Using an SMT Solver for Checking the Completeness of FSM-Based Tests

Evgenii Vinarskii, Andrey Laputenko, Nina Yevtushenko

Using an SMT solver for checking the completeness of FSM-based tests*

Evgenii Vinarskii[1](0000−0002−7328−0942), Andrey Laputenko[2], and Nina Yevtushenko[3]

[1] Lomonosov Moscow State University, 1 Leninskiye Gory street, 119991 Moscow, Russia
[2] National Research Tomsk State University, 36 Lenin Ave., 634050, Tomsk, Russia
[3] Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25 Alexander Solzhenitsyn street, 109004, Moscow, Russia
vinevg2015@gmail.com, laputenko.av@gmail.com, evtushenko@ispras.ru

Abstract. Deriving tests with guaranteed fault coverage by FSM-based test methods is rather complex for systems with a large number of states. At the same time, formal verification methods allow to effectively process large transition systems; in particular, SMT solvers are widely used to solve analysis problems for finite transition systems. In this paper, we describe the known necessary and sufficient conditions of completeness of test suites derived by FSM-based test methods via the first-order logic formulas and use an SMT solver in order to check them. In addition, we suggest a new sufficient condition for test suite completeness and check the corresponding first-order logic formula via the SMT solver. The results of computer experiments with randomly generated finite state machines confirm the correctness and efficiency of a proposed approach.

Keywords: FSM based testing · SMT solver · Fist order logic formulas

1 Introduction

Finite state machines (FSM) based test derivation is well-known in the software testing of communication protocols and other reactive systems [2–4, 10]. In this case, the behavior of the specification as well as of an implementation under test (IUT) is modeled by a corresponding FSM and testing is performed for determining whether the implementation conforms to the specification. As the number of input sequences is infinite, different limitations are imposed for an implementation under test [2, 7, 10, 11] when deriving finite tests with guaranteed fault coverage. One of the well known limitations is to assume that the number of states of an implementation FSM is not bigger than the number of states of the specification and methods for deriving such tests are developed for various kinds of FSMs, complete and partial, deterministic and non-deterministic as well as for various kinds of conformance relations [3, 9, 11].

Most methods are developed for initialized complete deterministic FSMs where the conformance means the equivalence. Despite the big number of derivatives of such methods the necessary and sufficient conditions for a finite test suite to be a complete test suite are still unknown.

* This work is partly supported by RFBR project No 18-01-00854.

The main idea when deriving a complete test suite, is to establish an isomorphic relationship between state sets of the specification and an implementation under test. For this purpose, so-called state identification sequences of the specification FSM are utilized [2, 3, 10, 11], especially, distinguishing or separating sequences. Intuitively, a distinguishing sequence is a sequence that being applied at two different states provides different output sequences and, thus, for a deterministic machine, different states can be implicitly distinguished without their direct observation. In [2,3,7,10], it is shown that the use of distinguishing sequences allows constructing a complete rather reduced test suite without explicit enumeration of all implementations of the fault domain. However, as an example in [3] demonstrates, the conditions based on distinguishing sequences are not necessary. The idea of the example is to show that two implementation states still can be implicitly distinguished without applying distinguishing sequences at these states. Such distinguishing sequences can be applied at appropriate predecessors or successors of these states. Since this fact is difficult to express as an analytical feature, we turned our attention to the formal verification that now is widely used for checking appropriate properties of different kinds of discrete event and hybrid systems [4].

In this paper, we use the following steps to verify that a test suite TS is or is not complete, i.e., whether requested fault coverage is guaranteed: (1) Given necessary or sufficient conditions for the test suite completeness and a test suite TS, we represent the conditions as a first-order logic formula. If the formula is true, then the TS satisfies the conditions, and if the formula is false then the TS does not satisfy the conditions. (2) The formula satisfiability is checked by the Z3 solver [12]. In addition, we suggest a new sufficient condition for the test suite completeness, describe this condition via a first-order logic formula and check this formula via the Z3 solver. This formula becomes true for the example in [3], i.e., a considered test suite is complete according to this formula.

This paper is organized as follows. Sections 2 and 3 briefly describe finite state machines and properties of a test suite with guaranteed fault coverage as well as include first-order logic formulas for describing some necessary and sufficient conditions for the test suite completeness. Section 4 contains the preliminary experimental results of using Z3 solver for checking the test suite completeness and Section 5 concludes the paper.

2 Deriving test suites using Finite State Machine based methods

The section briefly presents the preliminary concepts that are used in the paper. Most of the definitions are taken from [3, 5].

