2012 IEEE Fifth International Conference on Software Testing, Verification and Validation

Generating Checking Sequences for Nondeterministic Finite State Machines

Alexandre Petrenko
Centre de recherche informatique de Montreal (CRIM)
Montreal, Quebec, Canada
alexandre.petrenko@crim.ca

Adenilso Simao
São Paulo University
São Carlos, São Paulo, Brazil
adenilso@icmc.usp.br

Nina Yevtushenko
Tomsk State University
Tomsk, Russia
ninayevtushenko@yahoo.com

***Abstract* – A checking sequence is a single input sequence which is able to reveal all the faults in a given fault domain. There are many methods for generating checking sequences for deterministic finite state machines (FSM); however, we are not aware of any generalization to nondeterministic machines. Nondeterministic specifications are needed for software testing, as they describe the behavior of a wider class of reactive systems than deterministic FSMs when depending on the environment conditions, a nondeterministic system is allowed to take different runs under the same input sequence. In this paper, we propose a method for constructing checking sequences when both the specification and implementations under test are modeled by nondeterministic FSMs.**

I. INTRODUCTION

Testing using finite state machines (FSMs) is getting more and more embraced by a most of the industry. While the theory of testing with finite state machines has a long history traced back to the 50-ties of the last century [15], most of the theoretical work done concerns deterministic machines. The mainstream methods elaborated since that time address a fundamental problem of generating so-called complete tests, which are exhaustive in a predefined fault domain, i.e., a finite set of all possible implementations under test (IUTs), modeled by deterministic FSMs. The methods can be roughly divided into checking sequence and multiple checking experiment construction methods. A checking sequence is a single input sequence (i.e., it is a simple experiment [15]), while a multiple checking experiment includes multiple input sequences. Examples of the first type of methods are [6] [7] [9], while the methods W [3] [25], Wp [5], HSI [26], H [4] and SPY [22] are examples of the second type of methods. Both, checking sequences and experiments, provide complete tests. To execute a checking sequence it is required first to initialize an IUT into its initial state, using either a reset operation or a homing sequence. If the first option is feasible then the reset operation is executed only once, as opposed to a checking experiment; to execute the latter, one needs to reset the IUT before each input sequence.

While the problem of test generation from a deterministic FSM is still an active research topic (see, e.g., [9] [22]), recently in the 90-ties, nondeterministic machines have come to the attention of researchers [1] [2] [10] [11] [16] [24] [26] [27]. Nondeterministic specifications are needed for software testing, as they describe the behavior of a wider class of reactive systems than deterministic FSMs when, depending on the environment conditions, a nondeterministic system is allowed to take different runs under the same input sequence. At the same time, nondeterministic models are unavoidable once abstractions are employed. Some work assumes that a specification FSM is nondeterministic, but all IUTs are deterministic, see, e.g., [8] [17].

To test a nondeterministic IUT the existence of a reset operation in IUTs becomes a necessity, since a nondeterministic implementation has alternative runs for the same input sequence and to check all of them, the tester need to repeatedly apply the input sequence over and over again. This check is only possible assuming some fairness of a nondeterministic IUT, as in [13] [14] [23]. In the context of test execution against nondeterministic IUTs, the reset operation has to be repeatedly used to execute either a single input sequence or several of them. Hence, for the nondeterministic case, the choice between checking sequences and checking experiments is not based on whether a reliable reset is used or not, as it is in the deterministic case; instead, other constraints, specific to the application domain of the IUT, play an important role. For instance, in some application domains it may be preferable to have short tests even if their number is big, while in others, on the contrary, it may be more interesting to have as few tests as possible, even if they are longer. On the one hand, short tests facilitate the debugging; on the other hand, longer tests tend to have bigger fault coverage, far exceeding the fault domain in which their completeness is provided by construction. Minimizing the number of tests which need to be executed is also justified in situation when the test execution harness needs to be adjusted each time when a new test is provided. For example, different test scripts should be produced for each new test. These adjustments can be costly and time-consuming (especially when manual work is involved); therefore it is of a practical interest to elaborate methods for generating checking sequences from nondeterministic FSMs.

To the best of our knowledge, all the methods developed for complete test generation from nondeterministic FSMs follow the ideas of constructing checking experiments initially elaborated for deterministic FSMs, see, e.g., [13] [18]. We have found no attempts in generalizing the problem of checking sequence construction to the nondeterministic FSMs.

The main contributions of this paper are twofold. First, we generalize the problem of generating a checking sequence to nondeterministic FSMs. We state conditions which are sufficient to ensure that a checking sequence can be generated. Second, we propose a method for generating an input sequence which satisfies those conditions. Thus, we propose a method for generating a checking sequence for a nondeterministic FSM.

The remaining of this paper is organized as follows. In Sections 2, we introduce the basic notations and definitions for FSMs. In Section 3, we discuss how the notion of checking sequence can be generalized to nondeterministic FSMs. In Section 4, we investigate how the properties regarding the convergence and divergence of traces can be determined and, in Section 5, we propose a method for generating checking sequence which is based on these properties. In Section 6, we present an example of the execution of the proposed method and Section 7 concludes the paper.

II. Definitions

A Finite State Machine is a tuple $\mathsf{S} = (S, s_0, X, Y, h_\mathsf{S})$, such that:

- S is the finite set of states, with the initial state s_0;
- X is the nonempty finite set of inputs;
- Y is the nonempty finite set of outputs; and
- $h_\mathsf{S} \subseteq S \times X \times Y \times S$ is the set of transitions

As usual, we denote $(s, x, y, s') \in h_\mathsf{S}$ as $(s, x/y, s')$. S is *observable* if $(s_1, x/y, s_2) \in h_\mathsf{S}$ and $(s_1, x/y, s_3) \in h_\mathsf{S}$ implies that $s_2 = s_3$. In this paper, we further consider only observable machines. Given a state $s \in S$, $\alpha = x_1y_1x_2y_2\ldots x_ky_k$ is a *trace* of s if there exist $s_1, s_2, \ldots, s_{k+1} \in S$, such that $s_1 = s$ and $(s_i, x_i/y_i, s_{i+1}) \in h_\mathsf{S}$, for $1 \le i \le k$; we let $\delta_\mathsf{S}(s, \alpha)$ denote the final state s_{k+1}. Moreover, for each $1 \le i \le k$; we say that α *covers* transition $(s_i, x_i/y_i, s_{i+1})$. The FSM S is *strongly-connected* if for any pair of states (s, s') there exists a trace α such that $\delta_\mathsf{S}(s, \alpha) = s'$. For the initial state s_0, we often write $\delta_\mathsf{S}(\alpha)$ instead of $\delta_\mathsf{S}(s_0, \alpha)$. Figure 1 shows the example of FSM. It has four states 1, 2, 3, and 4 with state 1 as the initial state; three inputs a, b, and c; and two outputs 0 and 1. It is a strongly-connected machine.

