $f(\Delta(s_0, \beta)) = \Delta(m_0, \beta)$. ♦

Corollary 2. If $M \simeq_E S$ then for any $\alpha, \beta \in \Psi_S$, if $\Delta(s_0, \alpha) = \Delta(s_0, \beta)$ then $\Delta(m_0, \alpha) = \Delta(m_0, \beta)$.

Proof. Let $M \simeq_E S$ and $\alpha, \beta \in \Psi_S$, $\Delta(s_0, \alpha) = \Delta(s_0, \beta)$. First, we prove by induction on the prefixes of α that there exists a sequence $\varphi \in C$, such that $\Delta(s_0, \alpha) = \Delta(s_0, \varphi)$ and $\Delta(m_0, \alpha) = \Delta(m_0, \varphi)$.

For the base case, we have $\alpha = \varepsilon$. As $\varepsilon \in C$, the property holds for $\varphi = \alpha = \varepsilon$, since obviously $\Delta(s_0, \alpha) = \Delta(s_0, \varphi)$ and $\Delta(m_0, \alpha) = \Delta(m_0, \varphi)$.

For the inductive case, assume that $\alpha = \alpha' g$ and there exists $\varphi' \in C$, with $\Delta(s_0, \alpha') = \Delta(s_0, \varphi')$ and $\Delta(m_0, \alpha') = \Delta(m_0, \varphi')$. We have that $\varphi' g \in T$. As C is a state cover for S, there exists $\varphi \in C$, such that $\Delta(s_0, \varphi' g) = \Delta(s_0, \varphi)$; hence $\Delta(s_0, \alpha) = \Delta(s_0, \varphi)$. Thus, due to the properties for the bijection f, it follows that $f(\Delta(s_0, \varphi' g)) = \Delta(m_0, \varphi' g)$ and $f(\Delta(s_0, \varphi)) = \Delta(m_0, \varphi)$. It then follows that $\Delta(m_0, \alpha) = \Delta(m_0, \varphi' g) = f(\Delta(s_0, \varphi' g)) = f(\Delta(s_0, \varphi)) = \Delta(m_0, \varphi)$, hence $\Delta(m_0, \alpha) = \Delta(m_0, \varphi)$, concluding the induction proof.

In the same vein, we can prove that there exists a sequence $\varphi' \in C$, such that $\Delta(s_0, \beta) = \Delta(s_0, \varphi')$ and $\Delta(m_0, \beta) = \Delta(m_0, \varphi')$. As $\Delta(s_0, \alpha) = \Delta(s_0, \beta)$ and C contains only one sequence that reaches each state, we have that $\varphi = \varphi'$. Thus, $\Delta(m_0, \beta) = \Delta(m_0, \varphi) = \Delta(m_0, \varphi') = \Delta(m_0, \alpha)$, i.e., $\Delta(m_0, \beta) = \Delta(m_0, \alpha)$. ♦

Now we are ready to prove Theorem 2.

Proof of Theorem 2. Let $M = (M, m_0, V, O, F_M, \delta_M, \lambda_M)$ be a SIFSM from the fault domain $\mathfrak{I}(V, n)$, such that $M \simeq_E S$. We prove by contradiction that $M \simeq S$. Assume that $M \not\simeq S$. Let β be the shortest symbolic input sequence such that $M \simeq_{\{\beta\}} S$ and there exists $g \in G^*$ such that βg is a separating sequence, i.e., $M \not\simeq_{\{\beta g\}} S$. Thus, $\Lambda(\Delta(m_0, \beta), g) \neq \Lambda(\Delta(s_0, \beta), g)$.

