Refining Specifications in Adaptive Testing of Nondeterministic Finite State Machines

Alexandre Petrenko [+] and Nina Yevtushenko *
[+] CRIM, Centre de recherche informatique de Montréal, 550 Sherbrooke Street West, Suite 100, Montreal, H3A 1B9, Canada
petrenko@crim.ca
* Tomsk State University, 36 Lenin Street, Tomsk, 634050, Russia
ninayevtushenko@yahoo.com

Abstract. This paper addresses testing of nondeterministic FSMs. An implementation FSM is allowed to be less nondeterministic than its specification, so the reduction relation between machines is the conformance relation. Execution of a given test needs to be adaptive to avoid application of unneeded input sequences, since the test can have different inputs depending on output produced by the implementation under test in response to previous input. To further reduce testing efforts during test execution, the paper suggests performing refinement of the specification FSM by removing traces absent in the implementation, which has already passed some subtest of a given complete (sound and exhaustive) test. The test execution process can be terminated as soon as the executed so far subtest is complete for the current refined version of the specification. The sufficient conditions formulated in the paper can be used for this purpose.

1. Introduction

The need for adaptive tests for nondeterministic machines has widely been discussed in various work, see, e.g., [1-10], [13, 14], [17-21]. When testing a nondeterministic implementation FSM each input sequence used in a given test needs to be repeatedly executed to make sure that all the implemented traces with this input sequence are observed. The number of repetitive applications of a sequence of input stimuli to satisfy the all-the-weather, aka complete testing assumption [11], is application dependent; however, it is even unknown when it is sufficient to terminate this process in a general case. To the best of our knowledge, the work of Hwang et al. [8] is the only attempt in determining the repetition numbers of test sequences in a probabilistic setting. However, there has been little work done on minimizing the number of input stimuli of a given test that need to be executed against a given possibly nondeterministic implementation FSM.

In this paper, we consider the offline testing of nondeterministic FSMs. As opposed to online testing, where test generation and execution are merged into a single process, see, e.g., [3], [14], in offline testing, tests are first generated from a given specification and then executed against a given implementation. This type of testing avoids repetitive test generation especially when several implementations (or its versions) have to be tested against a given specification. In this setting, the overall testing efforts can be reduced by minimizing a part of given tests that needs to be executed against a particular implementation.

If both, a specification and its implementations are allowed to be nondeterministic then the reduction relation between FSMs is a natural choice for a conformance relation, since an Implementation Under Test (IUT) is allowed to be less nondeterministic than its specification. Several publications in the area use the notion of a test from the case of deterministic specification FSMs; where tests are simply input sequences. This creates a number of problems in defining an adaptive test and formalizing its adaptive execution.

For this reason, an adaptive test case is presented as a tree in [2, 4-7, 9, 17], where in each node only a single input triggers outgoing transitions.

Hierons in [4, 5], derives an adaptive test assuming that implementation FSMs are deterministic. The idea is to use candidate machines which may model the behavior of an implementation FSM, in other words, to refine the fault domain which contains all possible implementation machines. As a result, the algorithm terminates when an implementation FSM fails the test or is recognized up to distinguishable states. Thus, this approach deals with the problem of identification of an implementation FSM, while the initial problem statement concerns just checking of the implementation which is usually less expensive than machine identification. Hierons and Ural [7] also address the problem of adaptive execution of test sequences against a deterministic implementation. The proposed approach consists in finding such an ordering of input sequences that minimizes the total number of input sequences that needs to be executed. In [6], Hierons extends the approach for the case when an implementation FSM can be non-deterministic. In this paper, in order to terminate traces of a test earlier than in [16] Hierons refines not the fault domain but the product of the specification and an implementation FSM when the latter already produced expected output responses to some input sequences. Termination rules and algorithms in [4-6] depend on the number of states of an implementation FSM, and should be completely reconsidered when extending the methods proposed there to another fault domain. In [17], we elaborated an approach to test generation using the notion of a test as a tree FSM. A complete test is defined as sound and exhaustive, i.e., each conforming IUT (a reduction of the specification FSM) has to pass the test, while each non-conforming implementation (not a reduction of the specification FSM) of the fault domain has to fail the test.

All the known methods for deriving tests with respect to the reduction relation keep the same original specification FSM during test generation, while in this paper, we propose to refine the specification FSM by removing traces absent in the implementation, which has already passed some subtest of a given complete (sound and exhaustive) test. A refined specification usually becomes more deterministic and thus, a shorter complete test may exist for the refined specification FSM. The test execution process can be terminated as soon as the executed so far subtest is complete for the current refined version of the specification. The sufficient conditions for the completeness of a test with respect to the reduction relation when the upper bound on the number of states of an IUT is known, formulated in the paper can be used for this purpose. This approach does not differentiate between deterministic and nondeterministic implementation FSMs and can be easily extended to other fault domains with respect to which sufficient conditions for the test completeness are known. A proposed approach is based on checking experiments with FSMs, which are usually known to be shorter that the identification experiments; however, additional research is needed to compare the effectiveness of the proposed approach and that of the approach in [4] when an implementation FSM is deterministic.

The paper is organized as follows. Section 2 contains basic definitions related to nondeterministic finite state machines including the reduction relation. Section 3 defines a test as a nondeterministic acyclic FSM. A method for adaptive test execution of a given complete test against a given possibly nondeterministic implementation FSM which is based on a consecutive refinement of a given specification is elaborated and illustrated on an example in Section 4. Section 5 concludes the paper.

2. Definitions

Definition 1. A *Finite State Machine* (FSM) A is a 5-tuple (S, s_0, I, O, h), where

- S is a finite set of states with the initial state s_0;
- I and O are finite non-empty disjoint sets of inputs and outputs, respectively;
- h is a behavior function h: $S \times I \rightarrow 2^{S \times O}$, where $2^{S \times O}$ is the powerset of $S \times O$. The function h has two projections h^1 and h^2: for each $(s, a) \in S \times I$, it holds that $h^1(s, a) = \{s' \mid \exists o \in O \text{ s.t. } (s', o) \in h(s, a)\}$ and $h^2(s, a) = \{o \mid \exists s' \in S \text{ s.t. } (s', o) \in h(s, a)\}$.

We slightly abuse the notation h by referring to the behavior function extended to input words.

An input a is a *defined* input at state s if $h(s, a) \neq \varnothing$; otherwise, an input a is *undefined* at state s.

A word $\alpha \in (I{\times}O)^*$ of the automaton $A_\times$ in state s is a *trace* of A in state s; let $Tr(s)$ denote the set of all traces of A in state s, while $Tr(A)$ denote the set of traces of A in the initial state. Given a trace set T, we denote $Max(T)$ the set that contains *completed* trace of the set T, i.e., those traces are not proper prefixes of the other traces of T. Given sequence $\alpha \in (I{\times}O)^*$, the *input projection* of α, denoted $\alpha_{\downarrow I}$, is a sequence obtained from α by erasing symbols in O. Input sequence $\beta \in I^*$ is a *defined* input sequence in state s of A if there exists $\alpha \in Tr(s)$ s.t. $\beta = \alpha_{\downarrow I}$. We use $\Omega(s)$ to denote the set of all defined input sequences for state s and $\Omega(A)$ for the state s_0, i.e., for A. $\Omega(A) = Tr(A)_{\downarrow I} = I^*$ holds for any complete machine A, while for a partial FSM $\Omega(A) \subseteq I^*$.

