

ADAPTIVE TEST GENERATION FOR NONDETERMINISTIC NETWORKS

PETRENKO A., VETROVA M., YEVTUSHENKO N.

(Design and Test)

Abstract. In this paper, we discuss transition coverage testing that is test purpose based testing. We introduce the notion of a test as a tree FSM, define preset and adaptive tests for a non-deterministic specification and demonstrate that adaptive tests achieve the above test purpose more effectively. We illustrate our approach for a so-called monolithic representation of a non-deterministic FSM.

1. Introduction

Non-deterministic networks have recently been considered in various areas of logic synthesis [1] and are usually used as a joint compact representation of deterministic behaviors. In this case, a system specification is non-deterministic while each implementation is deterministic. Testing is needed to determine whether the behavior of a given implementation is contained in that of the specification.

The problem of testing a non-deterministic network can be formulated as testing against a non-deterministic specification Finite State Machine (FSM) w.r.t. the containment (reduction) relation when each implementation is assumed to be a deterministic FSM. A number of methods have been elaborated for generating tests from a non-deterministic FSM [2, 3, 4, 5]. Some of these methods are based on test purposes and return tests with unknown fault coverage; other methods derive tests with the guaranteed fault coverage w.r.t. an appropriate fault model. The main difference of test generation from a non-deterministic specification FSM compared to deterministic ones is that tests depend not only on the specification, but also on an Implementation Under Test (IUT), as testing becomes an adaptive process of alternating between test execution and test generation, taking into account responses already produced by an IUT.

In this paper, we focus on transition coverage testing that is test purpose based testing (as opposed to fault coverage). Our test purpose is to cover transitions in deterministic submachines of the specification FSM. Our motivation is that transition covering tests are known to be of a high quality at least in the case of deterministic specifications. We introduce the notion of a test as a tree FSM, define preset and adaptive tests for a non-deterministic specification and demonstrate that adaptive tests achieve the above test purpose more effectively. We illustrate our approach for a so-called monolithic representation of a non-deterministic FSM, where the transition relation describes transitions between states of the FSM. At the same time, similar to deterministic FSM, the partitioned representation of the transition relation can be used when the relation involves not states and inputs but internal and input variables [6]. To our knowledge, this problem is solved only for deterministic specification FSMs.

This paper is organized as follows. Section 2 contains basic definitions, Section 3 presents our definition of tests. The proposed method for testing on-the-fly is in Section 4. Conclusions are given in Section 5.

2. General Definitions

Given alphabet Z , let 2^Z denote the set of all subsets of Z . A *Finite State Machine* (FSM) A is a 5-tuple (S, I, O, h, s_0) , where S is a finite set of states with the initial state s_0 , I and O are finite non-empty sets of inputs and outputs, respectively, which satisfy the condition $I \cap O = \emptyset$; h is a behavior function $h: S \times I \rightarrow 2^{S \times O}$. In this paper, we consider only observable machines; an FSM A is *observable* if the underlying automaton $A_x = (S, I \times O, \delta, s_0)$, where $\delta(s, \alpha) = s'$ iff $(s', o) \in h(s, \alpha)$ is deterministic. FSM A is *completely specified* (a complete FSM) if $h(s, \alpha) \neq \emptyset$ for all $(s, \alpha) \in S \times I$; otherwise, FSM A is *partially specified* (a partial FSM); FSM A is *deterministic* if $|h(s, \alpha)| \leq 1$ for all $(s, \alpha) \in S \times I$; otherwise, FSM A is *non-deterministic*. We denote a transition (s, a, b, s') , where $h(s, a) = (s', b)$.