Since E is an HSI experiment $\{\alpha\gamma \mid \alpha \in (C \cup T), \gamma \in Id(\delta_S(s_0, \alpha))\}$, it contains a sequence $\varphi \in C$, such that $\Delta(s_0, \varphi) = \Delta(s_0, \beta)$. Then $\Delta(m_0, \varphi) = \Delta(m_0, \beta)$, according to Corollary 2. Since $\Lambda(\Delta(m_0, \beta), g) \neq \Lambda(\Delta(s_0, \beta), g)$, it also holds that $\Lambda(\Delta(m_0, \varphi), g) \neq \Lambda(\Delta(s_0, \varphi), g)$. The HSI experiment contains a transition cover of S then there exists a symbolic input g', such that $(\Delta(s_0, \varphi), g') \in F_S$, $g \subseteq g'$, and $\varphi g' \in E$. $M \simeq_E S$ implies that $M \simeq_{\{\varphi g'\}} S$. We have that $\Delta(m_0, \varphi) = \Delta(m_0, \beta)$, then $\Lambda(\Delta(m_0, \beta), g') = \Lambda(\Delta(s_0, \beta), g')$. This contradicts the assumption that $\Lambda(\Delta(m_0, \beta), g) \neq \Lambda(\Delta(s_0, \beta), g)$, as $g \subseteq g'$.♦

For simplicity, we have considered symbolic experiments for the fault domain $\mathfrak{I}(V, n)$. Nevertheless, based on the previous results, e.g., [30], [33], the case of a wider fault domain $\mathfrak{I}(V, m)$, where $m > n$ can also be considered.

Symbolic experiments could be used in the context of white-box testing when symbolic execution of code/model of an implementation SIFSM is possible; however, they cannot be executed against an implementation SIFSM considered as a black box. We further assume that only instances of symbolic experiments can be executed against any implementation SIFSM in a given fault domain.

Consider first the case, when an SIFSM S has a finite number of concrete inputs, in other words, it is a compact representation of a Mealy FSM S' over the finite input set

D_V. Then the set of all possible instances of a symbolic checking experiment is finite and is in fact a concrete checking experiment for the SIFSM $\mathcal{S}$. The latter is also a classical checking experiment for the FSM $\mathcal{S}'$.

Theorem 3. Given a specification machine $\mathcal{S}$ with a finite input set D_V, let E be a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$. Let also E' be the set of all possible instances of E and $\mathcal{S}'$ be the FSM obtained by unfolding $\mathcal{S}$. Then, E' is a concrete checking experiment for $\mathcal{S}$ and $\mathcal{S}'$ in $\mathfrak{I}(V, n)$.

Proof. As E is a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$, for each $\mathcal{M}$ in $\mathfrak{I}(V, n)$, E contains a separating sequence α distinguishing $\mathcal{S}$ and $\mathcal{M}$. Thus, there exists an instance of α which distinguishes $\mathcal{S}$ and $\mathcal{M}$ and E' contains this instance; hence E' distinguishes $\mathcal{S}$ and any SIFSM in $\mathfrak{I}(V, n)$ which is distinguished by E. As E is a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$, it follows that E' is a checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$. As $\mathcal{S}'$ is equivalent to $\mathcal{S}$, we have that E' is also a checking experiment for $\mathcal{S}'$ in $\mathfrak{I}(V, n)$. ♦

The theorem suggests that checking experiments for SIFSMs with a finite set of concrete inputs can be constructed directly from a given specification machine without first unfolding it into a classical Mealy machine and using one of the existing methods for checking experiment generation. It might also be computationally simpler to first determine all the ingredients of a checking experiment in the symbolic form and then generate all the concrete instances of symbolic sequences one by one.

Next we consider the case when the input variables do not yield a finite set of concrete inputs and we investigate faults detectable by concrete experiments. Since we can execute only a finite number of concrete input sequences, it is interesting to know in which cases the set of single instances of each sequence in a symbolic checking experiment remains a checking experiment for a given fault domain. In the following, we identify several such cases.

Let E be a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$ and let Σ be a subset of $\mathfrak{I}(V, n)$. We say that E is *safely-instantiable* for Σ if any instance of E is a concrete checking experiment for $\mathcal{S}$ in Σ. We will use $\mathfrak{I}(V, n, \Psi_s)$ to denote the subset of $\mathfrak{I}(V, n)$, which consists of Ψ_s-converters.

Theorem 4. Let E be a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$. Then, E is safely-instantiable for $\mathfrak{I}(V, n, \Psi_s)$.