In this paper, a *Finite State Machine* (FSM) is an initialized complete deterministic machine, i.e., a 6-tuple $\mathcal{M} = \langle S, I, O, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}}, s_0 \rangle$ [5], where S is a finite set of states with the designated initial state s_0, I and O are finite *input* and *output* alphabets, $\delta_{\mathcal{M}} : S \times I \to S$ is the *next state* (or *transition*) function, $\lambda_{\mathcal{M}} : S \times I \to O$ is the *output* function. In the usual way, both functions are extended to input sequences. In this paper, we consider (initially) *connected* FSMs, i.e., we assume that each state is reachable from the initial state via an appropriate input sequence.

A set SC of input sequences is called a *state cover set* of FSM $\mathcal{M}$ if SC has the empty sequence ε and for each state s_i of S, there is an input sequence $\alpha \in SC$ that takes FSM

$\mathcal{M}$ from the initial state to state s_i. If the FSM is connected then such a state cover set always exists. As usual, $seq' = seq.i$ specifies that a sequence seq' is the concatenation of sequences seq and i and $SC.I$ is the set $\{seq.i \in I \mid seq \in SC \& i \in I\}$. A set TC of input sequences is a *transition cover set* if for each state $s \in S$ and each input $i \in I$ there exists sequence $seq \in TC$ such that $seq = seq_1.i$ where $\delta_{\mathcal{M}}(s_0, seq_1) = s$.

Given two complete deterministic FSMs $\mathcal{M}$ and $\mathcal{P}$ over the same input and output alphabets, two states s_i of $\mathcal{M}$ and p_j of $\mathcal{P}$ are *equivalent*, written $s_i \cong p_j$, if for each input sequence $\alpha \in I^*$ it holds that $\lambda_{\mathcal{M}}(s_i, \alpha) = \lambda_{\mathcal{P}}(p_j, \alpha)$. Otherwise, we say that states s_i and p_j are *distinguishable*, written $s_i \ncong p_j$ [5]. FSMs $\mathcal{M}$ and $\mathcal{P}$ are *equivalent*, written $\mathcal{M} \cong \mathcal{P}$, (*distinguishable*, written $\mathcal{M} \ncong \mathcal{P}$) if their initial states are equivalent (distinguishable). Given distinguishable states s_i of $\mathcal{M}$ and p_j of $\mathcal{P}$, an input sequence $\alpha \in I^*$ such that $\lambda_{\mathcal{M}}(s_i, \alpha) \neq \lambda_{\mathcal{P}}(p_j, \alpha)$ is said to *distinguish* states s_i and p_j, written $s_i \ncong_\alpha p_j$. An input sequence that distinguishes the initial states of distinguishable FSMs $\mathcal{M}$ and $\mathcal{P}$ *distinguishes* FSMs $\mathcal{M}$ and $\mathcal{P}$, written $\mathcal{M} \ncong_\alpha \mathcal{P}$. FSM $\mathcal{M}$ is *reduced* if each two different states s, s' of $\mathcal{M}$ are distinguishable. For each initialized complete deterministic FSM, there exists a reduced connected equivalent FSM [5] and in this paper, we assume that all the FSMs are connected and reduced unless the contrary is explicitly stated.

Let $\mathcal{M}$ be the specification FSM with n states, $n > 0$; a test suite TS is a finite set of finite input sequences of the specification FSM $\mathcal{M}$. Given an implementation FSM $\mathcal{P}$ over alphabets I and O, the FSMs $\mathcal{M}$ and $\mathcal{P}$ are TS-equivalent if for each input sequence of TS, the output responses of both machines coincide. A test suite TS is *n-complete* if, for each implementation $\mathcal{P}$ with at most n states that is distinguishable from $\mathcal{M}$, there exists a sequence in TS that distinguishes $\mathcal{M}$ and $\mathcal{P}$. In other words, in this paper, we assume that the number of states of an implementation FSM is not bigger than that of the specification. We now present some conditions when a test suite is or is not n-complete. Let $TS = \{\alpha_1, \ldots, \alpha_k\}$ be a test suite that is checked for the n-completeness where α_1, $\ldots, \alpha_k$ are test sequences. One of the known necessary (Proposition 1) conditions for a test suite to be n-complete is to contain a state / transition cover of the specification FSM. Proposition 2 gives sufficient conditions for a test suite TS to be n-complete [3].