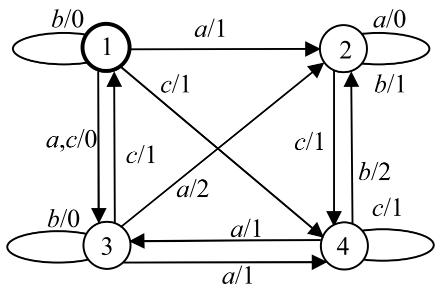


Figure 1. An FSM S

The set of traces of state s is denoted by $tr_\mathsf{S}(s)$. We denote by ε the *empty* trace, such that for any $s \in S$, $\varepsilon \in tr_\mathsf{S}(s)$, and $\delta_\mathsf{S}(s, \varepsilon) = s$. The input projection of α, denoted by $\alpha_{\downarrow X}$, is the input sequence $x_1x_2\ldots x_k$, while the output projection of α, denoted by $\alpha_{\downarrow Y}$, is the output sequence $y_1y_2\ldots y_n$. Given a state $s \in S$ and an input sequence $\beta \in X^*$, we use $traces_\mathsf{S}(s, \beta)$ to denote the traces of s with the input projection β, i.e., $traces_\mathsf{S}(s, \beta) = \{\alpha \in tr_\mathsf{S}(s) \mid \alpha_{\downarrow X} = \beta\}$. For the initial state s_0, we simply write $traces_\mathsf{S}(\beta)$. Given a set $traces_\mathsf{S}(\beta)$ of sequences, the notation $pref(traces_\mathsf{S}(\beta))$ is used to denote the set of all prefixes of sequences of the set. Given two set of traces A and B, we denote by $A.B$ the set of traces obtained by appending traces of B to traces of A, i.e., $A.B = \{\alpha\beta \mid \alpha \in A \text{ and } \beta \in B\}$.

The set of all the traces of state $s \in S$ is $tr_\mathsf{S}(s)$ and that of the FSM S is $tr_\mathsf{S}(s_0)$ or simply tr_S. For any FSM S, there exists an observable FSM S', such that $tr_{\mathsf{S}'} = tr_\mathsf{S}$. Two states (FSMs) are *equivalent* if they have the same set of traces. An FSM is *minimal* if it has no equivalent states. We note that the FSM in Figure 1 is minimal.

We now lift δ_S, tr_S and $traces_\mathsf{S}$ to sets of states and sets of traces. Given a set of states $R \subseteq S$, a set of traces A and a set of input sequences B, we define:

- $\delta_\mathsf{S}(R, A) = \{\delta_\mathsf{S}(s, \alpha) \mid s \in R \text{ and } \alpha \in A\}$;
- $tr_\mathsf{S}(R) = \cup tr_\mathsf{S}(s)$ over $s \in R$
- $traces_\mathsf{S}(R, B) = \cup\ traces_\mathsf{S}(s, \alpha)$ over $s \in R$ and $\alpha \in B$.

For simplicity, we use the only element of a singleton set as the set itself, i.e., a is used instead of $\{a\}$.

S is *deterministic* if $(s_1, x_1/y_1, s_2) \in h_\mathsf{S}$ and $(s_1, x_1/y_2, s_3) \in h_\mathsf{S}$ implies that $y_1 = y_2$ and $s_2 = s_3$. S is *complete* if for each $s \in S$ and $x \in X$, there exists $(s, x/y, s') \in h_\mathsf{S}$ for some $y \in Y$ and $s' \in S$. In this paper, we assume that the FSMs are observable and complete, but possibly nondeterministic. In fact, the FSM in Figure 1 is non-deterministic, but observable.

A set of traces A is a *state cover*, if for each state A contains a trace which leads to this state, i.e., for each $s \in S$ there exists $\alpha \in A$ such that $\delta_\mathsf{S}(\alpha) = s$; A is a *transition cover* (for S), if for each transition, A contains a trace which covers it. The set of traces A is *initialized* if $\varepsilon \in A$.

III. Problem Statement

In fault model-based testing, a fault domain is a finite set of all possible implementations which represent all the faults that a tester is interested in detecting in an implementation under test (IUT); once the fault domain is established, a set of tests can be generated from a given specification to check whether the IUT contains any of those faults. Usually, when the specification is a formal model, the implementation is assumed to be some unknown model of the same kind (see, [13] [23]). This is called a *testing hypothesis*. In our case, as we are dealing with FSM, we assume that the implementation can also be represented as some FSM. We denote by $\Im(\mathsf{S})$ the set of complete, observable FSMs with at most as many states as S and the same set of inputs. This set is then a *fault domain*, since each machine models a particular combination of implementation faults.

Fault detection relies on a conformance relation between specification and implementation models. An implementation $\mathsf{N} \in \Im(\mathsf{S})$ *passes* an input sequence ω if $traces_{\mathsf{S}}(\omega) = traces_{\mathsf{N}}(\omega)$; otherwise, N *fails* the input sequence ω. N is *trace equivalent* to S if $tr_{\mathsf{N}} = tr_{\mathsf{S}}$. We assume that for an FSM $\mathsf{N} \in \Im(\mathsf{S})$ and an input sequence ω, there exists a natural k such that if ω is applied to N k times, each trace in $traces_{\mathsf{N}}(\omega)$ is observed. This assumption is used to reason about nondeterministic implementations. It is called all-weather conditions in [14], fairness in [23] and complete testing assumption in [13]. In practical situations, the higher the value of k the more reliable the test verdict pass. We also assume that N has a "reliable reset", which brings N to its initial state and is needed to be able to observe all the traces with a given input sequence according to the all-weather conditions assumption.

As the implementation is (assumed to be) an FSM from a given fault domain, we can generate tests by considering each and every FSM from the fault domain and, if it is not trace equivalent to the specification, determine an input sequence for which the implementation fails. These input sequences can eventually be merged into a single input sequence following the ideas of Moore [15]. However, even though the fault domain is finite (upon isomorphism between machines), it is usually huge and the approach described above is unfeasible. Our goal is then to determine input sequences which can detect any implementation which is not trace equivalent, without having to enumerate them. In particular, we investigate whether it is possible to generate a *single* input sequence with this capacity. Such a sequence is called a *checking sequence*, which is formally defined as follows.

Definition 1. *An input sequence $\omega \in X^*$ is a* checking sequence *for S (w.r.t. $\Im(\mathsf{S})$) if for each $\mathsf{N} \in \Im(\mathsf{S})$, N passes ω only if S and N are equivalent, i.e., $traces_{\mathsf{N}}(\omega) = traces_{\mathsf{S}}(\omega)$ implies that $tr_{\mathsf{N}} = tr_{\mathsf{S}}$.*

Definition 1 generalizes the notion of checking sequence when both the specification and the implementation FSMs are deterministic [7]. For the deterministic case, there are numerous methods for generating checking sequence [6] [7] [9] [19]. However, to the best of our knowledge, there is no method which considers a more general case when the specification and the implementation can be nondeterministic.