We use the following categorization of state machines.

Definition 2. FSM $A = (S, s_0, I, O, h)$ is

- *completely specified* (a complete FSM) if $h(s, a) \neq \varnothing$ for all $(s, a) \in S \times I$;
- *partially specified* (a partial FSM) if $h(s, a) = \varnothing$ for some $(s, a) \in S \times I$;
- *trace-harmonized* if for each sequence $\alpha a \in \Omega(A)$ input a is defined at each state $h^1(s_0, \alpha)$;
- *trivial*, if $S = \{s_0\}$ and $h(s_0, a) = \varnothing$ for all $a \in I$;
- *state deterministic* if $|h^1(s, a)| \leq 1$ for all $(s, a) \in S \times I$;
- *output deterministic* if $|h^2(s, a)| \leq 1$ for all $(s, a) \in S \times I$;
- *deterministic* if $|h(s, a)| \leq 1$, i.e., if it is state and output deterministic;
- *nondeterministic* if $|h(s, a)| > 1$ for some $(s, a) \in S \times I$;
- *observable* if the automaton $A_\times = (S, s_0, I \times O, \delta)$, where $\delta(s, (a,o)) = s'$ iff $(s', o) \in h(s, a)$, is deterministic;
- *acyclic* if its transition diagram is acyclic;
- *output-complete* if for each pair (s, a), $h(s, a) = \varnothing$ or $h^2(s, a) = O$;
- *regular* if it is observable, output-complete, acyclic and has at each state at most one defined input; the set of its traces is called *regular*.

We define several relations between states of observable and traced-harmonized FSMs.

Definition 3. Given observable and traced-harmonized FSM $A = (S, s_0, I, O, h)$ and $s, t \in S$,

- s and t are (*trace*-) *equivalent*, $s \cong t$, if $Tr(s) = Tr(t)$;
- t is *quasi-equivalent* to s, $t \sqsupseteq s$, if $\Omega(t) \supseteq \Omega(s)$ and $\{\beta \mid \beta \in Tr(s) \mathbin{\&} \beta_{\downarrow I} = \alpha\} = \{\beta \mid \beta \in Tr(t) \mathbin{\&} \beta_{\downarrow I} = \alpha\}$ for each $\alpha \in \Omega(s)$;
- t is *trace-included* into (is a *reduction* of) s, $t \leq s$, if $Tr(t) \subseteq Tr(s)$;
- s is *not a reduction* of t, written $s \nleq t$, if $Tr(s) \nsubseteq Tr(t)$; if s is not a reduction of t and there exists an input sequence $\alpha \in \Omega(s) \cap \Omega(t)$ s.t. $\{\beta \mid \beta \in Tr(s) \mathbin{\&} \beta_{\downarrow I} = \alpha\} \nsubseteq \{\beta \mid \beta \in Tr(t) \mathbin{\&} \beta_{\downarrow I} = \alpha\}$ we use the notation $s \nleq_\alpha t$ in order to refer to the input sequence α that evidences that s is not a reduction of t; in this case, we say that a trace $\beta \in Tr(s)$ that is not a trace at state t, i.e., $\beta \notin Tr(t)$, *distinguishes* s from t;
- s and t are *r-distinguishable*, $s \not\simeq t$, if there exists a regular set T of traces s.t. $Max(T) \cap Tr(s) \cap Tr(t) = \varnothing$, called a set *r-distinguishing* s and t; then the set of traces $T \cap Tr(s)$ is the *r-distinguishing* set of s w.r.t. t, denoted $d(s, t)$, and $T \cap Tr(t)$ is the *r*-

distinguishing set of t w.r.t. s, denoted $d(t, s)$; we also use the notation $s \not\simeq_T t$ when we need to refer to the set T of traces r-distinguishing s and t.

The notion of r-distinguishability (or adaptive distinguishability) is used in several work, e.g., [15-17, 2, 6]; below we briefly sketch how states can be efficiently checked for the r-distinguishability and how corresponding r-distinguishing sets can be determined. As usual, for a trace $\alpha \in Tr(s)$, s-after-α denotes the state reached by A when it executes the trace α from state s. If s is the initial state s_0 then instead of s_0-after-α we write A-after-α.

Definition 4. Given an FSM $A = (S, s_0, I, O, h)$, states $s, t \in S$,

- s and t are $r(1)$-*distinguishable* if there exists input a s.t. $h^2(s, a) \cap h^2(t, a) = \varnothing$;
- given $k > 1$, s and t are $r(k)$-*distinguishable*, if the states are $r(k - 1)$-distinguishable or there exists an input $a \in I$ s.t. for each trace (a,b), $b \in h^2(s, a) \cap h^2(t, a)$, states s-after-(a,b) and t-after-(a,b) are $r(k - 1)$-distinguishable.

Proposition 1. States s and t are r-*distinguishable* if there exists $k > 0$ s.t. states s and t are $r(k)$-distinguishable.

□

If $h^2(s, a) \cap h^2(t, a) = \varnothing$ for some input a then the set of traces $\{(a,b) \mid b \in O\}$ r-*distinguishes* states s and t. If the states s and t are $r(k)$-distinguishable, but not $r(k - 1)$-distinguishable and there exists an input $a \in I$ s.t. for each $b \in h^2(s, a) \cap h^2(t, a)$, states s-after-(a,b) and t-after-(a,b) are $r(k - 1)$-distinguishable, while $d(s\text{-after-}(a,b), t\text{-after-}(a,b))$ and $d(t\text{-after-}(a,b), s\text{-after-}(a,b))$ are their r-distinguishing sets, respectively, then the set of traces $d(s, t) = \{(a,b) \mid b \in h^2(s, a)\} \cup \{(a,b)\gamma \mid b \in (h^2(s, a) \cap h^2(t, a)) \wedge \gamma \in d(s\text{-after-}(a,b), t\text{-after-}(a,b))\}$ is a r-*distinguishing* set of s *w.r.t.* t and $d(t, s) = \{(a,b) \mid b \in h^2(t, a)\} \cup \{(a,b)\gamma \mid b \in (h^2(s, a) \cap h^2(t, a)) \wedge \gamma \in d(t\text{-after-}(a,b), s\text{-after-}(a,b))\}$ is that of t w.r.t. s [16, 17].

The above relations could also be applied to states from different machines. Considering the disjoint union of these machines, we have $B\ r\ A$, if and only if $t_0\ r\ s_0$, where $r \in \{\leq, \not\leq, \not\simeq\}$.

There exists a simple means to characterize the maximal common behavior of two machines.