A word α of the underlying automaton A_x at state s is a *trace* of A at state s ; $\text{Tr}_A(s)$ denotes the set of all traces of A at state s and Tr_A denotes the set of traces of A at the initial state. Given sequence $\alpha \in (I \times O)^*$, the *input projection* of α , denoted $\alpha_{\downarrow Z}$, is a sequence obtained from α by erasing symbols in O . An input sequence $\beta \in I^*$ is a *defined* input sequence at state s of A if there exists $\alpha \in \text{Tr}_A(s)$ such that $\beta = \alpha_{\downarrow Z}$. Given a trace $\beta \in \text{Tr}_A(s)$, s -after- β denotes the state reached by A when it executes the trace β from state s . If s is the initial state s_0 then instead of s_0 -after- β we write A -after- β .

FSM $B = (S_B, I, O, h_B, s_0)$ is a *submachine* of $A = (S, I, O, h, s_0)$ if $S_B \subseteq S$ and $h_B(s, \alpha) \subseteq h(s, \alpha)$ for each $(s, \alpha) \in S_B \times I$.

We denote Ω_A the set of defined input sequences of A in the initial state.

Given two FSM A and B , FSM B is a *reduction* of FSM A , $A \leq B$, if $\text{Tr}_B \subseteq \text{Tr}_A$. FSM B is a *quasi-reduction* of FSM A , $A \lesssim B$, if $\Omega_B \supseteq \Omega_A$ and $\{\beta \in \text{Tr}_B \mid \beta_{\downarrow I} = \alpha\} \subseteq \{\beta \in \text{Tr}_A \mid \beta_{\downarrow I} = \alpha\}$ for all $\alpha \in \Omega_A$. FSM B is *not a reduction* of FSM A , written $B \not\leq A$, if $\{\beta \in \text{Tr}_B \mid \beta_{\downarrow I} = \alpha\} \not\subseteq \{\beta \in \text{Tr}_A \mid \beta_{\downarrow I} = \alpha\}$ for some $\alpha \in \Omega_A \cap \Omega_B$; we use the notation $B \not\leq_a A$ when we need to refer to the input sequence α which detects that B is not a reduction of A . For complete FSM the quasi-reduction and the reduction relations coincide.

The reduction relation captures the notion of trace inclusion or containment (preorder) relation. Given two complete machines B and A , FSM B is a reduction of FSM A if and only if for each input sequence α , the set of output responses of the B to α is a subset of that of the FSM A . If FSM B produces an output sequence in response to some input sequence that the A cannot, then B is not a reduction of A .

3. FSM Tests

In this paper, we assume that a specification FSM A from which we generate tests is a complete observable, but not necessarily deterministic, machine, while any implementation FSM is a complete deterministic machine. In this case, the reduction relation is used to declare a verdict on conformance of an IUT.

Given input I and output O alphabets, a *test* U is an FSM $U=(T,I,O,\lambda,\varepsilon)$, where

- T is a finite prefix-closed subset of $(I \times O)^*$,
- If $\alpha x z \in T$ then $\lambda(\alpha, x) = (\alpha x z, z)$.
- Each completed trace α of the test, i.e., such that $\alpha (I \cup O)^* \cap T = \emptyset$, is a *verdict* state, labeled either *pass* or *fail*, i.e., a pass- or fail-state (trace).

The tester produces the verdict fail whenever the implementation FSM executes an input/output sequence that is a fail-trace of the test. If the implementation FSM does not execute fail-traces of the test then the verdict pass is produced. Any test of A can be obtained by unwinding the graph of A into a tree, while skipping some inputs and outputs of A . The tree structure of a test is fully determined by the set of its traces. A test may have transitions with different inputs from a same state, we assume, therefore, that a reliable reset operation is available in any implementation.

Let $\mathfrak{S}(A)$ be a set of complete deterministic (implementation) machines over the input alphabet of A , called a *fault domain*. FSM $B \in \mathfrak{S}(A)$ is a *conforming* implementation machine of A if $A \leq B$.

Given a test $U=(T,I,O,\lambda,\varepsilon)$ and an implementation FSM $B \in \mathfrak{S}(A)$,

- the test U *execution (observation)* on B is the FSM $B \cap U = (Q, I, O, \varphi, q_0)$;
- a state $(s, t) \in Q$ is a *fail-state*, if t is a verdict fail state.

