Design by contract approach to test generation for EFSMs using GA

10th Annual International Software Testing Conference 2010

Andrey Zakonov, Anatoly Shalyto
Faculty of Information Technologies and Programming
St. Petersburg State University of Information Technologies, Mechanics and Optics
Saint-Petersburg, Russia
e-mail: andrew.zakonov@gmail.com, shalyto@mail.ifmo.ru

Abstract

Design by contract approach prescribes that developer should define formal and verifiable interface specifications for software components and makes it possible to automate process of software testing. We propose to adapt this approach for Extended Finite State Machines (EFSMs), which are often used in model-based development and for modeling VHDL specifications. This paper proposes an approach for automated test generation for EFSM models. Design by contract approach is applied to formalize specification requirements. Genetic algorithm is proposed to find set of values that triggers given path in the EFSM and reveals inconsistensies with the specification.

1. Introduction

Extended Finite State Machines (EFSMs) are used in different areas to describe behaviour of the systems with complex logic: embedded systems, modeling VHDL specification, protocol descriptions, model-based development. In an EFSM the transition can be expressed by an "if statement" consisting of a set of trigger conditions. If trigger conditions are all satisfied, the transition is fired, bringing the machine from the current state to the next state and performing the specified data operations [1].

It's highly important to check conformance of the system's implementation against its specification. Model Checking [2] is commonly used to check conformance of the model against given requirement. However, verification techniques don't allow checking the system in whole, as model usually interacts with some environment, which is not suitable for Model Checking.

In this paper we propose to use testing to check the EFSM-based system in whole. Software testing is normally a labor intensive and very expensive task. It accounts for about half of a typical software project life cycle [2]. This means that straightforward approach to testing, such as manual testing, is not the best option. Recently there has been much interest in automated test data generation [4]. Even though testing cannot guarantee the correctness of a program, large number of tests does contribute significantly to the identification and reduction of faults, improving the likelihood that the software implementation will succeed. Therefore this paper includes description of an approach for testing EFSM-based programs and a way to automate this process using genetic algorithms. Design by contract approach [3] are used to extend model with specification requirements and we demonstrate how genetic algorithms could be applied to automate generation of tests that reveal faults in the system in whole.

Overall, this paper addresses number of problems:

- propose an approach for testing EFSM-based programs;
- automate test creation by providing a tool, which finds suitable sequence of events and set of external variables for a given transition path and generates test code;
- automate validation of specification requirements, included in the EFSM, while executing tests;
- attempt to generate tests that lead to violation of specification requirements and so reveal faults in implementation.

The rest of the paper is organized as follows. Section 2 describes how design by contract approach can be applied to EFSMs. Section 3 gives details on proposed approach for testing EFSM-based programs. Problem of test generation is described at Section 4. Section 5 describes genetic algorithm applied to find external variables' values. Section 6 tells about proof-of-concept tool being developed and preliminary results; Section 7 concludes.

2. Design by contract approach for EFSMs

Design by contract approach prescribes that developer should define formal and verifiable interface specifications for software components, which are expressed by preconditions, postconditions and invariants. We propose to adapt this approach to EFSM models, by writing requirements for the variables used in EFSM in guard conditions and action sections:

- Invariants are added to the states of the model and are used to describe specification of the system for the selected state of the model;
- Pre- and postconditions are added to the transitions, similar to function calls, and define requirements on values that model recieves from its environment and also requirements for model's output to the environment.

Having specification requirements included into the program makes it possible to automate

checking of such issues as incorrect input to the model or incorrect implementation of the model itself. We propose to use Java Modelling Language (JML) to include specification contracts into the model.

3. Approach for testing EFSMs

Even though testing cannot guarantee the correctness of a program, large number of tests does contribute significantly to the identification and reduction of faults, improving the likelihood that the software implementation will succeed. Software testing is normally a labor-intensive activity. It accounts for about half of a typical software project life cycle [4]. This means that straightforward approach to testing is not the best option and it is highly desirable to automate this process.