Definition 5. Let $A = (S, s_0, I, O, h)$ and $B = (T, t_0, I, O, g)$ be FSMs. The *intersection* of A and B is an FSM $A \cap B = (Q\ q_0, I, O, \varphi)$ with the state set $Q \subseteq S \times T$, the initial state $(s_0, t_0) = q_0$, and the behavior function φ: $Q \times I \rightarrow 2^{Q \times O}$, where Q is the smallest state set s.t. $((s', t'), o) \in \varphi((s, t), a)$ iff $(s', o) \in h(s, a)$ and $(t', o) \in g(t, a)$.

This operation on machines will be used in refining a given specification FSM. We also define a complete FSM which contains a given prefix-closed set of traces. This construction is useful in characterizing the set of implementation machines which have passed some test and thus shown to contain a certain set of traces of the specification machine.

Given a finite set T of traces over the sets I and O, we derive the observable machine $M(T) = \langle pref(T), \varepsilon, I, I, \lambda_T \rangle$, where the state set $pref(T)$ is the set of all prefixes of each trace of the set T with ε as the initial state, and the transition relation $\lambda_T = \{(\beta, a, \beta a) \mid \beta a \in pref(T)\}$. By construction, the set $Tr(M(T))$ coincides with the prefix closure of the set T. We complete the FSM $M(T)$ by adding a designated *dnc* state that has a self loop transition for each pair $(a, o) \in I \times O$. For each state $\beta \in pref(T)$ and each input a undefined at state β, we define $\lambda_T(\beta, a) = \{(dnc, o) \mid o \in O\}$ and denote $\hat{M}(T)$ the completed form of $M(T)$. The set $\{C \mid C \sqsupseteq M(T)\}$ contains each FSM C s.t. $Tr(C) \supseteq T$ and C is a reduction of $\hat{M}(T)$.

The following proposition indicates how refinement of an FSM can be performed by using a set of common traces and intersection of machines.

Proposition 2. Given a complete FSM $A = (S, s_0, I, O, h)$ and a subset T of traces of A, for each complete FSM $B \in \{C \mid C \sqsupseteq M(T)\}$, if $B \leq A$ then $B \leq A \cap \hat{M}(T)$.

Proof. Consider FSM $B = (Q, q_0, I, O, f)$ of the set $\{C \mid C \sqsupseteq M(T)\}$, i.e., $B \sqsupseteq M(T)$. Given a trace $\gamma \in Tr(A)$, let α be the longest prefix of γ that is in the prefix closure of T, i.e., $\gamma = \alpha\beta$. Then, by definition of $\hat{M}(T)$, $\alpha\beta \in Tr(\hat{M}(T)$. Thus, $\gamma \in Tr(A \cap \hat{M}(T))$.

□

3. FSM Tests

In this paper, we assume that a specification FSM is a complete observable, but not necessarily deterministic, machine, while any implementation FSM is complete, but not necessarily deterministic and observable. One could use a standard procedure for automata determinization to convert a given FSM into an observable one.

Definition 6. Given initially connected FSM $U = (T, t_0, I, O, \lambda)$, U is a *test* if it is an observable acyclic output-complete FSM with the designated ***pass*** and ***fail*** states, s.t. $\lambda(t, a) = \emptyset$ implies $t \in \{$***pass***, ***fail***$\}$ and if $($***fail***$, o) \in \lambda(t, a)$ then t-after-$(a,o')\beta =$ ***pass*** for some $o' \neq o$ and $(a,o')\beta \in (I \times O)^*$.

We use the notion of a *trivial* test to denote a trivial FSM with the single initial ***pass*** state where no input is defined. Given a test U, we further refer to traces which take U from the initial state to the ***fail*** state as to *fail* traces and denote $Tr^{\text{fail}}(U)$ the set of all fail traces. *Pass* traces are defined as follows, $Tr^{\text{pass}}(U) = Tr(U) \backslash Tr^{\text{fail}}(U)$.

Note that inputs (outputs) of the specification machine are also inputs (outputs) of a test, so a tester, executing such a test, applies inputs of the test to an implementation under test, observes its outputs and produces a corresponding verdict after observing completed traces. The tester produces the verdict *fail* whenever the implementation FSM executes an input/output sequence that takes the test to the state ***fail***.

The test may have transitions with different inputs from a same state. In several work, it is not allowed to ensure the controllability of test execution. We leave the choice of inputs to a tester; executing such a test it simply selects one among alternative inputs during a particular test run. To execute another input, the tester first uses a reliable reset operation assumed to be available in any implementation to reset the implementation to its initial state and then re-executes the preamble taking the test to this input. Test execution continues until no more executable input sequences in the test are left. A strategy to achieve this is considered as an implementation detail of the tester and is not discussed here. Note that a test, as defined above, corresponds to what is called a test suite in other work, where a test case has no choice between inputs. We also assume that all possible output reactions of a non-deterministic implementation to any input sequence can be observed in testing by repetitive application of the sequence (the complete testing assumption [11]).

Given two tests $U_1 = (T_1, t_{10}, I, O, \lambda_1)$ and $U_2 = (T_2, t_{20}, I, O, \lambda_2)$, U_1 is a *subtest* of U_2 if $Tr(U_1) \subseteq Tr(U_2)$ and $Tr^{\text{pass}}(U_1) \cap Tr^{\text{fail}}(U_2) = \emptyset$. To combine tests, we define the union of tests.

Definition 7. Let $U_1 = (T_1, t_{10}, I, O, \lambda_1)$ and $U_2 = (T_2, t_{20}, I, O, \lambda_2)$ be two tests s.t. the sets $Tr^{\text{pass}}(U_1)$ and $Tr^{\text{fail}}(U_2)$ as well the sets $Tr^{\text{pass}}(U_2)$ and $Tr^{\text{fail}}(U_1)$ are disjoint and $T_1 \cap T_2 = \{$***pass***, ***fail***$\}$. The *union* $U_1 \cup U_2$ of tests U_1 and U_2 is the initially connected observable form of an FSM (Q, p_0, I, O, ψ), where $Q = \{p_0\} \cup T_1 \cup T_2$, $p_0 \notin T_1 \cup T_2$, and the behavior function $\psi : Q \times I \to 2^{Q \times O}$ is defined as follows: for each $a \in I$ it holds that $\psi(p_0, a) = h(s_0, a) \cup g(t_0, a)$; for each $s \in T_1$, $\psi(s, a) = h(s, a)$, and for each $t \in T_2$,

$\psi(s, a) = g(t, a)$. A trace γ of $U_1 \cup U_2$ that is not a proper prefix of another trace is a pass trace if γ is a pass trace of U_1 or U_2; otherwise, γ is a fail trace of $U_1 \cup U_2$.

Given a state s of FSM A and a finite prefix closed set T of traces at state s, we derive the test $U_s(T)$ called the *test closure* of the set T in state s. For each prefix $\mu(i,o) \in (I\times O)^*$ of each trace of the set T, the test $U_s(T)$ has a fail trace $\mu(i,o')$ for each $o' = O$ s.t. $\mu(i,o') \notin T$. Each trace $\gamma \in T$ that is not a proper prefix of another trace is a pass trace of $U_s(T)$.