- B *fails* U , if $B \cap U$ has a fail-state.
- B *passes* U , if $B \cap U$ has no fail-states.

Given a test U and a fault domain $\mathfrak{S}(A)$,

- U is *sound* in $\mathfrak{S}(A)$ for the FSM A and reduction relation, if each $A \leq B$, $B \in \mathfrak{S}(A)$, passes U .
- U is *complete* in $\mathfrak{S}(A)$ for FSM A and reduction relation, if each $B \not\leq A$, $B \in \mathfrak{S}(A)$, fails U .
- U is *exhaustive* in $\mathfrak{S}(A)$ for FSM A and reduction relation, if it is sound and complete in $\mathfrak{S}(A)$.

A test may have transitions with different outputs from a state under the same input. We call a test U a *preset* test if for each two of its traces α and β , $\alpha_{\downarrow I} = \beta_{\downarrow I}$, the test U has a trace $\alpha\gamma$ if and only if the test has a trace $\beta\kappa$, $(\beta\kappa)_{\downarrow I} = (\alpha\gamma)_{\downarrow I}$. In this case, an output produced by an IUT does not decide which next input to apply in the preset test U .

If the above property does not hold for a test, in other words, if a next move (input to the IUT) of the tester depends on an output of the IUT, the test has to be adaptively executed by the tester. In other words, in such a scenario, the tester adapts itself to the behavior of the implementation FSM. For this reason, we refer to such test as an *adaptive* test. The tree of an adaptive test can be derived in advance before test execution begins; however, usually only a part of thus obtained test may be obtained if a test is generated on-the-fly, by alternating test generation and test execution steps until some testing criterion is met. Thus, adaptive testing usually reduces computations in test generation.

4. Testing for Covering Transitions in a Specification FSM

In the case when a specification FSM is deterministic (so is any implementation FSM), a (preset) test that covers all the transitions in the specification is a transition tour of the specification FSM. It can be determined using standard graph algorithms. The problem becomes, however, much more complicated when a specification FSM is non-deterministic, while any implementation FSM is known to be deterministic. In this case, covering all the transitions of the specification FSM while testing a given implementation FSM is impossible, as the latter may only “implement” at most one transition out of all transitions sharing the same starting state and input action. Thus, in this testing scenario, the transition covering criterion has to be rephrased as follows. A test should cover transitions of deterministic submachine of the specification FSM.

A naive, straightforward approach for determining such transition coverage of the specification FSM

involves the explicit enumeration of all its submachines and can hardly constitute a satisfactory solution, since the number of submachines can be huge even for small specifications. Moreover, all these tests will never be used for testing a given implementation FSM since it is deterministic. The tester needs only such a test that is relevant to a given (though, unknown) implementation FSM. We propose a method for deriving a test providing transition coverage on-the-fly during the process of testing an implementation at hand.

Method: On-the-fly test generation for transition coverage in a non-deterministic specification FSM while testing a deterministic implementation FSM B.

Step 1. Assign $M := A$, $T = \{\varepsilon\}$, $\beta := \varepsilon$.

Step 2. Repeat the following until for a current trace β there exists an input α such that the transition from state M -after- β with input α is not colored in M :

- apply $\beta \downarrow \alpha$ to the FSM B and observe the output b in state B -after- β in response to the input α ;
- if the FSM M does not have the trace $\beta(\alpha b)$ then add this trace to the test as a fail-trace, produce the verdict fail and END; otherwise, delete from the FSM M each transition (M -after- β , α , b' , s'), where $b' \neq b$ and color in the FSM M the transition (M -after- β , α , b , s'); add the trace $\beta(\alpha b)$ to the set T; and assign $\beta := \beta(\alpha b)$.

