FSMTest-1.0: a manual for researches

N.Shabaldina, M.Gromov
National Research Tomsk State University
NataliaMailBox@mail.ru, gromov@sibmail.com

Abstract

In this paper we describe software tool «FSMTest-1.0» that was developed by group of authors from the department of Computer science of Tomsk State University. The tool contains implementations of well-known and original test suites generation methods for different models with finite numbers of transitions. The main contribution of our tool is that it derives test suites with the guaranteed fault coverage.

1. Introduction

It is well known that testing is an important stage within the software and hardware development cycle. The main weak place of most testing tools is relying on heuristics, so such tools can't guarantee fault coverage. However, if the system under test is discrete and can be described by the model with finite number of transitions, then we can pretend to derive finite test suites with guaranteed fault covering (w.r.t. the fault model). Our scientific group under the leadership of N.Yevtushenko works in the area of developing methods for generating exhaustive test suites for FSM (Finite State Machine) model, theirs modifications and compositions. The purpose of our investigation is to evaluate proposing methods. In order to do this we create the tool «FSMTest-1.0». In this work we describe our tool and the underlying methods and observe possible applications for researching and educational needs.

2. Related works

Most works related to our work are, of course, different tools for testing. The first tool we would like to mention is TorX [1] and its more recent implementation in Java jTorX [2]. This tool is very close to our tool. TorX uses a Labelled Transition System [3] as underlying model and strong theory for test generation [3]. It even can run test on a system under test whenever a tester provides an adaptor to connect TorX to the SUT, what is not available in our tool. But unfortunately TorX does not have notion of fault model and therefore can guarantee test to be exhaustive only for infinite test. Whenever test is finite (and of course it is every case one uses this tool) nothing about fault coverage is guaranteed. This issue can be overcome for some extent with notion of test purposes [4], but then one needs specify separate test purpose for each fault, which is not a simple job.

Next tool is UniTESK [5]. It is group of tools for different languages and even systems (e.g. MicroTESK [6] is specially designed to test processors) which can not only generate tests but also execute them. All these sub-tools share same concepts: boundary analysis, flow graph coverage and test purposes. Tests provided by UniTESK tools are exhaustive according to the mentioned concepts, but might not be enough when testing real systems.

Among commercial tools we would like to note bunch of tools provided by the company called Conformiq [7]. It is quit user friendly good looking set of tools, providing not only testing facilities, but accompanying functionality as well. Underlying model which is used in these tools is UML state-charts. Mechanisms for test derivation are: boundary analysis, state chart coverage and requirements (test purposes). And again, generated test are exhaustive according to these approaches but might not be enough to find an error.

Of course this list far from being complete but the main weak place of most testing tools is relying on heuristics (most popular — boundary analysis and different coverages) rather than fault model. Such heuristics specify just some points where fault can be, and fault model specifies a whole class of faults. Our tool includes test suite generation methods for different models with finite number of transitions and so we derive exhaustive test suites for conformance and interoperability testing.

3. Preliminaries

In this paper we just mention some notions in a very informal manner.

We are meaning under models with finite numbers of transitions well-known *Finite State Machine* (FSM) model [8] and the modifications of this model. We refresh that FSM is discrete model with the memory (finite number of states) and it produces output sequence in response to the given input sequence. If an FSM produces not more than one output sequence to the given input sequence, then FSM is called *deterministic*. Otherwise, FSM is called *nondeterministic*. Nondeterministic FSM is said to be *observable* if we always can determine the current state of FSM by knowing the initial state, input sequence and observed output sequence.

Timed FSM in this work is FSM with time delays. In this model in addition to the ordinary transitions under inputs there are transitions under time-outs when no input is applying [9, 10].

Testing as a process can be rather different.

An *active testing* is such a process when we apply input sequences, observe output sequences and make a conclusions based on our specification or on the criteria of the faulty-free system. *Passive testing* is such a process when we just can observe sequences (no applying at all).