To characterize conformance in this paper, we use the reduction relation (Definition 3). Recall that reduction conformance relation requires that in response to each input sequence a conforming implementation FSM produces only output sequences of the specification FSM. Given a complete observable FSM $A = (S, s_0, I, O, h)$, let $\Im(A)$ be a set of complete (implementation) machines over the input alphabet I and the output alphabet O, called a *fault domain*. FSM $B \in \Im(A)$ is a *conforming* implementation machine of A w.r.t. the reduction relation if $B \leq A$.

Definition 8. Given the specification FSM A, a test $U = (T, t_0, I, O, \lambda)$, and an implementation FSM $B \in \Im(A)$,

- *B passes* the test U, if the intersection $B \cap U$ has no state where the FSM U is in the state ***fail***. The test U is *sound* for FSM A in $\Im(A)$ w.r.t. the reduction relation, if any $B \in \Im(A)$, which is a reduction of A, passes the test U.
- *B fails* U if the intersection $B \cap U$ has a state where the FSM U is in the state ***fail***. The test U is *exhaustive* for FSM A in $\Im(A)$ w.r.t. the reduction relation, if any $B \in \Im(A)$, which is not a reduction of A, fails the test U.
- The test U is *complete* for FSM A in $\Im(A)$ w.r.t. the reduction relation, if it is sound and exhaustive in $\Im(A)$ w.r.t. the reduction relation.

The set $\Im(A)$ that contains all complete FSMs with at most m states is denoted $\Im_m(A)$. A test is *m-complete* if it is complete in the fault domain $\Im_m(A)$. Clearly, an m-complete test is also complete in any subset of $\Im_m(A)$, i.e., an m-complete test is also k-complete for any $k < m$.

The following proposition states that a test complete for a refined specification machine is also complete for the original specification machine once all machines which do not possess traces used in the process of refinement are excluded from the fault domain.

Proposition 3. Given the complete specification FSM A, let T be a subset of traces of A. If a test W is complete for $A \cap \hat{M}(T)$ in $\Im(A)$ then W is complete for A in $\Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$.

Proof. Let W be a complete test for $A \cap \hat{M}(T)$. Consider FSM $B \in \Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$ that is a reduction of A. Due to Proposition 2, B is a reduction of $A \cap \hat{M}(T)$ and thus, B passes the test W, i.e., W is a sound test for A in $\Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$. Let now $B \in \Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$ be not a reduction of A. In this case, B is not a reduction of $A \cap \hat{M}(T)$, i.e., B fails the test W, and thus, W is exhaustive for A in $\Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$.

Since W is sound and exhaustive for A in $\Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$, the test W is complete for A in $\Im(A) \cap \{C \mid C \sqsupseteq M(T)\}$.

□

4. Testing Nondeterministic Machines

4.1. Sufficient conditions of the test *m*-completeness

In this section, we formulate conditions stating when a test is m-complete for a given nondeterministic specification machine, mostly relying on a test generation method which we elaborated in [17], where, however, sufficient conditions are not stated explicitly.

Given a specification FSM A, states $s, p \in S$ and trace $\beta \in Tr(s)$, let $pref_{s,p}(\beta)$ be the set $\{\omega \mid \omega \in pref(\beta) \wedge s\text{-after-}\omega \le p\}$. We define a relation $\le_{s,p}$ on the set $pref_{s,p}(\beta)$, s.t. $\omega \le_{s,p} \omega'$ for $\omega, \omega' \in pref_{s,p}(\beta)$, if $|\omega| \le |\omega'|$ and $s\text{-after-}\omega \le s\text{-after-}\omega'$.

Proposition 4 [17]. Given a specification FSM A, states $s, p \in S$ and trace $\beta \in Tr(s)$, the relation $\le_{s,p}$ is a partial order on the set $pref_{s,p}(\beta)$.

Let $\le_{s,p}$ be a partial order relation on the set $pref_{s,p}(\beta)$. Given a chain $C_{s,p}(\beta)$ of the poset $(pref_{s,p}(\beta), \le_{s,p})$, the length of $C_{s,p}(\beta)$ is denoted $l(C_{s,p}(\beta))$. By definition, we assume that $l(C_{s,p}(\beta)) = 0$ if the set $pref_{s,p}(\beta)$ is empty.

A state $s \in S$ is *deterministically* reachable (d-reachable) in FSM A if there exists an input sequence χ s.t. for each trace α of FSM A with the input projection χ it holds that $A\text{-after-}\alpha = s$; in this case, the set of all traces of FSM A with the input projection χ is a *d-transfer* set for the state s. Given a set S_d of deterministically reachable states of FSM A, we determine a minimal subset (a set is minimal w.r.t. the inclusion) of the set S_d, called a *d-core* of the set S_d, that contains the initial state and, for each state $s \in S_d$, a state that is a reduction of s. A set K of traces of A is a d-core *cover* of the set S_d if K has the empty sequence and for each state s in the d-core of the set S_d, the set K contains a d-transfer set for the state s. Given a subset R of states of A, we denote $R_d = R \cap S_d$ and $K_d(R)$ a d-core cover of the set R_d. For each state $p \in R \cap S_d$, the set $K_d(R)$ contains a d-transfer set for a state that is a reduction of state p.

A set $R \subseteq S$ s.t. for all $p_1, p_2 \in R$, $p_1 \neq p_2$ implies $p_1 \not\simeq p_2$ is called a *set of pairwise r-distinguishable states* of A, we use $\mathcal{R}_A$ to denote the set of all such sets of states of A. Note that a trivial test is m-complete when there exists $R \in \mathcal{R}_A$ s.t. $|R| > m$. The reason is that in this case, the fault domain $\Im_m(A)$ has no conforming implementation FSM. In the following, we assume that $m \ge |R|$ for all $R \in \mathcal{R}_A$. We also assume that each singleton $R = \{p\}$ is an item of $\mathcal{R}_A$; in this case, by definition, the empty sequence is said to r-distinguish states of R.

Theorem 1. Given a complete specification FSM A, let U be a sound test for A in $\Im_m(A)$. The test U is m-complete for A if each trace of U that is not a trace of A is a fail trace and there exists a set S_d of d-reachable states of A, a d-core cover K_d of the set S_d with the following properties:

- $\varepsilon \in K_d$;
- for each trace $\alpha \in K_d$, $h^1(s_0, \alpha) = \{s\}$, and each sequence β of length mn that is a trace of A, it holds that there exists a pass trace in U with a prefix γ of β s.t. for some set $R \in \mathcal{R}_A$ and for each $p \in R$ there exists a chain $C_{s,p}(\gamma)$ s.t. $\Sigma_{p\in R} l(C_{s,p}(\gamma)) + |R \cap S_d| = m + 1$, and for each pair of r-distinguishable states s_1 and s_2, the set $Tr(U)$ contains the sets $\mu d(s_1, s_2)$ and $\chi d(s_2, s_1)$ where $A\text{-after-}\mu = s_1$, $A\text{-after-}\chi = s_2$, and $\mu, \chi \in \{\nu \mid \nu \in K_d(R) \cup \cup_{p\in R} \alpha C_{s,p}(\gamma)\}$.