If for each input α the transition from state M -after- β under input α is colored in the FSM M then:

- if each trace of the set T takes FSM M only to a state from which all transitions are colored then each completed trace of a test is a pass-trace, produce the verdict pass and END. Otherwise, select the shortest trace γ in T that takes FSM M to a state from which there exists an uncolored transition;
- assign $\beta := \gamma$ and repeat Step 2.

In the case of a deterministic specification FSM, the method produces a test that covers each and every transition of the specification FSM.

We claim that the proposed method delivers a test covers all transitions in the initially connected deterministic submachine of A. Recall that the initially connected part of an FSM is the submachine obtained by removing states (and their transitions) that are not reachable from the initial state.

Theorem. Let T be the test derived by the above method. If the tested implementation FSM is equivalent to a deterministic submachine of A then it passes the test, i.e., T is sound. If the verdict pass is produced then T covers all transitions of some initially connected deterministic submachine of A.

Example. Consider the specification FSM A in Figure 1 and its deterministic submachine B in Figure 2. The FSM A has two internal variables and thus, four states 00, 01, 10 and 11 with 00 as the initial state, two

input variables and a single output variable. By direct inspection, one can assure that the specification FSM has $2^{10} = 1024$ deterministic submachines.

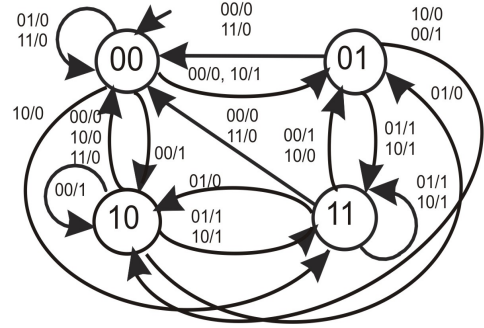


Figure 1. The specification FSM A.

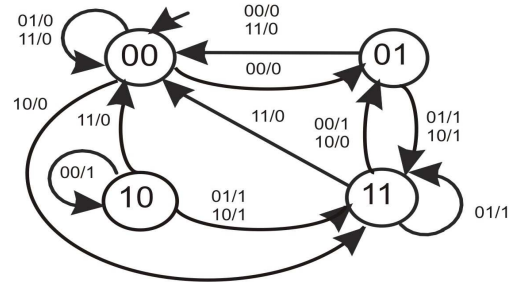


Figure 2. The deterministic submachine B of the specification FSM A.

The test derived by the above method for the implementation FSM B is shown in Figure 3, where all final states are pass-states. Total length of a test is 14 counting a reset. The specification FSM has a submachine with four states whose transition graph is not Euler. Thus, its transition coverage has at least two sequences. Then the total length of each preset test derived in advance, is not less than 18.

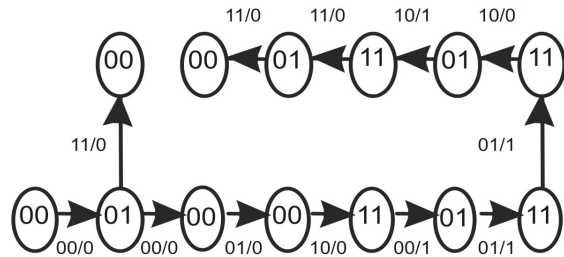


Figure 3. An adaptive test for the implementation FSM B.

5. Conclusions

In the paper, we considered the problem of testing nondeterministic networks as non-deterministic FSMs. We introduced the notions of preset and adaptive tests for

a non-deterministic FSM when testing is used to determine whether the behavior of an implementation at hand, that is a deterministic FSM, is contained in that of the specification. A method for deriving an adaptive test covering all the transitions of a submachine of the specification FSM is proposed. A transition cover of the specification FSM is known to be a high quality test when the specification FSM is deterministic. For example, such a test is known to capture all output faults and almost all single stuck-at- and permutation faults [7]. However, we notice that a transition cover of a non-deterministic specification FSM does not guarantee detection of all output faults even in deterministic submachines of the specification FSM. The reason is an output fault may become latent, as in a non-deterministic FSM, there may exist a transition to another state with the same erroneous output. Additional research is needed to evaluate the fault coverage of transition covering tests for non-deterministic specifications and to elaborate methods for detecting stuck-at-, permutation and other traditional fault models when testing deterministic circuits against a non-deterministic specification network.