Proof. Let $\mathcal{M} \in \mathfrak{I}(V, n, \Psi_s)$; thus, for each $\alpha \in \Psi_s$, $|\Lambda(m_0, \alpha)| = 1$. According to Theorem 1, the machine $\mathcal{M}$ is either equivalent to or strongly-distinguishable from $\mathcal{S}$. Let C be an instance of E. Assume that $\mathcal{M}$ is not equivalent to $\mathcal{S}$; thus, $\mathcal{M}$ is strongly-distinguishable from $\mathcal{S}$. As E is a symbolic checking experiment for $\mathcal{S}$ in $\mathfrak{I}(V, n)$ and $\mathfrak{I}(V, n, \Psi_s) \subseteq \mathfrak{I}(V, n)$, there exists a symbolic separating sequence $\alpha \in E$, such that $\mathcal{M} \not\simeq_{\{\alpha\}} \mathcal{S}$; by Corollary 1, any instance of the sequence α is also a separating sequence. Thus, the result follows. ♦

Theorem 4 says that any concrete experiment derived from a symbolic checking experiment is also a checking experiment for the machine $\mathcal{S}$ in the fault domain $\mathfrak{I}(V, n, \Psi_s)$. In other words, a complete concrete test suite can be obtained from a symbolic

checking experiment. The question arises as to which structural faults in the implementation machines preserve their property of being Ψ_S-converters. Addressing this question, we follow the same approach for describing faults as in the classical deterministic Mealy machines, see, e.g., [2], [24]. Implementation faults are usually modeled by mutants of a given machine. Elements of transitions, namely, output and end state, are subjects for mutations, which yield output faults, transfer faults and transition faults combining the first two types of faults.

It is not difficult to see that all possible mutants of the specification SIFSM $\mathcal{S}$ with output faults are Ψ_S-converters, i.e., they are in the fault domain $\mathfrak{I}(V, n, \Psi_S)$.

As to transition faults, they should not violate the property of Ψ_S-converters, namely, a mutant with transition faults should react to any symbolic input sequence defined the specification with a single output sequence. It turns out that mutants with transition faults remain to be Ψ_S-converters under the following conditions.

Assume that the specification SIFSM $\mathcal{S}$ has a fixed set of guards in each and every state, i.e., $G(s) = G(s') = G_S$ for all states $s, s' \in S$. Let G_M denote the set of guards of an implementation SIFSM $\mathcal{M}$ with the same property.

Theorem 5. Given a specification SIFSM $\mathcal{S}$ with the set of guards G_S then $\{\mathcal{M} \in \mathfrak{I}(V, n) \mid G_M = G_S\} \subseteq \mathfrak{I}(V, n, \Psi_S)$.

Proof. Let $\mathcal{M} \in \mathfrak{I}(V, n)$, such that $G_M = G_S$. Indeed, $G_M = G_S$ implies $\Psi_M = \Psi_S$. As $\mathfrak{I}(V, n)$ includes only deterministic machines, we have that $|\Lambda(m_0, \alpha)| = 1$ for each $\alpha \in \Psi_M$; therefore $|\Lambda(m_0, \alpha)| = 1$ for each $\alpha \in \Psi_S$. Thus, $\mathcal{M} \in \mathfrak{I}(V, n, \Psi_S)$. ♦

We identify another sufficient condition considering the case when the set of defined input sequences of each implementation machine in a fault domain is a superset of that of the specification machine. Intuitively, an implementation machine is assumed to preserve in each state guards of the specification or merge some of them.

Theorem 6. Given a specification SIFSM $\mathcal{S}$, it holds that $\{\mathcal{M} \in \mathfrak{I}(V, n) \mid \forall \alpha \in \Psi_S, \forall g \in G_S(\delta(s_0, \alpha)), \exists g' \in G_M(\delta(m_0, \alpha)), g \subseteq g'\} \subseteq \mathfrak{I}(V, n, \Psi_S)$.