We also generalize the definition of a distinguishing sequence, which is used in many checking sequence generation methods. The input sequence γ is a *distinguishing sequence* for S if for any two different states s and s' of S, $traces_{\mathsf{S}}(s, \gamma) \neq traces_{\mathsf{S}}(s', \gamma)$. It is known that not every complete reduced FSM has a distinguishing sequence. In this paper, we consider the specification FSM which has such a sequence and thus, is minimal. The distinguishing sequence γ of the FSM S allows one to separate states reached after different traces by the sets of traces which emanate from these states as response to γ. By direct inspection one can assure that the FSM S has a distinguishing sequence $\gamma = ab$. Indeed, $traces_{\mathsf{S}}(1, ab) = \{a0b0, a1b1\}$, $traces_{\mathsf{S}}(2, ab) = \{a0b1\}$, $traces_{\mathsf{S}}(3, ab) = \{a2b1, a1b2\}$ and $traces_{\mathsf{S}}(4, ab) = \{a1b0\}$.

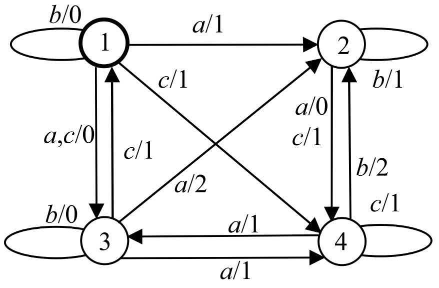
We can now state the problem we are addressing in this paper. Given a strongly-connected, complete, observable (possibly nondeterministic) FSM with a distinguishing sequence, generate a checking sequence. In the next section, we establish properties of traces which will be used to elaborate an algorithm for checking sequence generation. The properties, which are based on convergence and divergence of traces, generalize our previous work [19] [20] to dealing with nondeterministic FSMs.

IV. Trace Convergence and Divergence

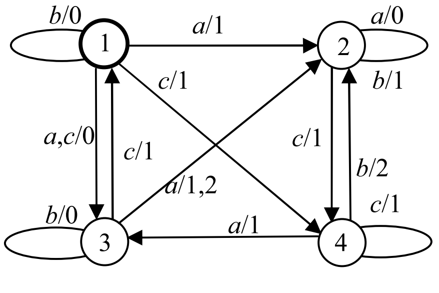
In a nondeterministic machine, a given input sequence may take the machine into several states, but in an observable machine, a trace leads to a unique state. Any two traces are either convergent if they lead to the same state or divergent otherwise. These notions are generalized to a set of NFSMs as follows.

Definition 2. *Given a set of observable FSMs Σ over input alphabet X and output alphabet Y two traces of all FSMs in Σ are* Σ-convergent*, if they converge (i.e., lead from the initial state to the same state) in each FSM of the set Σ; and two traces are* Σ-divergent*, if they diverge (i.e., lead from the initial state to different states) in each FSM of Σ. Two traces are* S-convergent *(*S-divergent*) if they are $\{\mathsf{S}\}$-convergent ($\{\mathsf{S}\}$-divergent). Moreover, when it is clear from the context, the set in which tests are convergent or divergent will be omitted.*

Here we notice that differently from [19] [20], the notions of convergence and divergence are defined in terms of traces of nondeterministic FSMs and not input sequences as in the case of deterministic FSMs. To illustrate the notion of trace divergence and convergence, consider the FSMs S in Figure 1, and M and N in Figure 2. The FSM M was obtained from the FSM S by replacing the transition (2, *a*/0, 2) by the transition (2, *a*/0, 4), while the FSM N was obtained from the FSM S by replacing the transition (3, *a*/1, 4) by the transition (3, *a*/1, 2), The traces *a*1*a*0 and *a*0*a*1 are {S}-divergent, while the traces *a*1*a*0 and *c*1*b*2 are {S}-convergent. The traces *a*1*a*0 and *a*0*a*1 are neither {S, N}-divergent nor {S, M}-divergent. In fact, the traces *a*1*a*0 and *a*0*a*1 are {M, N}-convergent.



(a) FSM M



(b)
(b) FSM N

Figure 2. Two FSMs in $\Im(S)$

Trace convergence and divergence with respect to a single observable FSM are complementary, i.e., any two traces are either convergent or divergent. However, when a fault domain Σ with more than one FSM is considered, some traces may be neither Σ-convergent nor Σ-divergent. Nevertheless, it is possible to relate convergent and divergent traces as follows.

Lemma 1. *Given Σ-convergent traces α and β, the following properties hold:*

a) $\alpha\gamma$ and $\beta\gamma$ are also Σ-convergent, for any trace γ over input alphabet X and output alphabet Y;

b) For any trace φ over input alphabet X and output alphabet Y, if α and φ are Σ-divergent, then β and φ are Σ-divergent as well.

c) For any trace φ over input alphabet X and output alphabet Y, if α and φ are Σ-convergent, then β and φ are Σ-convergent as well.

Proof. Property (a) follows from the fact that the FSMs in Σ are observable. The proof of Properties (b) and (c) is straightforward.♦

Let Σ_ω be the set of FSMs in Σ which pass ω. These are implementations conforming to the specification (recall that the trace equivalence is our conformance relation).

Definition 3. *Given a specification FSM* S*, a set of traces A is*

- Σ_ω-convergent *if any pair of traces of A is Σ_ω-convergent;*
- Σ_ω-convergence-preserving *if any pair of* S*-convergent traces of A are Σ_ω-convergent;*
- Σ_ω-divergent *if any pair of traces of A are Σ_ω-divergent.*

The notion of convergence preserving traces is central in the proposed method; in fact it is straightforward to verify if two traces are convergent in the specification. If two traces are convergent in the specification and we can ensure that they both belong to a Σ_ω-convergence-preserving set, by definition, those two traces are convergent in any FSM in Σ which passes ω. Consider again the FSMs in Figure 1 and 2. Let Σ = {S, M, N} and S be the specification FSM. Let ω be the empty sequence; thus, both M and N pass ω. The set of traces {*b*0, *a*0*a*1, *a*0*c*1} is Σ_ω-convergence-preserving, since the only pair of traces which is S-convergent is *b*0 and *a*0*c*1, which is also convergent in M and N. On the other hand, the set {*a*0*a*1, *c*1} is not Σ_ω-convergence-preserving, since the traces are S-convergent, but are not N-convergent. Notice that only the convergence of the traces which are convergent in the specification is relevant. Moreover, only the FSM which passes ω should be considered. Thus, for the input sequence χ = *aaa*, we have that the set {*a*0*a*1, *c*1} is Σ_χ-convergence-preserving, since N does not pass χ.