Proposition 1. *Given the complete deterministic reduced connected specification FSM $\mathcal{M}$ with n states and a test suite TS, if TS does not contain the state cover set SC (the transition cover set TC) of $\mathcal{M}$, then TS is not n-complete.*

Proposition 2. *Given the complete deterministic reduced connected specification FSM $\mathcal{M}$ with n states and a state cover set SC of $\mathcal{M}$, let TS be a finite set of finite input sequences of $\mathcal{M}$ that contains the set SC.I. The test suite TS is n-complete if the following conditions hold:*

1. *For each two different states of $\mathcal{M}$ there exist sequences α and β in SC such that TS has sequences $\alpha.\gamma$ and $\beta.\gamma$ where γ is a distinguishing sequence of the states $\delta_M(s_0, \alpha)$ and $\delta_M(s_0, \beta)$.*
2. *For each sequence $\alpha.i$, $\alpha \in SC$, that takes the specification FSM $\mathcal{M}$ from s_0 to state s, TS has sequences $\alpha.i.\gamma$ and $\beta.\gamma$, where $\beta \in SC$, $\delta_{\mathcal{M}}(s_0, \beta) \neq s$, and γ is a distinguishing sequence of states s and $\delta_{\mathcal{M}}(s_0, \beta)$.*

According to Proposition 2, given a state s of $\mathcal{M}$, different state identification sequences can be used when checking incoming transitions to state s, i.e. distinguishing

sequences for the ending state of a transition can be derived on-the-fly using already a constructed part of a test suite. Based on the above proposition the algorithm has been developed for deriving an n-complete test suite, called the *improved* H-method [3]. Note that since an implementation FSM has at most n states, the fulfillment of the first condition implies that an implementation under test has exactly n different states. However, the conditions of Proposition 2 are not necessary conditions for a test suite to be n-complete and this fact is illustrated by the example of the specification FSM in Figure 1 taken from [3].

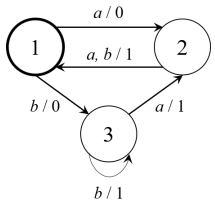

Fig. 1: The example of the specification FSM.

Consider a test suite $TS = \{raaa, rabb, rbaba, rbbab\}$. By direct inspection, one can assure that Proposition 2 does not hold, since states 2 and 3 that are reachable through sequences a and b are not distinguished with suffixes aa, bb and aba, bab. Nevertheless, if an implementation FSM $\mathcal{P}$ reaches the same state under input sequences a and b, then this machine reaches the same state under input sequences aa, ba and ab, bb. The latter means that $\mathcal{P}$ has four different responses to the set $\{a,b\}$ of input sequences and thus, the deterministic machine $\mathcal{P}$ has more than three states. This example shows that states of an IUT reached by two sequences of SC can be implicitly distinguished not only after applying a distinguishing sequence exactly at these states. This intuition leads to a new statement written as Proposition 3.

Proposition 3. *Given the reduced connected specification FSM $\mathcal{M}$ with n states and an FSM $\mathcal{P}$ with at most n states, let TS be a finite set of finite input sequences of $\mathcal{M}$ such that FSMs $\mathcal{M}$ and $\mathcal{P}$ are TS-equivalent. The FSMs $\mathcal{M}$ and $\mathcal{P}$ are equivalent if the following conditions hold:*

1. *For each state s and an input sequence α such that $\delta_{\mathcal{M}}(s_0, \alpha) = s$ there exist input sequences $\beta_1, \ldots, \beta_{n-1}$ such that all the states reached from the initial state p_0 in $\mathcal{P}$ via input sequences $\beta_1, \ldots, \beta_{n-1}$ are pair-wise different.*
2. *Given a sequence $\alpha.i$, $i \in I$, such that $\delta_{\mathcal{M}}(s_0, \alpha) = \delta_{\mathcal{M}}(s_0, \alpha.i) = s$, the states reached from the initial state p_0 in $\mathcal{P}$ via input sequences $\alpha, \beta_1, \ldots, \beta_{n-1}$ are pair-wise different.*
3. *Given a sequence $\alpha.i$, $i \in I$, such that $\delta_{\mathcal{M}}(s_0, \alpha.i) = \delta_{\mathcal{M}}(s_0, \beta_t) = s$ for $t \in \{1, \ldots, n-1\}$, the states reached from the initial state p_0 in $\mathcal{P}$ via input sequences $\alpha.i, \beta_1, \ldots, \beta_{t-1}, \beta_{t+1}, \ldots, \beta_{n-1}$ are pair-wise different.*

Sketch of the Proof. Indeed, if the condition (1) of the above proposition holds then there exists the one-to-one correspondence between sets of states of $\mathcal{M}$ and $\mathcal{P}$. The conditions (2) and (3) of the proposition show that this correspondence is valid for all transitions of $\mathcal{M}$.