Proof. According to Proposition 13 [17], given $B = (T, t_0, I, O, g)$, $B \in \Im_m(A)$, $B \not\le A$, that passes the test U, there exist $\alpha \in K_d$, $h^1(s_0, \alpha) = \{s\}$, $\alpha\gamma \in U$, s.t. for any set $R \in \mathcal{R}_A$

if $\Sigma_{p\in R} l(C_{s,p}(\gamma)) + |R \cap S_d| = m + 1$ then there exist (s', t) and (s'', t), $s' \not\simeq s''$, in the set $\{(A \cap B)\text{-after-}\omega \mid \omega \in (K_i(\gamma) \cup \cup_{p\in R} \alpha C_{s,p}(\gamma))\}$, where $K_i(\gamma) \subseteq K_d$ and for each $r \in (R \cup \{A\text{-after-}\omega \mid \omega \in \cup_{p\in R} \alpha C_{s,p}(\gamma)\})$ that has a reduction in the d-core of S_d there exists $\sigma \in K_i(\gamma)$ s.t. A-after-$\sigma \leq r$. Let sequences ω_1 and ω_2 take $A \cap B$ to (s', t) and (s'', t). According to Theorem 1, the set $Tr(U)$ contains the sets $\omega_1 d(s', s'')$ and $\omega_2 d(s'', s')$. By the definition of the sets $d(s', s'')$ and $d(s'', s')$, the set of traces of FSM B at state t is not contained in $d(s', s'')$ or in $d(s'', s')$, and thus, B fails the test U.

□

For more detail and explanation on m-complete test generation for nondeterministic FSM, we refer the reader to our previous work [17].

According to Theorem 1, the length of an m-complete test does not significantly depend on the number of states of the specification FSM, it rather depends on the sets of d-reachable and r-distinguishable states and the degree of nondeterminism in the specification FSM. Thus, it is worth to check if given an FSM $A = (S, s_0, I, O, h)$ and a natural m, there exists a more deterministic reduction A_m of the FSM A s.t. for each complete FSM C with at most m states, it holds that if $C \leq A$ then $C \leq A_m$. FSM A_m is called an *m-reduction* of A. According to Theorem 1, we can expect that a shorter m-complete test can be derived for the specification FSM A_m. A straightforward approach to derive an m-reduction of A is to explicitly enumerate all possible reductions of A with m states, derive the union of these reductions and intersect the union with the FSM A. However, such an approach is impractical, since the number of possible reductions of a given FSM with the given number of states can be exponential. On the other hand, it is difficult to know in advance whether an obtained m-reduction will be simpler than the given specification FSM, for example, whether an m-reduction has fewer states than the FSM A. Moreover, it is an open question what is the relationship between two m-reductions; i.e., whether any two m-reductions of a given FSM are equivalent. All the above issues need additional research before using m-reductions in test derivation.

Another way of making the specification FSM more deterministic is to use some knowledge about an implementation FSM which can be obtained through testing. For example, if an implementation FSM is more deterministic that the specification then after applying some inputs to the implementation FSM and observing the corresponding output sequences, we could refine the specification FSM. For some applied input sequence, only fewer traces with this input projection should be left in the specification and if T is the set of such traces then the machine $\hat{M}(T)$ is the "loosest" machine that can represent each implementation FSM which has T as a subset of the trace set. According to Proposition 3, a complete test can now be derived for a more deterministic specification FSM $A \cap \hat{M}(T)$ and due to Theorem 1; this test is expected to be shorter than the one derived for the initial, more nondeterministic, specification FSM A. In the following section, we formally describe such a refinement of the specification FSM in adaptive test execution.

4.2. Adaptive Test Execution Method

A simple minded test execution method would try to repeatedly execute each and every sequence of input stimuli used in a given test against an implementation under test and terminate its application only when all the produced traces take the test into pass state or a trace occurs which takes the test into fail state. As an example, consider the specification FSM shown on the left-hand side of Figure 1 and the test in Figure 2. The trace $(a,1)(a,0)$ is one of the shortest traces of the test, this trace uses a shortest sequence of input stimuli of length two, namely aa, not counting the reset input. At the same time, if a given implementation produces in response to the input sequence aa the output sequence 02 instead of 10, the input sequence aa needs be extended to a longest input sequence, e.g., *abbaa*. The test shown in Figure 2 contains 13 different input sequences of the total length of 65 inputs (including resets and excluding prefixes) which need to be executed in the worst case, when, for example, the implementation is equivalent to the specification machine. Moreover, in this case, some input sequences should be executed several times in order to allow an IUT to produce all possible output responses to each input sequence.

The problem we are trying to address in this section is as follows: can the results of executing a part of a given test be used to terminate the test execution process earlier?

The idea of the proposed solution is to refine the specification FSM, by removing traces absent in a given implementation under test, which has already passed some subtest of a given complete test. The test execution process can be terminated as soon as the executed so far subtest is complete for the current refined version of the specification. The sufficient conditions formulated in Section 3 can be used for this purpose. This idea leads to the following.

Adaptive test execution method for checking whether an implementation under test with at most m states is a reduction of the specification FSM

Input. A complete FSM $A = (S, s_0, I, O, h)$, an m-complete test $U = (T, t_0, I, O, \lambda)$ for FSM A w.r.t. the reduction relation, and an implementation FSM B in the form of a procedure $Procedure(V)$ which for a given test V returns ***fail*** if B fails V or if B passes V the non-empty set of completed traces $Tr^{\text{pass}}(V) \cap Tr(B)$.

Output. The verdict ***fail*** if B is a non-conforming implementation and the verdict ***pass*** if B is a conforming implementation as well as a subtest G of the test U that is a complete test for A in $\mathfrak{I}_m(A) \cap \{C \mid C \sqsupseteq M(Tr^{\text{pass}}(G))\}$ and a refined specification A_i s.t. for each complete FSM $D \in \mathfrak{I}_m(A) \cap \{C \mid C \sqsupseteq M(Tr^{\text{pass}}(G))\}$, it holds that if $C \leq A$ if and only if $C \leq A_i$.

1. Initialize G as a trivial test, $i := 1$, $A_i := A$.
2. Choose an arbitrary subtest U' of U s.t. $U'{\downarrow_I} \not\subset G{\downarrow_I}$ and for each pass trace $\alpha \in U'$, the subtest U' contains all the pass traces of U with the input projection $\alpha{\downarrow_I}$ and go to Step 3. If there is no such subtest then go to Step 7.
3. Call $Procedure(U')$, where $Procedure(U') = Tr^{\text{pass}}(U') \cap Tr(B)$ if it does not return ***fail***, else stop and return the verdict ***fail***.
4. Derive the test closure of $Procedure(U')$ in the initial state, i.e., the test $U_i = U_{s_0}(Procedure(U'))$; $G := G \cup U_i$.
5. Determine and reduce the FSM $\hat{M}\,(Procedure(U'))$ by merging its equivalent states.
6. Increment i by one and derive $A_i := \hat{M}\,(Procedure(U')) \cap A_{i-1}$. If the test G is not complete for A_i in $\mathfrak{I}_m(A_i)$ according to the sufficient conditions of Theorem 1, then delete from U each trace which contains $\alpha \in Tr^{\text{fail}}(G)$ as its proper prefix and replace the ***pass*** state U-after-α by ***fail*** state and go to Step 2.
7. Return the complete test G for A in the set $\mathfrak{I}_m(A) \cap \{C \mid C \sqsupseteq M(Tr^{\text{pass}}(G))\}$ and produce the verdict ***pass***.