Now we are ready to state sufficient conditions for an input sequence to be a checking sequence of a given possibly nondeterministic FSM.

Theorem 1. *Given an FSM* S*, let ω be an input sequence such that pref(traces*$_S$*(ω)) contains an $\Im(S)_\omega$-convergence-preserving initialized transition cover. Then, ω is a checking sequence of* S.

Proof. Let $T \subseteq pref(traces_S(\omega))$ be an $\Im(S)_\omega$-convergence-preserving initialized transition cover and $N \in \Im(S)_\omega$. Define the relation $\xi : S \times N$ as $\xi = \{(\delta_S(\alpha), \delta_N(\alpha)) \mid \alpha \in T\}$. As *T* is a transition cover of S, thus *T* contains a state cover, i.e., for each $s \in S$, there exists $n \in N$, such that $(s, n) \in \xi$. Moreover, as *T* is $\Im(S)_\omega$-convergence-preserving and FSM N is observable, for each $s \in S$, there exists only one $n \in N$, such that $(s, n) \in \xi$; thus, ξ is a mapping. As *T* is initialized,

$$\xi(s_0) = n_0. \qquad (1)$$

Let $(s, x/y)$ be a transition of S. As T is a transition cover for S, there exists α, $\alpha xy\beta \in T$, such that $\delta_S(\alpha) = s$ and β could be an empty trace. Thus,

$$\xi(\delta_S(\delta_S(\alpha), xy)) = \xi(\delta_S(\alpha xy)) = \delta_N(\alpha xy) = \delta_N(\delta_N(\alpha), xy) = \delta_N(\xi(\delta_S(\alpha)), xy) \quad (2)$$

As N passes ω, then $traces_S(\omega) = traces_N(\omega)$. Thus, for $\alpha \in T$, $tr_S(\delta_S(\alpha)) = tr_N(\delta_N(\alpha))$ and, consequently, for each $x \in X$,

$$tr_S(\delta_S(\alpha), x) = tr_N(\delta_N(\alpha), x) = traces_N(\xi(\delta_S(\alpha)), x). \quad (3)$$

Therefore, from (2) and (3), we conclude that ξ is an isomorphism and, from (1), that N is equivalent to S. ♦

Thus, to construct a checking sequence, it is sufficient to generate an input sequence that produces a set of traces covering transitions of the specification FSM, which contains the empty word and is convergence-preserving in all the machines that correctly react to this sequence. In the following we first present a chain of statements which show how this could be done. The idea is to use a distinguishing sequence to ensure the divergence of prefixes of the traces caused by the input sequence being constructed. Then, based on the fact that the maximal number of states in the implementation is assumed to be known and not exceeding that of the specification FSM, the convergence of the prefixes which do not diverge is guaranteed first to obtain a state cover and then a transition cover with the desired properties.

The S-divergence and eventually Σ_ω-divergence of a pair of traces can be ensured when the FSM S reaches two distinct states after them. Then their Σ_ω-divergence can be demonstrated by using the distinguishing sequence in those states. To this end, we introduce the notion of separable traces.

Definition 4. *Given an input sequence ω, two traces $\alpha, \beta \in pref(traces_S(\omega))$ are* ω-separable *if there exists an input sequence γ, such that $(\alpha_{\downarrow X})\gamma$, $(\beta_{\downarrow X})\gamma \in pref(\omega)$ and $traces_S(\delta_S(s_0, \alpha), \gamma) \neq traces_S(\delta_S(s_0, \beta), \gamma)$.*

In fact, a distinguishing sequence of a given FSM S will be used while constructing an input sequence ω with ω-separable traces. According to Definitions 2 and 4, two ω-separable traces are divergent in all FSMs which pass ω, as stated in the next lemma.

Lemma 2. *Given an input sequence ω, two ω-separable traces are $\Im(S)_\omega$-divergent.*

Proof. Let traces α and β be ω-separable. Thus, there exists an input sequence γ such that $(\alpha_{\downarrow X})\gamma$, $(\beta_{\downarrow X})\gamma \in pref(\omega)$ and $traces_S(\delta_S(\alpha), \gamma) \neq traces_S(\delta_S(\beta), \gamma)$. Let $N \in \Im(S)_\omega$; then $traces_S(\omega) = traces_N(\omega)$. It follows that $traces_S(\delta_S(\alpha), \gamma) = traces_N(\delta_N(\alpha), \gamma)$ and $traces_S(\delta_S(\beta), \gamma) = traces_N(\delta_N(\beta), \gamma)$. We have that $traces_N(\delta_N(\alpha), \gamma) \neq traces_S(\delta_N(\beta), \gamma)$. Thus, $\delta_N(\alpha) \neq \delta_N(\beta)$. ♦

While the divergence of traces can be directly certified by the different responses (i.e., different sets of traces) exhibited for the same input sequence, the convergence of two traces cannot be straightforwardly verified, since they can lead to different states which happen to produce the same set of traces. However, as the number of states of any FSM in the fault domain $\Im(S)_\omega$ does not exceed that of the specification FSM, a set of divergent traces can be used to ensure that two traces are convergent.

Lemma 3. *Given an input sequence ω, let T be a $\Im(S)_\omega$-divergent set with n traces, where n is the number of states of* S. *Let α and β be* S*-convergent traces, such that $\beta \in T$ and the set $T \setminus \{\beta\} \cup \{\alpha\}$ is $\Im(S)_\omega$-divergent. Then, α and β are $\Im(S)_\omega$-convergent.*

Proof. Let N be an FSM in $\Im(S)_\omega$. As T is $\Im(S)_\omega$-divergent, the traces in T reach n distinct states in N, as T is a state cover of S. As $T \setminus \{\beta\} \cup \{\alpha\}$ is $\Im(S)_\omega$-divergent, α is $\Im(S)_\omega$-divergent with each trace in $T \setminus \{\beta\}$. As $T \setminus \{\beta\}$ is $\Im(S)_\omega$-divergent, the traces in $T \setminus \{\beta\}$ reach $(n - 1)$ distinct states in N. If α and β are N-divergent, then the traces in $T \setminus \{\beta\} \cup \{\alpha\}$ would reach $(n - 1 + 2) = (n + 1)$ distinct states in N. As N has at most n states, we conclude that α and β are N-convergent and, therefore, $\Im(S)_\omega$-convergent. ♦

V. Generating Checking Sequences

In this section, a method for generating checking sequences from a nondeterministic FSM with a distinguishing sequence is proposed. First, we apply the results of the previous section to obtain a suitable convergence-preserving set.