We propose to use scenario testing approach: sequence of transitions (transition path in the EFSM) is considered to be a convenient way to describe a test scenario. Such representation of the test could be easily derived from a natural language description of a user story. Moreover proposed approach doesn't require writing any program code in order to create tests, which makes process of testing less time consuming. Executable code of the tests that check the selected transition path can be generated automatically.

Due to specification contracts included into the model system contains the instruments for its verification. Evalutation of tests can be also automated by using a JML Runtime Assertion Checker tool [5].

Also there is number of researches available [6] that addresses the problem of finding transition paths in EFSM to achieve selected coverage criteria (e.g. state or transition coverage in the EFSM). Such techniques can be successfully used together with manual test paths selection and, combined with the approach presented in this paper, could help to automate producing of valuable test suites.

4. Problem of test generation

Test scenario is described as a sequence of transitions in the model. An EFSM reacts to the events and perform transitions depending on the transition guards. Therefore to make the EFSM to traverse the given path one would need to:

1. Emulate correct sequence of events;
2. Provide such values of the EFSM variables, that all the transition guards would be fulfilled.

Obtaining sequence of events for the path is straightforward. However there is no easy way to guess values of the EFSM variables to fulfill all the transition guards on the given path. We propose to apply genetic algorithms to find suitable variable values.

Traversing selected path in the EFSM model makes it possible to automate process of test generation but it gives no guarantee that faults in the system would be revealed. Genetic algorithm proposed in the paper looks for the values of the EFSM model that aim two targets:

1. To fulfill all the conditions on the given path;
2. To violate specification requirement that is included in the model in the form of JML contract.

Obtaining such values makes it possible to generate an executable test that will reveal an inconsistency between implementation and system specification.

5. Genetic algorithm to obtain variable values

5.1 Optimization problem

Set of external variables can be represented as a vector of values $<x_1, x_2, \ldots, x_n>$, where x_i is an external variable, and n is number of external variables required for this transition path. Fitness function takes this vector as an argument and returns fitness value for an external variables set. The smaller fitness value is the better the proposed vector suits the given transition path. From this point of view task can be considered as a minimization problem, where we look for the set of variables with the minimum fitness value.

5.2 Candidate encoding

Candidate is a vector of values, as defined above. We use one-point crossover operator, which operates by choosing a random position in the vector, and then new candidate is composed of first candidate's sub-vector before that position and second candidate's sub-vector after that position.

Mutation operator replaces random position of the vector to a new random value.

5.3 Fitness function

Fitness function aims to provide metric for candidates, which tells how good is this candidate for a specified task. In our case task is to execute given sequence of transitions in the automaton. There is no unambiguous answer for the question of what fitness function to choose.

Approaches for testing of structured programs propose to use such criteria as branch distance [10] for fitness calculation. A branch distance is a measure of how close a particular candidate is to executing the target branch that is missed e.g., $|A-B|$ is the branch distance for the predicate $(A > B)$. The lower $|A-B|$ is the closer is A to B and the closer the candidate is to filfulling the condition. For the filfulled condition branch distance equals zero. There are researches [6] that show effectiveness of described approach for structured programs testing.

In [6] branch distance based approach is used to find input test data that can cause a feasible path in an EFSM model to be traversed. In our research we extend this approach to apply it to EFSM-based systems. As it was described above, we must take into account not a standalone EFSM, but an EFSM-based program enriched with system's and control objects' specification. Moreover we aim to find set of variables not only to execute selected path, but to filfull control objects' requirements and ideally to reveal inadequacy of implementation and specification.

To obtain variable values to execute given path there are two types of conditions that should be taken into the account:

- guard conditions on the transitions of the EFSM;
- specification requirements of conrolled objects that provide external variables.

These conditions are obligatory to be filfulled. Candidate that fail any of these conditions are not appropriate for test generation, as specification doesn't require system to support such inputs. So in this case fitness function should estimate how close this particular candidate was to filfulling failed conditions.