Theorem 2. The above procedure terminates and produces the ***pass*** verdict if and only if B is a reduction of A.

Proof. The verdict ***fail*** can be produced in one of the intermediate steps if and only the IUT has a trace that is a fail trace of the test U and thus, the verdict ***fail*** can be produced if and only the IUT is not a reduction of A.

Let the test G satisfy sufficient conditions of Theorem 1 for A_i in $\mathfrak{I}_m(A_i)$ and the verdict ***fail*** was not produced in one of the intermediate steps, an implementation at hand is a reduction of A_i and thus, is a reduction of A.

If in Step 2 no new subtest can be chosen then $G = U_{s_0}(T)$, where T = *Procedure*(U). Three cases are possible for an FSM $B \in \mathfrak{I}(A)$: 1) B is a reduction of $\hat{M}(T) \cap A$; 2) B is a not a reduction of $\hat{M}(T) \cap A$, but B is a reduction of A; 3) B is not a reduction of A.

Case 1: Consider a trace $\gamma \in U_{s_0}(T) \cap Tr(B)$. If $\gamma \in T$ then, by construction of $U_{s_0}(T)$, γ is a pass trace in $U_{s_0}(T)$. If $\gamma \notin T$ then γ is a prefix of a pass trace in $U_{s_0}(T)$, since U is sound for A in $\mathfrak{I}(A)$.

Case 2: There exists a trace $\gamma \in Tr(B)$ s.t. $\gamma_{\downarrow I} \in T_{\downarrow I}$. Denote μ the shortest prefix of γ that is not a trace of $\hat{M}(T) \cap A$. By construction of $U_{s_0}(T)$, μ is a fail trace in $U_{s0}(T)$.

Case 3: FSM B fails the test U, i.e., FSM B fails the test $U_{s_0}(T)$, as the latter preserves all the fail traces of U.

□

We now comment on the complexity of the above algorithm. The length of a resulting test suite can only decrease; in the worst case the whole initial test suite will be applied to an IUT. The intersection of two FSMs has the polynomial complexity w.r.t. the number of their states. The union of two FSMs, generally, has the exponential complexity according to the determinization operator. However, in Step 4, the input projections of G and U_i do not intersect and thus, there is no need to use the determinization operator. We also can stop to check the sufficient conditions in Step 6 any time facing increasing complexity; though, in this case, a longer resulting test suite might be obtained. Thus, it is always possible to execute the algorithm in polynomial time.

The method is illustrated in the next section.

4.3. Example

Consider the specification and implementation machines, shown in Figure 1. The specification FSM A with three states is nondeterministic, while the implementation FSM B with two states is deterministic and chosen to be a reduction of A. We use the FSM B as an implementation of the procedure *Procedure*(V) which for a given test V returns the non-empty set $Tr^{\text{pass}}(V) \cap Tr(B)$ of completed pass traces, since B is a conforming implementation in this example.

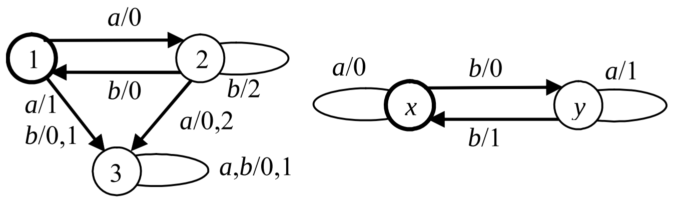

Fig. 1: The specification A and implementation B FSMs; initial states are depicted in bold.

Assuming that a given implementation has at most two states (as the FSM B), i.e., m = 2, we generate an m-complete test U for FSM A w.r.t. the reduction relation using the method [17]. Note that any two states of the specification A are not r-distinguishable; states 1 and 3 are d-reachable, thus, S_d = {1, 3}, the d-core cover K_d = {ε, (b,{0, 1})}, where (b,{0, 1}) = {(b,0), (b,1)}. The 2–complete test U is shown in Figure 2, where the nodes inherit the names of states of the FSM A, note that all the deadlock states are ***pass*** states. To reduce the clutter, all the fail traces are omitted, for example, two transitions from the initial state to the ***fail*** state labeled a/2 and b/2 are not shown in the figure.

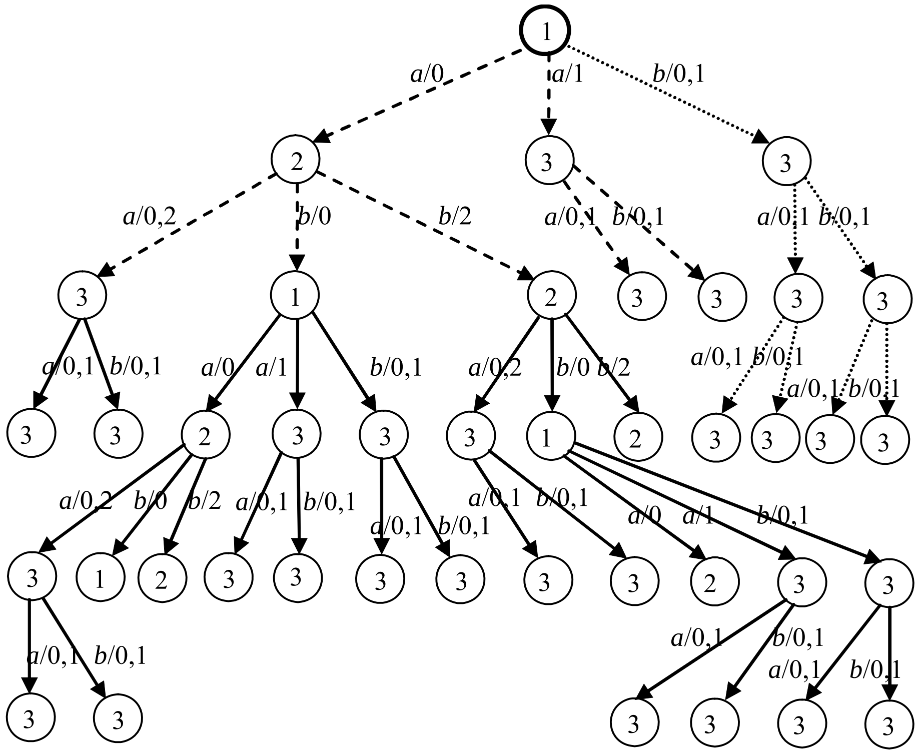

Fig. 2: The pass traces of 2-complete test *U* for the FSM *A* w.r.t. the reduction relation.