An active experiment (or testing process) is *preset* if input sequences are known before starting the experiment. An active experiment is *adaptive* if at each step of the experiment the next input is selected based on previously observed outputs.

In order to derive a test suite with the guaranteed fault coverage we need a fault model.

In *conformance testing* a traditional fault model is a triple $<A, \sim, \Re>$, where A is a specification FSM, $\sim$ is a conformance relation, fault domain $\Re$ is the set of all possible (faulty and non-faulty) implementation FSMs with the same input and output alphabets as the specification FSM.

A *test case* is a finite input sequence of the specification FSM. As usual, a *test suite* is a finite set of test cases. We say that an implementation FSM *passes* the test case if an output response is in the set of output responses of the specification FSM. Otherwise, the implementation FSM *fails* the test case. Given a test suite, an implementation *passes* the test suite if it passes each test case. If each faulty FSM from the fault domain fails the test suite, then test suite is said to be *complete*. If each faulty-free FSM from the fault domain passes the test suite, then test suite is said to be *sound*. If test suite is complete and sound then it is said to be *exhaustive*.

In *interoperability* testing [11] the first aim is to assure that two or more implementations can work together without falling into deadlock or livelock (infinite internal or external dialog). In this case the fault model is a pair $<\Re,$ DEC$>$ where $\Re$ is the set of all possible implementation systems while DEC is the criterion of a faulty-free system. And in addition to check livelocks there is also a task of testing in context.

4. FSMTest-1.0: modules for single FSMs

4.1. Test suite generation for deterministic FSMs

Deterministic finite state machine model is a classical model and the only conformance relation that can be checked in black-box testing is a so-called *equivalence* relation. Two FSMs are called equivalent if they have the same behavior, i.e. for a given input sequence they produce the same output sequence. Since 1973 [12] it is known how to derive an exhaustive test suites w.r.t. the fault model $<A, \cong, \Re_m>$, where A is a specification FSM, $\cong$ is an equivalence relation, fault domain $\Re$ is the set of all possible (faulty and non-faulty) FSM implementations with the same input and output alphabets as the specification FSM and the number of state that is not more than known fixed integer number.

Our tool (Figure 1) allows to derive the exhaustive test suites for the deterministic FSMs using the following methods: W [12], Wp, Hsi and H [8]. The input FSM (the specification) should be reduced, connected and deterministic. And the user can set the upper-bound m that is the number of states in FSM implementations. By setting m the user set the fault domain.

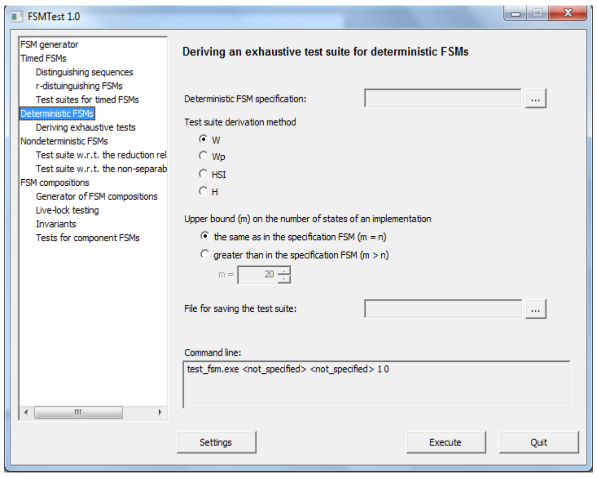
4.2. Test suite generation for nondeterministic FSMs

As we already mentioned, two FSMs are equivalent if they have the same input/output behavior. For nondeterministic FSMs we have more relations that can be checked by black-box testing.

An FSM T is a *reduction* of FSM S ($T \leq S$) if the input/output behavior of T is a subset of that of S [12].