To give an accurate estimation we examine each state and transition between states on the given path separately. Every transition has the event, which enables it and may have a guard condition and an action section. In the current implementation external variables are introduced in transitions' action sections.

Control objects' specification can be included in transition contracts: preconditions and postconditions. Precoditions verify, that EFSM model is in correct state to use controlled object; postconditions verify, that external variable value retrieved from the controlled object meets specification requirements.

From this point of view execution of each transition in the path is divided into three small steps:

- receive event, find transition and check guards;
- check preconditions and execute the transition;
- check transition postconditions.

Each of these steps contains coditions that can be failed. Therefore for each of these steps we calculate branch distance. Fitness value for a single transition is calculated as sum of steps' branch distances.

It's important to realize that transitions are executed sequentially. This means that to achieve second transition candidate must successfully complete first one. Therefore transitions in the beginning of the path are somehow more important then transitions in the end. This fact should be taken into the account in the fitness function calculcation. For more accurate fitness value we consider step approach level. In such approach each step is assigned a weight value, which depends on the step's position in the path. Last step weight is the smallest, first step weight is the greatest. Overall fitness of the candidate for the given path is calculated as sum of steps' fitness multiplied by their weights.

5.4 Specification requirements in fitness function

Fitness function described above is aimed to find set of variables that would make possible given path execution. More desirable is to find a candidate, which reveals an inconsistesy between implementation and specification. For this purpose we need take into consideration specification

requirements of the system represented as contracts that must be filfulled during the execution. We aim to fail any of these conditions, while guards and controlled objects' requirements are filfulled.

Such task requires iterated approach, as we need to select specific transition, which conditions we want to fail. For example, if we want any of the conditions on the second transition to be failed, we need all the conditions of the first transition to be filfulled, because there may be a dependency between these conditions. For different transitions selected as target fitness function is computed differently. Generally, if k^{th} transition is a target to fail some condition, then all conditions of the transitions with indexes less then k must be fulfilled.

In attempt to fail some conditions we use branch distance turned inside out. If condition is failed then value is zero. The closer the candidate is to failing the condition the lower the value. This reversed branch distance value is included in path fitness value calculation, similar to common step fitness, described above.

We aim to reveal faults at any transition so we iterate through the given path. At the first step we consider transition path of one transition, the first one. We perform fixed number of attempts to reveal a fault. If any found, test is generated. After fixed number of attempts we move to the next step: consider path of two transitions. We go on like this till we reach the whole given path length. Finally, after all the iterations are done, for all revealed faults test code is generated, which can be executed separately and used for debugging and bug fixing.

6. Case study

In this paper we present a case study and a proof-of-concept tool that is being developed during the research. Version of the tool used for the case study contained number of limitations: only integer variable types are supported and separate tools are used for variable values search and executable test code generation.

Example of specification for ATM-like machine is being examined and a model-based program is developed during the case study in order to illustrate our approach. Sample specification of an ATM machine:

- System must perform withdrawal operations from the specified account on user requests;
- Initial amount of money on the account is being retrieved from the bank and must be a positive number, less or equal to 1000000;
- Each time a user inputs amount of money on the keyboard a transaction must be initiated. Amount must be greater then 1000 and less then 5000;
- A transaction must be successfully completed only if after the transaction there would be a positive amount of money left on the account.
- While no error occurs user can make withdrawals unlimited number of times.

An EFSM model that implements desired behaviour and contains specification requirements as JML contracts is presented on Fig. 1.