Proof. Let $\mathcal{M} \in \mathfrak{I}(V, n)$, such that for each $\alpha \in \Psi_S$ and each $g \in G_S(\delta(s_0, \alpha))$, there exists $g' \in G_M(\delta(m_0, \alpha))$ such that $g \subseteq g'$. We prove by induction that for each $\alpha \in \Psi_S$, $|\Lambda(m_0, \alpha)| = 1$.

For the basis step, we have that $\Lambda(m_0, \varepsilon) = \{\varepsilon\}$, i.e., $|\Lambda(m_0, \varepsilon)| = 1$. For the induction step, assume that $\alpha = \beta g \in \Psi_S$ and $|\Lambda(m_0, \beta)| = 1$. Let also $g \in G_S(\delta(s_0, \beta))$, $g' \in G_M(\delta(m_0, \beta))$, such that $g \subseteq g'$. We can see that $|\Lambda(\delta(m_0, \beta), g')| = 1$; thus $|\Lambda(\delta(m_0, \beta), g)| = 1$. Consequently, $|\Lambda(m_0, \beta g)| = 1$, i.e., $|\Lambda(m_0, \alpha)| = 1$; therefore, $|\Lambda(m_0, \alpha)| = 1$ for each $\alpha \in \Psi_S$. Thus, $\mathcal{M} \in \mathfrak{I}(V, n, \Psi_S)$. ♦

This theorem addresses a specific fault model of symbolic implementation machines representing the mutation by merging transitions along with their guards which is not possible in classical FSM, since an implementation FSM should have all the inputs of a specification FSM. In case of SIFSMs, implementation machines have all the input variables of a specification machine, but not necessarily its guards.

Consider SIFSMs in Figure 1. The machine $\mathcal{S}$ can be considered a mutant of the specification machine $\mathcal{P}$, where transitions with guards $(x \le a)$ and $(x > a)$ are merged into a transition with the guard T. On the other hand, when the machine $\mathcal{S}$ serves as a

specification and the machine P is an implementation this mutant has a specific fault of splitting a guard used in the specification SIFSM. To detect such a fault, one has use to use at least two instances of a symbolic input sequence from the checking experiment. Since P is treated as a black box testing, and the way a guard is split is unknown, two concrete tests suffice if they properly "guess" it.

5 Conclusions

We investigated possibilities for lifting the checking experiments theory developed for the classical (Mealy) finite state machine model to its extension, where input alphabet is finite, but consists of predicates defined over input variables with large or even infinite domains. We call it FSM with symbolic inputs, SIFSM. On one hand, this model can be considered as a special type of Extended FSMs (EFSMs) [26], without context variables and operations on variables; on another hand, as symbolic automaton or symbolic transducer [23], [35]. The recent grow of interest towards symbolic models could be explained by advances in constraint solving technology, as SMT solvers become efficient [22].

We lifted the machine equivalence and distinguishability relations to SIFSM and identified new distinguishability relations which have no counterparts in classical deterministic Mealy machines. Then, we defined symbolic checking experiments for deterministic SIFSMs and demonstrated that they could be obtained by mimicking, e.g., a classical HSI method for constructing checking experiments of FSMs (other types of state identification facilities, such as W and Wp, might also be used). Since symbolic experiments could be used for white-box testing, but not for black-box testing, which requires concrete test values, we focused on investigating fault domains for which any concrete instance of a symbolic checking experiment remains a checking experiment.

As expected, in the most general setting, an arbitrary instance of a symbolic checking experiment may not be a checking experiment in the same fault domain. Nevertheless, we found some sufficient conditions for the specification and implementation machines under which any instance of a symbolic checking experiment is also a checking experiment in well-defined fault domains. Under these conditions, non-trivial faults modeled by the identified fault domains are detectable by concrete tests obtained from abstract (symbolic) tests in a symbolic checking experiment. These faults include transition merging, which is only relevant to implementations of SIFSM and not to classical Mealy machines.

The novelty of the results comes from the fact that while FSM checking experiments are known for about 60 years, EFSMs for about 30 years, there are no published results on checking experiments for EFSM which cannot be unfolded into FSM. To the best of our knowledge, it is the first attempt to advance the checking experiment theory to FSMs with a symbolic extension.