Consider again the example in Figure 1. Condition 1 of Proposition 3 holds for the state cover set ε, a, b due to the above comments. The final states of the transitions

under inputs a and b from a state reached after applying input sequence a are different from states reached under input sequences of the set ε, b according to an input sequence a that distinguishes state 1 from states 2 and 3. The final states of the transitions under inputs a and b from a state reached after applying input sequence b are different from states reached under input sequences of the set ε, a according to input sequences ba and ab that distinguish state 1 from states 2 and 3, i.e., Conditions 2 and 3 of Proposition 3 also hold. Therefore, according to Proposition 3, the TS is a 3-complete test suite. We still do not know whether the conditions of Proposition 3 are necessary and sufficient conditions for the test suite completeness. However, this novel definition of determining different states of an implementation under test without using distinguishing sequences directly at these states, opens new directions to find sufficient (necessary and sufficient) conditions for the test suite completeness. In order to check the above properties, we use the Z3 solver which has already illustrated its effectiveness when solving formal verification problems.

3 First-order formulas and complete test suites

In order to verify that appropriate properties hold for a given test suite (Propositions 1-3), we use first-order logic formulas. The notion of a first-order logic formula includes the notions of *predicate symbols*, *functional symbols* and *constants* [6]. Formally, a term $t ::= x|c|f(t,\ldots,t)$ where x ranges over a set of variables, c is a constant and f is a functional symbol. Then a first-order (quantified) logic formula is a formula $\varphi ::= P(t_1, \ldots, t_n) \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\forall x\varphi) \mid (\exists x\varphi)$ where P is a predicate of arity $n \geq 1$ and t_i is a term over functional symbols and variables.

We now describe conditions for the test suite completeness of Propositions 1, 2, 3 via first-order logic formulas. Given FSM $\mathcal{M} = \langle S, I, O, \delta_{\mathcal{M}}, \lambda_{\mathcal{M}}, s_0\rangle$, $|S| = n$, $TS = \{t_1, t_2, \ldots, t_m\}$ is a test suite for FSM $\mathcal{M}$, where t_j, $j \in \{1,2,\ldots,m\}$, is a test sequence, with length $|t_j|$. Let $SS = \{\theta_1, \theta_2, \ldots, \theta_m\}$ ($OS = \{\zeta_1, \zeta_2, \ldots, \zeta_m\}$) be a set of sequences of states (sequences of outputs) which an FSM $\mathcal{M}$ passes when a corresponding test sequence is applied. For $j \in \{1,\ldots,m\}$ and $k \in \{1,\ldots,|t_j|\}$ we define a k-th element in a sequence t_j as t_j^k. Then the formula for Proposition 1 can be as follows: $\forall s \in S$, $i \in I, \exists j \in \{1,\ldots,m\}, \exists k \in \{1,\ldots,|t_j|\}((s = \theta_j^k) \wedge (i = t_j^k))$. This formula is described in python language and verified via Z3 solver. The corresponding source code is available in [8].

The formula for Proposition 2 is more complex (available in [8]) and we first describe the set of predicates for its simplification: (i) Predicate $sub\_seq(seq, TS)$ means that sequence $seq = a_1 \ldots a_k$ is a subsequence of some sequence t_j of TS; predicate $sub\_seq(seq, TS)$ can be expressed via first-order logic as follows: $\exists j \in \{1,\ldots,m\} \exists d \in \{1,\ldots,|t_j| - k\}((a_1 = t_j^d) \wedge \cdots \wedge (a_k = t_j^{d+k}))$. (ii) Predicate $reach(s_0, s, seq)$ means that state s is reached from state s_0 via sequence seq in FSM $\mathcal{M}$. (iii) Predicate $diff(s, s', seq)$ means that sequence seq is a distinguishing sequence for states s and s'. (iv) Predicate $home(s_0, s, seq, TS) = reach(s_0, s, seq) \wedge \forall s' \in S\ (s \neq s' \rightarrow \exists diff\_seq \in I^*$ $(sub\_seq(seq.diff\_seq, TS) \wedge diff(s, s', diff\_seq)))$ means that state s is reached from state s_0 via sequence seq in FSM $\mathcal{M}$ and for each state $s' \in S$, the test suite TS has a sequence which distinguishes states s and s'.