Executing the first step, we initialize *G* as a trivial test, *i*:= 1, A_1:= *A*. Next, we select the right-hand side subtree in Figure 2 (depicted with dotted transitions) for a subtest *U*′ with the following input projections {*baa*, *bab*, *bba*, *bbb*)}. The set of completed pass traces $Tr^{\text{pass}}(U')$ = {(*b*,{0, 1})(*a*,{0, 1})(*a*,{0, 1}); (*b*,{0, 1})(*a*,{0, 1})(*b*,{0, 1}); (*b*,{0, 1})(*b*,{0, 1})(*a*,{0, 1}); (*b*,{0, 1})(*b*,{0, 1})(*b*,{0, 1})} contains 32 traces. The procedure *Procedure*(*U*′) returns the set of completed traces $Tr^{\text{pass}}(U') \cap Tr(B)$ = {(*b*,0)(*a*,1)(*a*,1), (*b*,0)(*a*,1)(*b*,1), (*b*,0)(*b*,1)(*a*,0), (*b*,0)(*b*,1)(*b*,0)}.

In the forth step, we obtain the test closure of *Procedure*(*U*′) in the initial state, i.e., the test $U_1 = U_{s_0}(Procedure(U'))$ (Figure 3).

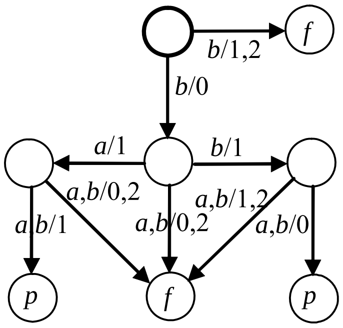

Fig. 3: The test $U_1 = U_{s_0}(Procedure(U'))$, where *p* denotes ***pass*** and *f* - ***fail*** states.

The fifth step returns the FSM $\hat{M}$ (*Procedure*(*U*′)), which represents all FSMs which can pass the test U_1, shown in Figure 4. The FSM has no equivalent states.

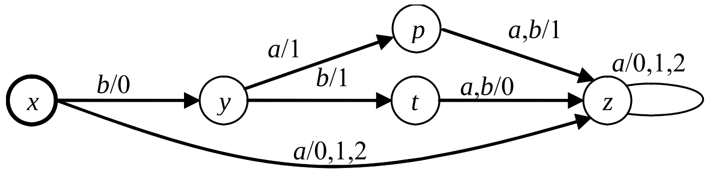

Fig. 4: The FSM $\hat{M}$ (*Procedure*(U')) in the first iteration.

Next, we construct the first refined specification A_2 = $\hat{M}$ (*Procedure*(U')) $\cap$ A_1 (Figure 5).

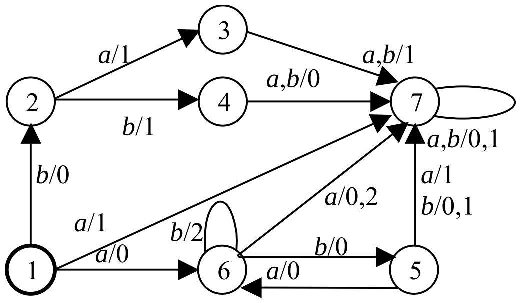

Fig. 5: The FSM A_2 = $\hat{M}$ (*Procedure*(U')) $\cap$ A_1, where states are renamed as follows: $x1 - 1$; $y3 - 2$; $p3 - 3$; $t3 - 4$; $z1 - 5$; $z2 - 6$; $z3 - 7$.

The FSM A_2 has five d-reachable states, 1, 2, 3, 4, and 7. There are four pairs of *r*-distinguishable states, (1, 2), (1, 3), (2, 4), (4, 5). The obtained machine refines the original specification FSM *A*, its traces constitute a proper subset of that of *A*; it is "more" deterministic than *A*, at the price of having more states. We delete from *U* each trace which contains $\alpha \in Tr^{\text{fail}}(G)$ as its proper prefix. For example, all the traces with the prefix (*b*,0) will be deleted from the set of pass traces in Figure 2. The test *G* is not *m*-complete for A_2 and $m = 2$, according to Theorem 1, as it violates its sufficient conditions. For example, it contains no trace which starts with the input *a*.

Step 2 has to be executed again. To satisfy the above mentioned sufficient condition, we select another subtest U' of the 2-complete test *U* (dashed transitions in Figure 2), with input *a* followed by *a* and *b*. The set of completed pass traces $Tr^{\text{pass}}(U')$ = $\{(a,0)(a,\{0, 2\}), (a,0)(b,\{0, 2\}), (a,1)(a,\{0, 1\}), (a,1)(b,\{0, 1\})\}$ contains eight traces. Clearly, $U'_{\downarrow I} \not\subset G_{\downarrow I}$, where the test $G = U_1$, see Figure 3. The procedure *Procedure*(U') returns now the set of completed pass traces $Tr^{\text{pass}}(U') \cap Tr(B) = \{(a,0)(a0), (a,0)(b.0)\}$.

In the forth step, we obtain the test closure of *Procedure*(U') in the initial state, i.e., the test $U_2 = U_{s_0}$(*Procedure*(U')) and $G = U_1 \cup U_2$ (Figure 6).

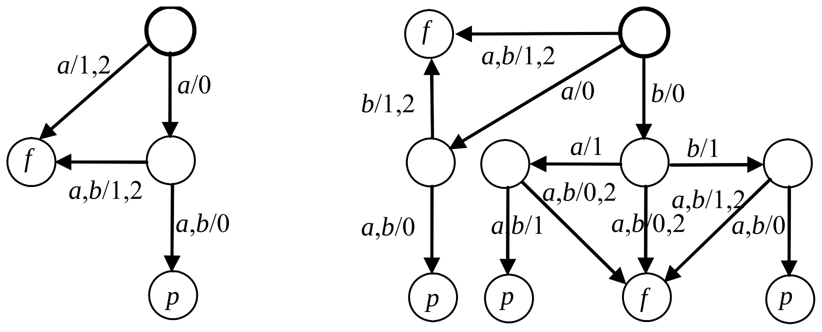

Fig. 6: The tests $U_2 = U_{s_0}$(*Procedure*(U')) and $G = U_1 \cup U_2$.

The fifth step returns now the FSM $\hat{M}$ (*Procedure*(U')), Figure 7, which represents all the FSMs as its reductions which can pass the test U_2, shown in Figure 6. The FSM has no equivalent states.

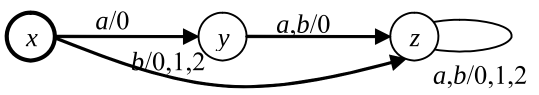

Fig. 7: The FSM $\hat{M}$ (*Procedure*(U')) in the second iteration.

The refined specification FSM becomes now $A_3 = \hat{M}$ (*Procedure*(U')) $\cap A_2$ (Figure 8).