While the problem of handling more general EFSMs remains open, we believe that the presented results open a new line of research in checking experiments for symbolic state machines and transition systems.

Our current work concerns, on one hand, relaxing the sufficient conditions of safe-instantiability, and on the other hand, extending the SIFSM model with operations on output variables, thus lifting the checking experiments theory to a wider class of extended finite state machines. We also plan to investigate other fault models for which symbolic checking experiments could be used as efficiently as for the faults satisfying the formulated sufficient conditions.

Acknowledgements. The first author acknowledges financial support of NSERC via Discovery grant RGPIN/194381-2012. The second author had financial support of Brazilian Funding Agencies, CNPq, Capes and FAPESP.

References

[1] Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P.: An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software, 86(8):1978-2001 (2013)

[2] Bochmann, G.V., Das, A., Dssouli, R., Dubuc, M., Ghedamsi, A., Luo, G.: Fault models in testing. In: Proceedings of the IFIP TC6/WG6. 1 Fourth International Workshop on Protocol Test Systems IV, pp. 17-30. North-Holland Publishing Co. (1991)

[3] Cheng, K.T., Krishnakumar, A.S.: Automatic Functional test generation using the extended finite state machine model. In: Proc. 30th Design Automation Conf., pp. 86-91, (1993)

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

[5] Chun, W., and Amer, P.D.L.: Test case generation for protocols specified in Estelle. In: Proceedings of the IFIP TC6/WG6. 1 Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols: Formal Description Techniques, III, pp. 191-206. North-Holland Publishing Co (1990)

[6] Dorofeeva, R., Yevtushenko, N., El-Fakih, K. and Cavalli, A.: Experimental evaluation of FSM-based testing methods. In: Third IEEE Int. Conf. Software Engineering and Formal Methods, pp. 23–32. IEEE Computer Society (2005)

[7] European Railway Agency: ERTMS—System Requirements Specification—UNISIG SUBSET-026 May 2014. http://www.era.europa.eu/Document-Register/Pages/Set-2-System-Requirements-Specification.aspx

[8] Frantzen, L., Tretmans, J., Willemse, T.A.C.: Test generation based on symbolic specifications, In: Formal Approaches to Software Testing (FATES), LNCS 3395, pp 1-15 (2004)

[9] Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language, Morgan Kaufmann, San Francisco (2014)

[10] Fujiwara, S., von Bochmann, G., Khendek, F., Amalou, M. and Ghedamsi, A.: Test selection based on finite state models. IEEE Transactions of Software Engineering, 17(6):591–603 (1991)

[11] Glässer, U., Gotzhein, R., Prinz, A.: The formal semantics of SDL-2000: status and perspectives. Computer Networks, 42(3):343-358 (2003)

[12] Harel D., Naamad, A.: The STATEMATE Semantics of Statecharts, ACM Trans. Software Eng. and Methodology, 5(4):293-333 (1996)