In addition, for each state s, we determine the set of all sequences of TS which take the specification FSM from the initial state to state s. If $S = \{s_0, \ldots, s_{n-1}\}$, then $SC_{s_j} = \{seq \in I^* \mid sub\_seq(seq, TS) \ \wedge \ reach(s_0, s_j, seq)\}$ for each $j \in \{0, \ldots, n-1\}$. The formula for Proposition 2 For checking the conditions of Proposition 2 the following formula can be used: $\exists seq_0 \in SC_{s_0} \ \ldots \ \exists seq_{n-1} \in SC_{s_{n-1}} \ (home(s_0, s_0, seq_0, TS) \wedge \ldots \wedge home(s_0, s_{n-1}, seq_{s_{n-1}}, TS)) \wedge \forall j \in \{0, \ldots, n-1\} \wedge \forall i \in I \wedge \forall k \in \{0, \ldots, n-1\}((s_\ell = \delta_{\mathcal{M}}(s_0, seq_{s_j}.i) \ \wedge \ (\ell \neq k)) \rightarrow (\exists diff\_seq \in I^* \ (sub\_seq(seq_{s_j}.i.diff\_seq, TS) \ \wedge \ diff(s_\ell, s_k, diff\_seq))))$.

The formula for Proposition 3 is rather complex for the analytic description; the source code is available in [8]. The main idea of this formula is as follows. Consider the maximal subset of states $S' \subseteq S$ such that for each state $s \in S'$ there exists a sequence seq for which $home(s_0, s, seq, TS)$ is true and the non-empty subset S'' of states for which this does not hold. Let $s_\alpha \in S'', s_\beta \in S''$ be states reachable in $\mathcal{M}$ via sequences α and β. If the formula is satisfiable then the following holds: if an implementation under test passes the test suite TS and implementation states reachable via sequences α and β coincide, then the implementation has more than n states.

4 Experimental results

In this section, we describe how fast the conditions of Propositions 1 and 2 can be checked when using the Z3 solver. Preliminary experiments were performed with randomly generated FSMs using FSM generator from the FSMTestOnline [1] web-service, which also allows deriving a complete test suite by various FSM-based methods. Experiments showcased that for each generated FSM and complete test suite each formula turned out to be satisfiable, as well as for an incomplete test suite the formula was unsatisfiable. For FSMs with 100 states and 8 inputs, the average time in seconds spent on checking the formula is 219.98 when the formula is satisfiable and 113.24 when the formula is unsatisfiable. The average time for checking the formula according to Proposition 2 was much bigger: for FSMs with five states and two inputs, the average time in seconds spent on checking the formula was 1203.01 when the formula is satisfiable and 18951.48 when the formula is unsatisfiable. For Proposition 3, the formula was checked only for the example of [3] where this formula was satisfiable illustrating that a considered test suite is 3-complete. To conduct experiments, a python script was developed that uses API for SMT solver Z3. The experiments were carried out using virtual machine with a AMD Ryzen 5 2500U CPU @ 2.0 GHz running OS GNU/Linux Ubuntu 16.04 with 7 GB RAM.

5 Conclusion

In this paper, we have described the known necessary or sufficient conditions for an FSM-based test suite to have guaranteed fault coverage via first-order logic formulas and checked these formulas using Z3 solver. In addition, we have suggested a new sufficient condition for the implicit distinguishability of implementation states. In the future, we are going to use this condition for deriving shorter complete test suites as well as for checking the completeness of a given test suite.

References

1. Test generation for finite state machine, http://www.fsmtestonline.ru/
2. Chow, T.S.: Test design modeled by finite-state machines. IEEE Trans. SE **4**(3), 178–187 (1978)
3. Dorofeeva, R., El-Fakih, K., Yevtushenko, N.: An improved conformance testing method. Formal Techniques for Networked and Distributed Systems pp. 204–218 (2005)
4. Fujiwara, S., Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A.: Test selection based on finite state models. IEEE Trans. Software Eng. **17**, 591–603 (06 1991). https://doi.org/10.1109/32.87284
5. Gill, A.: Introduction to the theory of finite-state machines. McGraw-Hill (1962)
6. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge university press (2004)
7. Koufareva, I., Dorofeeva, M.: A novel modification of w-method. Joint Bull. Novosibirsk Comput pp. 69–81 (01 2002)
8. Laputenko, A., Vinarskii, E.: Python scripts for z3 solver. https://github.com/vinevg1996/ictss_2020 (2020)
9. Petrenko, A., Yevtushenko, N., Lebedev, A., Das, A.: Nondeterministic state machines in protocol conformance testing. pp. 363–378 (01 1993)
10. Vasilevskii, M.P.: Failure diagnosis of automata. pp. 98–108. No. 4 (1973)
11. Yevtushenko, N., Petrenko, A.: Failure diagnosis of automata. No. 5 (1990)
12. Yurichev, D.: SAT/SMT by Example (2020)