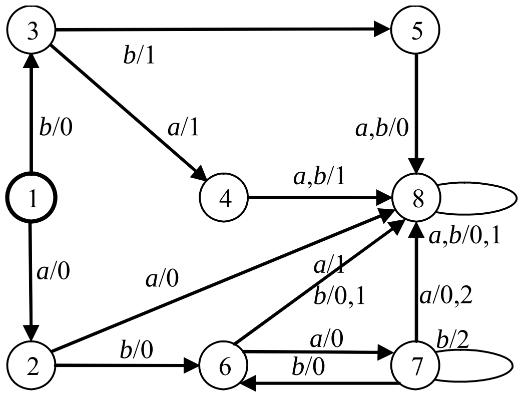

Fig. 8: The FSM $A_3 = \hat{M}$ (*Procedure*(U')) $\cap A_2$, where states are renamed as follows: $x1 – 1$; $y6 – 2$; $z2 – 3$; $z3 – 4$; $z4 – 5$; $z5 – 6$; $z6 – 7$; $x7 – 8$.

The reduction relation between states of A_2: $1 \leq 6$; $1 \leq 2 \leq 7$; $5 \leq 8$; $3 \leq 6$; $3 \leq 4 \leq 8$. Pairs of r-distinguishable states are (1, 3); (1, 4); (2, 3); (2, 4); (3, 5); (3, 7); (4, 5); (4, 7). The test G for FSM A_3 satisfies the conditions of Theorem 1, i.e., the test G is 2-complete for A_3. Due to Proposition 3, the test G is complete for A in $\mathfrak{I}_2(A) \cap \{C \mid C \sqsupseteq M(Tr^{\text{pass}}(G))\}$.

The original test U (Figure 2) contains, as mentioned above, 13 input sequences of the total length of 65 inputs (including resets) and all these sequences should be applied to the given implementation FSM B (Figure 1) if the tester does not take into account output responses produced by the FSM B, i.e., it executes this test in a preset manner.

However, the tester executes this test adaptively, i.e., it only applies a suffix of an input sequence that is defined in the test after the output response of B to the prefix of this sequence. For example, the tester would not apply the input sequence *abbab* to the FSM B, since it should be applied only when an IUT has the output response 02 to the prefix *ab*, but the FSM B replies with 00. In such adaptive execution of this test against the given deterministic implementation FSM B (Figure 1), the tester would use ten input sequences of the total length of 54.

At the same time, the proposed method needs only six input sequences of the total length of 18. This indicates that refining the specification FSM can bring a significant saving in testing efforts, especially in dealing with nondeterministic implementation FSMs which require repeated test runs driven by a same input sequence to observe all the implemented traces. In fact, each dropped input sequence of the original test would have been repeatedly executed to make sure that all the implemented traces with this input projection are observed. The saving tends to increase when a given implementation FSM is less nondeterministic than its specification FSM.

5. Conclusion

We considered testing of nondeterministic FSMs, when tests are first generated from a given specification and then executed against a given implementation FSM. This type of testing avoids repetitive test generation when several implementations (or its versions) have to be tested against a given specification. In this setting, the overall testing efforts can be reduced by minimizing a part of given tests that needs to be executed against a particular implementation. The novelty of the proposed solution lies in the iterative refinement of the original specification during adaptive test execution. The test execution terminates when the executed test is found to be complete for the refined specification FSM.

Several possible avenues for future work can be indicated. First, we have already mentioned that characterization of all possible reductions of a given FSM which have a given number of states (*m*-reductions) is an open interesting problem. Then, the proposed method does not provide any hint on which subtest of a predefined test has to be chosen at each step of the iteration. It looks like ideas similar to those proposed in [7] can be of help to address this issue. Finally, our current work is to use the idea of specification refinement in online testing, thus refining both, the specification and test.

References

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

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

[3] M. Dorofeeva, A. Petrenko, M. Vetrova, and N. Yevtushenko. Adaptive Test Generation from a Nondeterministic FSM, Radioelektronika i informatika. 2004, No. 3, pp. 91-95.

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

[5] R. M. Hierons, Testing from a Non-Deterministic Finite State Machine Using Adaptive State Counting, IEEE Transactions on Computers, 2004, 53, 10, pp. 1330-1342.

[6] R. M. Hierons. Using Candidates to Test a Deterministic Implementation against a Non-deterministic Finite State Machine, The Computer Journal, 2003, 46, 3, pp. 307-318.

[7] R. M. Hierons and H. Ural. Concerning the Ordering of Adaptive Test Sequences, 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2003), 2003, LNCS volume 2767, pp. 289-302.

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

[9] H. Kloosterman, 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, pp. 297-308.

[10] I. Kufareva, N. Yevtushenko, and A. Petrenko, Design of Tests for Nondeterministic Machines with Respect to Reduction, Automatic Control and Computer Sciences, Allerton Press Inc., USA, No. 3, 1998.

[11] G. L. Luo, G. v. Bochmann, and A. Petrenko. Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-method. IEEE Transactions on Software Engineering, 1994, 20(2), pp. 149–161.

[12] G. Luo, A. Petrenko, G. v. Bochmann. Selecting Test Sequences for Partially Specified Nondeterministic Finite State Machines, the IFIP Seventh International Workshop on Protocol Test Systems, Japan, 1994, pp. 95-118.

[13] R. Miller, D. Chen, D. Lee, R. Hao. Coping with Nondeterminism in Network Protocol Testing. Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference, TestCom 2005, Montreal, Canada, LNCS 3502, 2005, pp. 129-145.

[14] L. Nachmanson, M. Veanes, W. Schulte, N Tillmann, W. Grieskamp, Optimal Strategies for Testing Nondeterministic Systems, ISSTA 2004, Software Engineering Notes ACM, 29, pp. 55–64.

[15] A. Petrenko, N. Yevtushenko, A. Lebedev, and A. Das. Nondeterministic State Machines in Protocol Conformance Testing, Proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, France, 1993, pp. 363-378.
[16] A. Petrenko, N. Yevtushenko, and G. v. Bochmann. Testing Deterministic Implementations from their Nondeterministic Specifications, Proceedings of the IFIP Ninth International Workshop on Testing of Communicating Systems, 1996, pp. 125-140.
[17] A. Petrenko, N. Yevtushenko, Conformance Tests as Checking Experiments for Partial Nondeterministic FSM. In Proceedings of the 5th International Workshop on Formal Approaches to Testing of Software (Fates 2005), LNCS 3997, Edinburgh, Scotland, UK, 2005, pp. 118-133.
[18] J. Tretmans. Test Generation with Inputs, Outputs and Quiescence. TAGAS'96. PP. 127-146, Springer-Verlag, 1996.
[19] P. Tripathy and K. Naik, 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, pp. 309-320.
[20] F. Zhang and T. Cheung, Optimal Transfer Trees and Distinguishing Trees for Testing Observable Nondeterministic Finite-State Machines, IEEE Transactions on Software Engineering, 2003, Vol. 29, No. 1, pp. 1-14.
[21] N. Yevtushenko, A. Lebedev, and A. Petrenko. On Checking Experiments with Nondeterministic Automata. Automatic Control and Computer Sciences, 1991, 6, pp. 81–85.