[13] Hennie, F.C.: Fault-detecting experiments for sequential circuits. Proc. Fifth Annual Symp. On Circuit Theory and Logical Design, pp. 95–110 (1965)
[14] Hong, H.S., Lee, I., Sokolsky, O., Ural, H.: A temporal logic based theory of test coverage and generation. In: Tools and Algorithms for the Construction and Analysis of Systems. LNCS, pp. 327-341. Springer, Berlin (2002)
[15] Huang, W., Peleska, J.: Exhaustive model-based equivalence class testing. In: International Conference on Testing Software and Systems. LNCS, pp. 49–64. Springer, Berlin (2013)
[16] Huang, W., Peleska, J.: Complete model-based equivalence class testing. Int. Journal on Software Tools and Technology Transfer, DOI 10.1007/s10009-014-0356-8 (2014)
[17] Jéron, T., Veanes, M., Wolff, B.: (Ed.) Symbolic methods in testing, Report from Dagstuhl Seminar 13021 (2013)
[18] Kalaji, A.S., Hierons, R.M., Swift, S.: Generating feasible transition paths for testing from an extended finite state machine (efsm). In: International Conference on Software Testing, Verification and Validation, IEEE Computer Society, Silver Spring, pp. 230–239 (2009)
[19] Li, X., Higashino, T., Higuchi, M., Taniguchi, K.: Automatic generation of extended UIO sequences for communication protocols in an EFSM model. In: Proc. Seventh Int'l Workshop Protocol Test Systems, pp. 225-240 (1994)
[20] Maler, O., Mens, I.: Learning regular languages over large alphabets. In: Tools and Algorithms for the Construction and Analysis of Systems 2014, LNCS 8413, pp. 485–499 (2014)
[21] Moore, E.F.: Gedanken-experiments on sequential machines. Automata Studies, Princeton University Press, Princeton, NJ, pp. 129-153 (1956)
[22] Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems pp. 337-340. Springer Berlin Heidelberg (2008)
[23] Noord, G.v., Gerdemann, D.: Finite state transducers with predicates and identities. Grammars 4(3), pp. 263–286 (2001)
[24] Petrenko, A., Yevtushenko, N.: Test suite generation for a given type of implementation errors. In: Proc. IFIP XII International Conference Protocol Specification, Testing, and Verification, pp. 229-243 (1992)
[25] Petrenko, A., Yevtushenko, N., Bochmann, G.v: Fault models for testing in context. In: Formal description techniques IX—theory, application and tools. Chapman & Hall, London, pp. 163–177 (1996).
[26] Petrenko, A., Boroday, S., Groz, R.: Confirming configurations in EFSM testing. IEEE Transactions on Software Engineering, 30(1):29-42 (2004)
[27] Petrenko, A., Yevtushenko, N.: Conformance tests as checking experiments for partial nondeterministic FSM, In: Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software, 2005, LNCS 3997, pp. 118-133 (2005)
[28] Petrenko, A., Dury, A., Ramesh, S., Mohalik, S.: A method and tool for test optimization for automotive controllers. In: Proceedings of the 9th Workshop on Advances in Model Based Testing (A-MOST 2013) of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013), Luxembourg. (2013)
[29] Rusu, V., Du Bousquet, L., Jeron, T.: An approach to symbolic test generation. In: Proc. Second International Conference on Integrating Formal Methods (IFM'00), pp. 338-357 (2000)
[30] Simao, A., Petrenko, A., Yevtushenko, N.: Generating reduced tests for FSMs with extra states. In: Proceedings of TestCom/Fates, LNCS 5826, pp. 129–145 (2009)

[31] Simao, A., Petrenko, A.: Generating complete and finite test suite for ioco: is it possible? In: Proceedings of MBT 2014, Electronic Proceedings in Theoretical Computer Science, 141, pp. 56-70 (2014)
[32] Tiwari, A.: Formal semantics and analysis methods for Simulink Stateflow models. Technical Report, SRI International (2002)
[33] Vasilevskii, M.P.: Failure diagnosis of automata. Cybernetics, Plenum Publishing Corporation, New York, 4:653-665 (1973)
[34] Veanes, M.: Applications of symbolic finite automata. Implementation and Application of Automata. Springer Berlin Heidelberg, pp. 16-23 (2013)
[35] Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., Bjorner, N.: Symbolic finite state transducers: algorithms and applications. In: Proceedings of the 39th ACM Symposium on Principles of programming languages, pp. 137–150 (2012)
[36] Wang, C.J., Liu, M.T.: Generating test cases for EFSM with given fault model. In: Proceedings of Twelfth Conference of the IEEE Computer and Communications Societies, pp. 774-781 (1993)
[37] Watson, B.W.: Implementing and using finite automata toolkits. In: Extended Finite State Models of Language, pp. 19–36. Cambridge University Press, New York (1999)
[38] Yannakakis, M., Lee, D.: Testing finite state machines: fault detection. Journal of Computer and System Sciences, 50(2):209-227 (1995)
[39] Yevtushenko, N., Petrenko, A.: Synthesis of test experiments in some classes of automata. Automatic Control and Computer Sciences, 24(4):50-55 (1990)