The next lemma states that the set of traces which are followed by a distinguishing sequence form a convergence-preserving set, i.e., for any two of those traces, the convergence in the specification is certainly preserved in any FSM of the fault domain which passes the input sequence ω.

Lemma 4. *Let ω be an input sequence and γ be a distinguishing sequence. Let also $K \subseteq pref(\omega)$ be the set of prefixes of ω which are followed by γ in ω, i.e., $K = \{\alpha \mid \alpha\gamma \in pref(\omega)\}$. If $\delta_S(traces_S(K))$ is a state cover of* S *then $traces_S(K)$ are $\Im(S)_\omega$-convergence-preserving.*

Proof. We prove that if the traces $\alpha, \beta \in traces_S(K)$ are S-convergent, they are also $\Im(S)_\omega$-convergent. Assume that α and β are S-convergent. Consider a subset $T \subseteq traces_S(K)$ such that $\beta \in T$ and for each state $s \in S$, there exists one and only one trace $\chi_s \in T$, such that $s = \delta_S(\chi_s)$. Thus, for each trace $\chi_s \in T$, $\chi_s.traces_S(s, \gamma) \subseteq pref(traces_S(\omega))$. Let $N \in \Im(S)_\omega$; $\chi_s, \chi_{s'} \in T$, $\chi_s \neq \chi_{s'}$, $n = \delta_N(\chi_s)$, $n' = \delta_N(\chi_{s'})$. As N passes ω, it holds that $traces_S(\omega) = traces_N(\omega)$. Hence, $traces_S(s, \gamma) = traces_N(n, \gamma)$ and $traces_S(s', \gamma) = traces_N(n', \gamma)$. As γ is a distinguishing sequence and $s \neq s'$, we have that $traces_S(s, \gamma) \neq traces_S(s', \gamma)$ and, consequently, $traces_N(n, \gamma) \neq traces_N(n', \gamma)$. Thus, $n \neq n'$, i.e., χ_s and $\chi_{s'}$ are N-divergent. It follows that χ_s and $\chi_{s'}$ are $\Im(S)_\omega$-divergent and, therefore, T is $\Im(S)_\omega$-divergent.

We now show that $T \setminus \{\beta\} \cup \{\alpha\}$ is also $\Im(S)_\omega$-divergent. As $T \setminus \{\beta\}$ is $\Im(S)_\omega$-divergent, it is sufficient to show that α is $\Im(S)_\omega$-divergent with each trace in $T \setminus \{\beta\}$. Let $\chi \in T \setminus \{\beta\}$, $s = \delta_S(\alpha)$, $s' = \delta_S(\chi)$, $n = \delta_N(\alpha)$, $n' = \delta_N(\chi)$. As β and χ are S-divergent while α and β are S-convergent, we have that α and χ are S-divergent, i.e., $s \neq s'$. Following the same reasoning as stated above, we conclude that $n \neq n'$, i.e., α and χ are N-divergent. Thus, α and χ are $\Im(S)_\omega$-divergent and, consequently, $T \setminus \{\beta\} \cup \{\alpha\}$ is also $\Im(S)_\omega$-divergent. Finally, by Lemma 3, α and β are $\Im(S)_\omega$-convergent. ♦

Lemma 4 shows that a set of traces, which is a state cover (for the FSM S), is convergence-preserving if each of its traces is followed by a distinguishing sequence.

Constructing an input sequence ω such that the set of traces $pref(traces_S(\omega))$ contains an $\Im(S)_\omega$-convergence-preserving initialized transition cover, we proceed in two steps. First, we generate an input sequence σ such that the set of traces $pref(traces_S(\sigma))$ contains an $\Im(S)_\sigma$-convergence-preserving initialized state cover. Then, we extend the input sequence until the $\Im(S)_\sigma$-convergence-preserving set covers all transitions. Recall that to cover a transition $(s, x/y, s')$, this set of traces should contain a trace α leading to the state s, as well as its extension $\alpha xy\beta$.

Constructing a transition cover with the desired properties, it is convenient to use the following definition.

Definition 5. *Given an input sequence σ, a trace χ of a state s, we say that the trace χ is* verified *at the state s if there exists an $\Im(S)_\sigma$-convergence-preserving set C, such that α, $\alpha\chi \in C$ and $\delta_S(\alpha) = s$. A transition (s, x/y, s') is* verified *if xy is verified at s.*

Notice that if a transition is verified, then it is covered by an $\Im(S)_\sigma$-convergence-preserving set. Thus, it is clear that if all transitions are verified by an $\Im(S)_\sigma$-convergence-preserving initialized state cover, then it is also an $\Im(S)_\sigma$-convergence-preserving initialized transition cover. That is, according to Theorem 1, a checking sequence can be obtained by verifying each transition. Consider the transition $(s, x/y, s')$ and the input sequence σ. Let $\alpha \in traces_S(\sigma)$ be such that $s \in \delta_S(\alpha)$. Lemma 4 suggests a way for including αxy into an $\Im(S)_{\sigma x}$-convergence-preserving set; it is sufficient to extend the input sequence σx by the distinguishing sequence γ. According to Definition 5, to verify the transition $(s, x/y, s')$, it is also required that α itself is in the $\Im(S)_{\sigma x}$-convergence-preserving set. However, Lemma 4 cannot be used, since if we extend σ by the distinguishing sequence γ, we will not be able to extend σx with γ (except for the special case where $\gamma = xx\ldots x$, i.e., the distinguishing sequence is a sequence of the same input symbol). Hence, in order to be able to verify the transition $(s, x/y, s')$, we need first to ensure that all the traces of σ is an $\Im(S)_\sigma$-convergence-preserving set. This is achieved in the following way.

Assume that an input sequence σ is followed by two distinguishing sequences in a row, i.e., it is followed by $\gamma\gamma$. According to Lemma 4, all traces of σ and $\sigma\gamma$ are in an $\Im(S)_{\sigma\gamma\gamma}$-convergence-preserving set and, thus, each trace $\chi \in traces_S(s, \gamma)$ is verified at each state $s \in \delta_S(traces_S(\sigma))$. Suppose now that for each state $s \in S$, there exists an input sequence σ, such that σ is followed by $\gamma\gamma$ and $s \in \delta_S(traces_S(\sigma))$, i.e., each trace χ in $traces_S(s, \gamma)$ is verified at s. Then for any input sequence ϕ that is followed by a distinguishing sequence γ, all the traces of ϕ are in a $\Im(S)_{\phi\gamma}$-convergence-preserving set, due to Lemma 4, as well as all the traces of $\phi\gamma$, due to Lemma 1.a and Definition 5. In other words, once we have guaranteed that each state is reached and followed by two distinguishing sequences $\gamma\gamma$, all the traces of any input sequence which ends with γ is also in a convergence-preserving set. Therefore, in order to verify all of the transitions, we need first to verify all traces of the distinguishing sequence γ at each state. This is accomplished by Algorithm 1.