When deriving test suites with respect to the reduction and equivalence relations with the guaranteed fault coverage the so-called *«all weather conditions»* assumption is assumed to be satisfied (in the case of testing a nondeterministic implementation). In the case when this assumption cannot be satisfied the only relation that can be used for the preset test derivation with the guaranteed fault coverage is the separability relation. FSMs T and S are *separable* if there is an input sequence, called a *separating* *sequence*, such that the sets of output responses of these FSMs to the sequence do not intersect, i.e., the sets are disjoint. If such a sequence does not exist then FSMs T and S are non-separable ($T \nsim S$) [12].

Using our tool you can derive exhaustive test suites for the nondeterministic observable FSMs w.r.t. the reduction [13] and to the non-separability relation [12].



Deriving an exhaustive test suite for deterministic FSMs

Deterministic FSM specification:

Test suite derivation method

[✓ W] Wp HSI H

Upper bound (m) on the number of states of an implementation

[✓ the same as in the specification FSM (m = n)] greater than in the specification FSM (m > n)

m = 20

File for saving the test suite:

Command line:

```
test_fsm.exe <not_specified> <not_specified> 1 0
```

Figure 1. Screen of deriving an exhaustive test suites for deterministic FSMs

4.3. Test suite generation for timed FSMs

Our tool allows to derive test suites for deterministic timed FSMs and for observable nondeterministic timed FSMs.

There are two ways for deriving test suites for deterministic timed FSMs in our tool:

1) Test suite for the timed FSM is derived by conversion to classical FSM and then using one of the methods (W, Hsi or H) for test suite derivation [9].

2) Test suite for the timed FSM is derived directly (without conversion to a classical FSM). In this case the fault domain is determined by explicitly enumerating of deterministic FSM implementations. Such a test suite consists of timed input sequences [10]. Each sequence is a *distinguishing sequence* for the specification and one of the FSMs from the given fault domain (i.e. a sequence for which the output sequences of these machines are different).

For nondeterministic observable timed FSMs we consider *r*-distinguishability relation. Two complete (timed) machines can be distinguished by an adaptive experiment if they are *r-distinguishable*, i.e., if they have no common complete reduction. For the compact representation of the adaptive experiment we can use so-called r-*distinguishing* machine [10]. Using our tool you can derive an adaptive test suite for observable timed specification FSM and the fault domain that is determined by explicitly enumerating of observable timed FSM implementations.

4.4. FSM generator

In order to conduct an experiments we add into the tool an FSM generator that can generate one or more FSMs of specified type (classical or timed, deterministic or nondeterministic). If we generate deterministic FSM then we can choose an option «reduced». If we generate nondeterministic FSM then we can choose an option «observable».

In any case generated FSM will be connected, it means it will not have an isolation states.

In addition to FSM type you need to specify FSM's size: the number of inputs, outputs, states.

5. FSMTest-1.0: modules for FSM compositions

As we've mentioned in the introduction, conformance testing is just the first stage of testing. Then we have a stage of interoperability testing. In this stage we are dealing with the communicating systems. And if the work of each system can be described as a finite state machine, then in interoperability testing we are dealing with the composition of FSMs.

We assume, that a system at hand has at most one message to transit, it means that the next external input is submitted to the system only after it has produced an external output to the previous input. And a component machine accepting an input may produce either an internal or an external output. So we consider only binary parallel composition.

5.1. Test suite derivation for live-lock checking

A general fault model for interoperability testing is the pair <$\mathfrak{R}$, *DEC*>, where the fault domain $\mathfrak{R}$ is the set of all possible implementation systems while *DEC* is the criterion of a faulty-free system [11]. For first phase of interoperability testing we define the fault model <$\mathfrak{R}$, livelock- and deadlock-free>, where $\mathfrak{R}$ is the set of compositions of all possible component implementations. We say that *a test suite is complete* w.r.t. the fault model <$\mathfrak{R}$, livelock- and deadlock-free> if the test suite detects each implementation system with livelocks and/or deadlocks, the method is described in [11].