References:

1. *Mishchenko A., Brayton R.* A theory of Non-Deterministic Networks // Proc. Intl. Conference on Computer-Aided Design. San Jose, California, 2003. P. 709-717.
2. *Alur R., Courcoubetis C., Yannakakis M.* Distinguishing Tests for Non-deterministic and Probabilistic Machines // Proceedings of the ACM Symposium on Theory of Computing, 1995. P. 363-372.
3. *Tripathy P., Naik K.* Generation of Adaptive Test Cases from Nondeterministic Protocol Models Using UIO Sequences // Proc. of the 5th. IFIP International Workshop on Protocol Test Systems. Montreal, Canada, 1992. P. 166-179.
4. *Petrenko A., Yevtushenko N., Bochmann G. v.* Testing Deterministic Implementations Against their Non-deterministic Specifications // Proceedings of the IFIP Ninth International Workshop on Testing of Communicating Systems. Germany, 1996, P. 125-140.
5. *Hierons R. M.* Using Candidates to test a Deterministic Implementation against a Non-deterministic Finite State Machine // The Computer Journal, 2003. 46, 3. P. 307-318.
6. *Mishchenko A., Brayton R., Jiang R., Villa T., Yevtushenko N.* Efficient Solution of Language Equations Using Partitioned Representations // Proc. Design and Test in Europe. Munich, Germany, 2005. P. 412-417.
7. *Koufareva I.* Using Non-deterministic Finite State Machines for Testing Discrete Event Systems. Ph.D. thesis. Tomsk State University, 2000.

Alexandre Petrenko. Ph.D., Senior Researcher, Team Leader, Computer Research Institute of Montreal, CRIM, Montreal, Canada. Formal Methods, Automata Theory, Testing.

Vetrova Maria Viktorovna. Ph.D., Controllers Design, Digital Systems Optimization, Test Suite Derivation. Tomsk State University, 634050, 36 Lenin Street, (3822) 41-39-84.

Yevtushenko Nina Vladimirovna. Ph.D., Professor of Tomsk State University. Automata Theory, Test Suite Derivation,

Modal Checking, Protocol Testing, Digital Systems Optimization. Tomsk State University, 634050, 36 Lenin Street, (3822) 41-39-84.

СИНТЕЗ УСЛОВНЫХ ТЕСТОВ ДЛЯ НЕДЕТЕРМИНИРОВАННЫХ СЕТЕЙ

ПЕТРЕНКО А.Ф., ВЕТРОВА М.В., ЕВТУШЕНКО Н.В.

Аннотация. В статье предлагается описание теста в виде автомата, определяются условный и безусловный тесты для недетерминированного автомата и иллюстрируется эффективность условных тестов при проверке, содержится ли поведение проверяемого автомата в поведении эталонного недетерминированного автомата.

Петренко Александр Федорович. Д.т.н., руководитель группы, ведущий научный сотрудник CRIM, Монреаль, Канада. Формальные методы. Теория автоматов. Тестирование.

Ветрова Мария Викторовна. К.т.н., доцент каф. ИТИДиС, Томского государственного университета. Синтез контроллеров, оптимизация цифровых схем, синтез проверяющих тестов. Томский государственный университет, 634050, пр. Ленина, 36, (3822) 41-39-84.

Евтушенко Нина Владимировна. Д.т.н., проф., зав.каф. ИТИДиС Томского государственного университета. Теория автоматов, оптимизация цифровых схем, синтез проверяющих тестов, тестирование протоколов вычислительных сетей, верификация цифровых схем. Томский государственный университет, 634050, пр. Ленина, 36, (3822) 41-39-84.