Algorithm 1
Input: A strongly-connected, complete, observable FSM $S = (S, s_0, X, Y, h_S)$ with a distinguishing sequence γ.
Output: A sequence ω such that for each state s and each trace $\chi \in \delta_S(traces_S(s, \gamma))$, χ is verified at s and the set of traces $pref(traces_S(\omega))$ contains an $\Im(S)_\omega$-convergence-preserving initialized state cover.

```
1.    $R := \emptyset$
2.    $\omega := \varepsilon$
3.    while $R \neq S$ do
4.        Let $\alpha$ be a shortest input sequence, such that $\delta_S(traces_S(\omega\alpha)) \setminus R \neq \emptyset$.
5.        $\omega := \omega\alpha$
6.        $R := R \cup \delta_S(traces_S(\omega))$
7.        while $\delta_S(traces_S(\omega\gamma)) \setminus R \neq \emptyset$ do
8.            $\omega := \omega\gamma$
9.            $R := R \cup \delta_S(traces_S(\omega))$
10.       end
11.       $\omega := \omega\gamma\gamma$
12.   end
13.   return $\omega$
```

The next lemmas state the key properties of the obtained input sequence ω. Lemma 4 characterizes which prefixes of ω are followed by the distinguishing sequence; it also states that ω begins with the distinguishing sequence and ends with two distinguishing sequences in a row.

Lemma 5. *Let ω be the input sequence obtained by Algorithm 1 for a distinguishing sequence γ. Then,*

a) for each $s \in S$, there exists $\beta \in pref(\omega)$, such that $s \in \delta_S(traces_S(\beta))$ and $\beta\gamma\gamma \in pref(\omega)$;

b) $\gamma \in pref(\omega)$; and

c) there exists α, such that $\alpha\gamma\gamma = \omega$

Proof. First, notice that the algorithm terminates when R contains all states of S. Thus, whenever the outer loop (Steps 3-12) is executed, there exists at least one state which is not currently in R. The input sequence α is selected such that at least one state which is not in R is reached. As S is strongly-connected, such an input sequence always exists. Then the inner loop (Steps 7-10) will be executed, as long as extending ω with γ reaches states which have not been reached yet. Thus, eventually, all states will be added to R. Thus, for each state $s \in S$, there exists an input sequence β that is a prefix of ω such that $s \in \delta_S(traces_S(\beta))$. When the inner loop is completed, in Step 11, β is extended with $\gamma\gamma$. Therefore, Property (a) holds.

Property (b) also holds, since in the beginning of the execution, ω is the empty sequence and R is empty. Thus, the input sequence α selected in Step 4 should also be empty sequence, since $\delta_S(traces_S(\omega\alpha)) = \delta_S(traces_S(\varepsilon)) = \{s_0\}$, and, ε is a shortest such a sequence.

As the last step executed by the algorithm is the extension of ω with $\gamma\gamma$, Property (c) also holds. ♦

The next result states that if an input sequence α is followed by $\gamma\gamma$, then α, $\alpha\gamma$ and $\alpha\gamma\gamma$ are in a convergence-preserving set.

Corollary 6. *Let ω be the input sequence obtained by Algorithm 1 for a distinguishing sequence γ. Let also $K = \{\alpha, \alpha\gamma, \alpha\gamma\gamma \mid \alpha\gamma\gamma \in pref(\omega)\}$ Then, the following properties hold:*

a) $\delta_S(traces_S(K))$ is a state cover of S;

b) K is initialized;

c) $\omega \in K$;

d) the set $traces_S(K)$ is $\Im(\mathsf{S})_\omega$-convergence-preserving.

Proof. Properties a), b) and c) follow from the respective items of Lemma 5, whereas Property d) follows from Lemma 4 and Lemma 1.a. ♦

Thus, the set $C = traces_S(K)$ is an $\Im(\mathsf{S})_\omega$-convergence-preserving initialized set of traces. We then extend ω, if needed and, accordingly, the $\Im(\mathsf{S})_\omega$-convergence-preserving set C, so that C becomes a transition cover; thus, by Theorem 1, ω is a checking sequence.

Therefore, we extend ω, until all transitions are verified. Let P be the set of all states which may be reached in the specification after the application of input sequence ω, i.e., $P = \delta_S(traces_S(\omega))$, where ω is the input sequence obtained by Algorithm 1. By Corollary 6.c, it follows that

$$traces_S(\omega) \subseteq C. \qquad (4)$$

Thus, applying an input x after ω would result in traversing all transitions starting in any state of P with input x. Notice that, due to the all-weather conditions assumption, x is also applied sufficiently many times, and thus, each transition $(s, x/y, s')$, such that $s \in P$ and output $y \in Y$, is covered. We then apply the distinguishing sequence γ, to check the end states of the transitions, obtaining the input sequence $\omega x\gamma$. By Lemma 4,

$$C \cup traces_S(\omega x) \text{ is } \Im(\mathsf{S})_\omega\text{-convergence-preserving.} \quad (5)$$

From (4) and (5), it follows that each transition $(s, x/y, s')$ is verified. From (5) and Lemma 1.b, we have that $C \cup traces_S(\omega x) \cup traces_S(\omega x\gamma)$ is also $\Im(\mathsf{S})_{\omega x\gamma}$-convergence-preserving. Thus, we can update ω to $\omega x\gamma$, the set P to $\delta_S(traces_S(\omega x\gamma))$ and the set C to $C \cup traces_S(\omega x) \cup traces_S(\omega x\gamma)$, and we iterate again to verify transitions not yet verified.

It may happen that there are no unverified transitions starting in states in P. In this case, there is a path of verified transitions which leads to states with unverified transitions. Let Z be the set of states with unverified transitions. As the specification is strongly-connected, it is always possible to reach states in Z from states in P using a trace with only verified transitions; let β be a shortest such trace and $\alpha = \beta_{\downarrow X}$. Due to the all-weather conditions assumption, the trace β will eventually be executed. As β is a shortest trace from states in P to states in Z formed only by verified transitions, all traces in $traces_S(\alpha)$ are also produced only by verified transitions. By Lemma 1.b, we have that $C \cup traces_S(\omega\alpha)$ is $\Im(\mathsf{S})_\omega$-convergence-preserving. Thus, we update ω to $\omega\alpha$, the set P to $\delta_S(traces_S(\omega\alpha))$ and the set C to $C \cup traces_S(\omega\alpha)$. Then, we iterate again to verify transitions not yet verified. The idea is formalized in the following algorithm.

The algorithm for verifying the transitions is shown below.