5.2. Deriving forbidden invariants for live-lock passive testing

Our tool allows to derive the set of *forbidden invariants* (input-output sequences that indicates the possibility of livelocks/deadlocks in the system) based on the partial specification of one of the components of the FSM composition. This set of forbidden invariants we can use for the live-lock passive testing.

5.3. Test derivation for component FSMs

Sometimes we haven't a possibility to test an implementation in isolation and we need in this case to test a whole system on the hypothesis that one of the communicating systems is working correct. In our tool one of the components is assumed to be faulty and the conformance relation between specification composition and system under testing is an equivalence relation. The test suite is derived using H-method.

5.4. Generator of component machines for the composition of two FSMs

In order to conduct an experiments we also add into the tool an FSM generator of component machines for the binary parallel composition of two FSMs. Created generator allows to generate partial or complete deterministic FSMs that are describe the behavior of the component machines.

6. Conclusions and acknowledges

In this paper we describe software tool «FSMTest-1.0» that was developed by group of authors from the department of Computer science of Tomsk State University. This tool have been developed for the researching and education use and allows to derive test suites with guaranteed fault coverage. Our University have got the certificate of registration of the tool. The practice value of our investigation is the created tool that is used in the current researches and the educational process in such courses as «Model based testing», «Automata theory» at our department.

This work was supported by the basic part of the State Assignment of the Ministry of Education and Science of the Russian Federation (Project code No. 1975).

7. References

[1] G.J. Tretmans, and H. Brinksma, "TorX: Automated Model-Based Testing", *In First European Conference on Model-Driven Software Engineering*, Germany, 2003, pp. 31-43.

[2] ttps://fmt.ewi.utwente.nl/redmine/projects/jtorx/wiki/

[3] G.J. Tretmans, "Test Generation with Inputs, Outputs and Repetitive Quiescence", *Technical Report TR-CTIT-96-26*, Centre for Telematics and Information Technology University of Twente, Enschede, 1996.

[4] R.G. de Vries and J. Tretmans, "Towards Formal Test Purposes", *Formal Approaches to Testing of Software – FATES'01*, number NS-01-4 in BRICS Notes Series, University of Aarhus, Denmark, 2001, pp. 61–76.

[5] http://www.unitesk.com/

[6] http://forge.ispras.ru/projects/microtesk

[7] https://www.conformiq.com/

[8] R. Dorofeeva, K. El-Fakih, S. Maag, A. Cavalli, and N.Yevtushenko, "FSM-based conformance testing methods: a survey annotated with experimental evaluation", *Information and Software Technology Journal*, Elsevier, 2010 (52), pp. 1286-1297.

[9] M. Zhigulin, N. Yevtushenko, S. Maag, A. Cavalli, "FSM-Based Test Derivation Strategies for Systems with Time-Outs", *In Proc. of the 11th International Conference on Quality Software (QSIC)*, Madrid, 2011, pp. 141-149.

[10] M. Gromov, D. Popov, N. Yevtushenko, "Deriving test suites for timed Finite State Machines", *Proceedings of IEEE East-West Design & Test Symposium*, Kharkov, Ukraine, SPB FL Stepanov V.V., 2008, pp. 339-343.

[11] K. El-Fakih, V. Trenkaev, N. Spitsyna, N. Yevtushenko, "FSM Based Interoperability Testing Methods", *In Proc. of the IFIP 16th International Conference on Testing of Communicating Systems*, U.K., 2004, LNCS 2978, pp. 60–75.

[12] N. Shabaldina, K. El-Fakih, N. Yevtushenko, "Testing Nondeterministic Finite StateMachin es with Respect to the Separability Relation", *Springer*, Berlin, 2007, pp. 138-154.

[13] A. Petrenko, N. Yevtushenko, "Conformance tests as checking experiments for partial nondeterministic FSM", *Proc. 5th International Workshop on Formal Approaches to Testing of Software*, 2005.