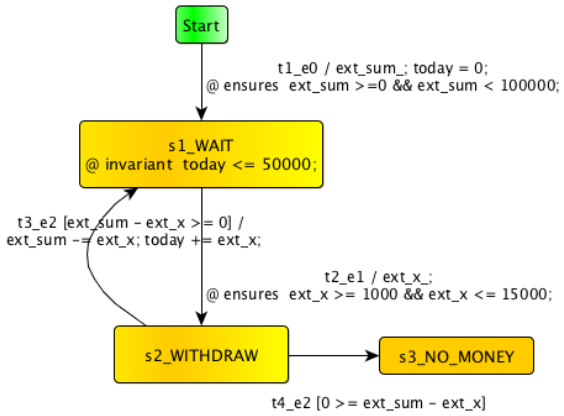

Fig. 1. EFSM model of the ATM machine

Model contains number of variables that come from the environment:

- Initial amount on the account;
- User inputs to withdraw.

Genetic algorithm would be apllied to find values of these variables to suit the given scenario. We

considered test scenarios of different complexity to evaluate our approach. Scenario examples:

- User withdraws 50 times and on 51th attempt transaction fails, as not enough money on the account;
- User withdraws 3 times and on 4th attempt transaction fails, as not enough money on the account.

We describe scenario as a sequence of transitions of the model: t1, t2, t3, t2, t3, t2, t3, t2, t4. Transition sequence and file with the EFSM model are given as an input to the proof-of-concept tool. Depending on the number of unknown variables used in the desired path search of the variable values by genetic algorithm takes from 10 seconds to 20 minutes for described test scenarious. When the values are obtained an executable test code is generated and evaluated automatically. If any conracts are violated during the execution then an exeption is generated so user can examine the discovered implementation's inconsistency with the specification.

7. Conclusion

Simultaneously with Model Checking testing is a useful technique that allows checking conformance of implementation and specification while developing EFSM models. For effective testing it is important to automate test generation process, as manual test creation is labor intensive and expensive task. In this paper we proposed an approach for testing of EFSM models and a proof-of-concept tool demonstrating benefits of described approach. Design contracts are used to create models containing specification requirements. Genetic algorithm is used to automate the test generation process.

We plan to provide an IDE plug-in for JetBrains MPS (Meta Programming System) [8], which has the StateMachine extension for model-based development [9]. Seamless integration of test creation into the development process would allow detecting possible implementation faults and design flaws at all development stages.

8. References

[1] Cheng, K-T; Krishnakumar, A.S. (1993). "Automatic Functional Test Generation Using The Extended Finite State Machine Model". International Design Automation Conference (DAC). ACM. pp. 86–91.

[2] E. M. Clarke, Jr. O. Grumberg and D. A. Peled, "Model Checking", MIT Press, 1999

[3] B. Meyer, "Applying design by contract," Computer, 25(10), pp. 40–51, Oct. 1992

[4] G. Myers, The Art of Software Testing, 2 ed: John Wiley & Son. Inc, 2004.

[5] Cheon Y., Leavens G. T. A Runtime Assertion Checker for the Java Modeling Language (JML). In Hamid R. Arabnia and Youngsong Mun (eds.), Proceedings of the International Conference on Software Engineering Research and Practice (SERP '02), Las Vegas, Nevada, USA, pages 322-328. CSREA Press, June 2002

[6] Kalaji, A.S., R.M. Hierons, and S. Swift. "Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)," in Software Testing, Verification, and Validation (ICST), 2009 2nd International IEEE Conference on. 2009. Denver, Colorado - USA: IEEE.

[7] Wegener, J., A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," Information and Software Technology, 2001. 43(14): p. 841-854.

[8] JetBrains Meta Programming System User's Guide. http://www.jetbrains.net/confluence/display/MPS/MPS+User%27s+Guide.

[9] Gurov V., Mazin M., Narvsky A., Shalyto A. UniMod: Method and Tool for Development of Reactive Object-Oriented Programs with Explicit States Emphasis, Proceedings of St. Petersburg IEEE Chapters. Year 2005. International Conference "110 Anniversary of Radio Invention", SPb ETU "LETI", 2005, vol. 2, pp. 106-110.

[10] Tracey, N., J. Clark, K. Mander, and J. McDermid. “An automated framework for structural test-data generation,” in Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on. 1998.