Algorithm 2

Input: A strongly-connected, complete, observable FSM $\mathsf{S} = (S, s_0, X, Y, h_S)$ with a distinguishing sequence γ and the input sequence ω obtained from Algorithm 1

Output: A checking sequence ω.

```
/* P is the set of states currently reached in the specification by ω */
$P := \delta_S(traces_S(\omega))$
/* V is the set of all pairs $(s, x) \in S \times X$ such that all transitions from the state s under the input x are verified. Initially, no transition is verified. * /
$V := \varnothing$
/* While there are unverified transitions do */
While $V \neq S \times X$
        /* R is the set of pairs (s, x) s.t. there exists an unverified transition starting at s with input x */
        Let $R = (S \times X) \setminus V$
        /* Z is the set of states with unverified outgoing transitions */
        Let $Z = \{s \mid \exists\, x \in X$, s.t. $(s, x) \in R\}$
        /* Check if there is any state which is currently reached and has unverified transitions */
        If $Z \cap P \neq \varnothing$ then
                /* Let x be input such that there is an unverified transition with input x from some state in $Z \cap P$*/
                Let x be s.t. $\{(s, x) \in R \mid s \in Z \cap P\} \neq \varnothing$.
                $\omega := \omega x\gamma$
                /* W is the set of all pairs 'state, input' where a transition from the state under the input is just verified */
                $W := (P \times \{x\}) \cap R$
                $V := V \cup W$
                /* Update the currently reached states */
                $P := \delta_S(P, traces_S(P, x\gamma))$
        Else /* there is no unverified transition in a current state */
                /* Then, we apply an input sequence which leads S from some state in the set P to a state with unverified transitions */
                Let α be a shortest input sequence, s.t. $\delta_S(P, traces_S(P, \alpha)) \cap Z \neq \varnothing$.
                $\omega := \omega\alpha$
                /* We update P and in the next iteration, the if condition will be satisfied. */
                $P := \delta_S(P, traces_S(P, \alpha))$
        End
End
return ω.
```

Theorem 2. *The above algorithms return a checking sequence ω.*

Proof. In fact, a sequence returned by Algorithm 1 establishes one-to-one correspondence between states of FSM S and an IUT $N \in \Im(S)$ and yields a convergence-preserving initialized state cover. The set of traces of a sequence extended by Algorithm 2 covers all transitions. Thus, the traces of the resulting ω contains an $\Im(S)_\omega$-convergence-preserving initialized transition cover, and, by Theorem 1, ω is a checking sequence for S.♦

Here we briefly discuss the complexity issues. Similar to deterministic FSMs, the length of a distinguishing sequence can be exponential [12], however, in most cases, when such a sequence exists its length is less than the number of states of the specification FSM. The complexity of algorithms 1 and 2 is polynomial w.r.t. the number of states (transitions) of the specification FSM. Correspondingly, the length of a checking sequence is polynomial w.r.t. the number of transitions of the specification FSM multiplied by the length of a distinguishing sequence. The main difference with deterministic FSMs is that when checking non-deterministic implementations a checking sequence should be applied several times in order to satisfy the complete testing assumption.

VI. Example

Consider the specification FSM *S* in Figure 1, which has the distinguishing sequence γ = *ab*. We now construct a checking sequence using Algorithms 1 and 2.

Initially, we apply Algorithm 1. The set *R* is empty and ω is the empty sequence ε. As $R \neq S$, the while loop is executed. As $\delta_S(traces_S(\omega))$ contains the state s_0, which is not in *R*, we have that α = ε is selected in Step 4. Thus, ω = ωα = ε, and $R = \{s_0\} = \{1\}$. Then, γ is applied and ω is updated to γ = *ab*. The states 2 and 3 are added to *R*, since $\delta_S(traces_S(ab)) = \{2, 3\}$; thus *R* is updated to {1, 2, 3}. Another application of γ does not reach any new states, since $\delta_S(traces_S(abab)) = \{2, 3\}$. To ensure that all traces of γ is verified at states 1, 2 and 3, the distinguishing sequence is applied twice, resulting in ω = *ababab*. Then, we determine an input sequence which reaches a state which is not in *R*; in this case, we have that $\delta_S(traces_S(ababab)) = \{3\}$ and only 4 is not in *R* yet. Thus, we select the input sequence *c* and obtain the input sequence ω = *abababc*. The set *R* is updated and now contains all states; finally, Algorithm 1 applies the distinguishing sequence twice and returns the sequence ω = *abababcabab*.

We now verify each transition of the specification FSM *S*. The set *P* of states reached by ω is {2} and no transition is verified yet. Thus, select the transition (2, *a*/0, 2) to be verified. We apply *a*γ = *aab*, obtaining ω = *abababcababaab*. The states reached by ω are still {2}. We verify the transition (2, *b*/1, 2), obtaining ω = *abababcababaabbab*. Again, the set of reached states remains {2}, thus, we verify the last unverified transition (2, *c*/1, 4). The resulting ω is *abababcababaabbabcab*, which reaches the state 3. We now verify the transitions (3, *a*/1, 4) and (3, *a*/2, 2), by applying *aab*. Thus, ω becomes *abababcababaabbabcabaab*; the set of reached states is {2, 3}. We verify the transition (3, *b*/0, 3) by applying *bab*. Notice that the transition (2, *b*/1, 2) was (re-)verified as well. The obtained ω is *abababcababaabbabcabaabbab*, which reaches the state 2. As there is no unverified transition at the state 2, a shortest input sequence leading to a state with unverified transitions is determined; in this case, the input *c* is chosen, since it leads to state 4. The obtained ω is *abababcababaabbabcabaabbabc*. Then, the transition (4, *a*/1, 3) is verified, by applying *aab*, resulting in ω = *abababcababaabbabcabaabbabcaab*. Up to this point, the following transitions have been verified: (2, *a*/0, 2), (2, *b*/1, 2), (2, *c*/1, 4), (3, *a*/1, 4), (3, *a*/2, 2), (4, *a*/1, 3), and ω reaches the state 2. The remaining steps of the execution of Algorithm 2 are presented in Table I. The resulting checking sequence ω is *abababcababaabbabcabaabbabcaabcbabccabcababcacabc aabcacbabccab*. The total number of inputs in ω is 62.

TABLE I. GENERATING A CHECKING SEQUENCE FOR THE FSM OF FIGURE 1.

Current States *P*	Verified Transitions	Applied Sequences (transfer sequence α, input *x*, and γ)
{2}	(4, *b*/2, 2)	(*c*, *b*, *ab*)
{2}	(4, *c*/1, 4)	(*c*, *c*, *ab*)
{2}	(3, *b*/0, 3)	(*ca*, *b*, *ab*)
{2}	(3, *c*/1, 1)	(*ca*, *c*, *ab*)
{2, 3}	(1, *a*/1, 2), (1, *a*/0, 3)	(*c*, *a*, *ab*)
{2}	(1, *b*/0, 1)	(*cac*, *b*, *ab*)
{2, 3}	(1, *c*/0, 3)	(*c*, *c*, *ab*)

For the sake of comparison, a checking experiment (complete test suite) was generated by the method from [18]. It contains 12 different input sequences, with 57 inputs. Even though the checking sequence has slightly more inputs than the checking experiment, but the latter contains multiple input sequences and, therefore, may require more efforts to execute, since in this case, each input sequence should be executed several times.

VII. CONCLUSIONS

In this paper, we addressed the problem of generating a checking sequence for nondeterministic FSMs, generalizing results from the deterministic case. We stated conditions under which a checking sequence can be generated and used them for proposing a method for generating an input sequence which satisfies these conditions. To the best of our knowledge, this is the first method developed for checking sequence generation from nondeterministic FSMs.

The presented results are also applicable to a class of Input/Output Transition systems (IOTS), called Mealy-IOTS in [21], which have no input enabled in an unstable state.

As future work, it would also be interesting to investigate possibilities of constructing checking sequences when conforming implementations can be "less nondeterministic" than their specifications.

ACKNOWLEDGMENT

The authors acknowledge financial supports of le ministère du Développement économique, de l'innovation et de l'exportation (MDEIE), Brazilian Funding Agency CNPq (Grants 305236/2009-1 and 474152/2010-3), and Russian Found of basic research (Grant 10-08-92003-HHC_a). The authors wish to thank the anonymous referees for their constructive and helpful comments.

REFERENCES

[1] AboElFotoh, H. Abou-Rabia O., and Ural H.: A Test Generation Algorithm for Protocols Modeled as Non-Deterministic FSMs. The Software Engineering Journal, 1993, 8(4):184-188.

[2] Alur R., Courcoubetis C., and Yannakakis M.: Distinguishing Tests for Nondeterministic and Probabilistic Machines. 27th ACM Symposium on Theory of Comp., 1995:363-372.

[3] Chow T. S.: Testing Software Design Modeled By Finite-State Machines. IEEE Transactions on Software Engineering 1978; **4**(3):178-187.

[4] Dorofeeva R., El-Fakih K., Yevtushenko N.: An Improved Conformance Testing Method. Proceedings of the Formal Techniques for Networked and Distributed Systems. (Lecture Notes in Computer Science, vol. 3731), Springer, Heidelberg, 2005:204-218.

[5] Fujiwara S., Bochmann Gv, Khendek F., Amalou M., Ghedamsi A.: Test Selection Based on Finite State Models. IEEE Transactions on Software Engineering 1991; **17**(6):591-603.

[6] Gonenc, G.: A Method for The Design of Fault Detection Experiments. IEEE Transactions on Computers 1970, **19**(6):551-558

[7] Hennie F. C.: Fault-Detecting Experiments For Sequential Circuits. Proceedings of the 5th Annual Symposium on Circuit Theory and Logical Design, Princeton, New Jersey, 1965:95-110.

[8] Hierons R. M.: Adaptive Testing of a Deterministic Implementation against a Nondeterministic Finite State Machine. The Computer Journal, 1998, **41**(5):349-355.

[9] Hierons, R. M., Ural, H.: Optimizing the Length of Checking Sequences. IEEE Transactions on Computers 2006; **55**(5):618-629.

[10] Hwang I., Kim T., Hong S., Lee J.: Test Selection for a Nondeterministic FSM, Computer Communications, 2001, 24/12, 7:1213-1223.

[11] Kloosterman H.: Test Derivation from Non-Deterministic Finite State Machines. Protocol Test Systems, V, Proceedings of the IFIP TC6/WG6.1 Fifth International Workshop on Protocol Test Systems, Canada, 1992. North-Holland 1993:297-308.

[12] Kohavi Z., Switching and Finite Automata Theory, 1970, McGraw-Hill.

[13] Luo G., Bochmann Gv., Petrenko A.: Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized WP-Method. IEEE Transactions on Software Engineering 1994; **20**(2):149-162.

[14] Milner R.: A Calculus of Communicating Systems, 1980, Springer Verlag.

[15] Moore E. F.: Gedanken-Experiments on Sequential Machines. Automata Studies, 1956; 34:129-153.

[16] Petrenko A., Yevtushenko N., Lebedev A., and Das A.: Nondeterministic State Machines in Protocol Conformance Testing, Proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, France, 1993:363-378.

[17] Petrenko A., Yevtushenko N., and Bochmann Gv.: Testing Deterministic Implementations from their Nondeterministic Specifications, Proceedings of the IFIP Ninth International Workshop on Testing of Communicating Systems, 1996:125-140.

[18] Petrenko A., Yevtushenko N.: Conformance Tests as Checking Experiments for Partial Nondeterministic FSM, In Proceedings of the 5th Int. Workshop on Formal Approaches to Testing of Software (FATES 2005). (Lecture Notes in Computer Science, vol. 3997), 2006:118-133.

[19] Simao A., Petrenko A.: Generating Checking Sequences for Partial Reduced Finite State Machines. 20th IFIP Int. Conference on Testing of Communicating Systems and 8th Int. Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2008). (Lecture Notes in Computer Science, vol. 5047), 2008:153-168.

[20] Simao A., Petrenko A.: Fault Coverage-Driven Incremental Test Generation. Computer Journal, 2010; **53**(9):1508-1522.

[21] Simao A., Petrenko A.: Generating Asynchronous Test Cases from Test Purposes, Information and Software Technology, 2011, **53**(11):1252-1262.

[22] Simao A., Petrenko A., Yevtushenko N.: Generating Reduced Tests for FSMs with Extra States. 20th IFIP Int. Conference on Testing of Communicating Systems and 8th Int. Workshop on Formal Approaches to Testing of Software (TestCom/FATES 2008). (Lecture Notes in Computer Science, vol. 5826). 2009:129-145.

[23] Tretmans J.: Test Generation with Inputs, Outputs and Repetitive Quiescence. Software - Concepts and Tools 1996; **17**(3):103-120.

[24] Tripathy P. and Naik K.: Generation of Adaptive Test Cases from Nondeterministic Finite State Models. Protocol Test Systems, V, Proceedings of the IFIP TC6/WG6.1 Fifth International Workshop on Protocol Test Systems, North-Holland 1993:309-320.

[25] Vasilevskii M.P.: Failure Diagnosis of Automata. Cybernetics, 1973; 4:653-665.

[26] Yevtushenko N., Petrenko A.: Synthesis of Test Experiments in Some Classes of Automata. Automatic Control and Computer Sciences, 1990; 24(4):50-55.

[27] Yevtushenko N., Lebedev A., and Petrenko A.: On Checking Experiments with Nondeterministic Automata. Automatic Control and Computer Sciences, 1991, 6:81-85.