N-Switch and All-Path Test Coverage Criterion for Extended Finite State Machine

Tomohiko Takagi
Department of Engineering and Design, Faculty of Engineering and Design, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Koichiro Sakata
Division of Reliability-based Information Systems Engineering, Graduate School of Engineering, Kagawa University
2217-20 Hayashi-cho, Takamatsu-shi, Kagawa 761-0396, Japan

Kouichi Akiyama
Japan WillTech Solution Co., Ltd.
14F Nihonbashi-Muromachi Mitsui Tower, 2-1, Nihonbashi-Muromachi 3-chome, Chuo-ku, Tokyo 103-0022, Japan
E-mail: takagi@eng.kagawa-u.ac.jp, s20g465@stu.kagawa-u.ac.jp

Abstract

This paper shows a new test coverage criterion for extended finite state machine to comprehensively test the combination of state transitions and accompanying actions in software. Our criterion requires that (i) test cases cover all the successive state transition sequences of specified length, and also (ii) the test cases cover all the paths on control flow graphs of actions that accompany each of the successive state transition sequences. (i) and (ii) are the characteristics of N-switch and all-path test coverage criterion, respectively. Its definition, example and effectiveness are discussed in this paper.

Keywords: model-based testing, finite state machine, test case, test coverage criterion

1. Introduction

One of the well-used software modeling languages in model-based testing[1] is EFSM (Extended Finite State Machine)[2,3]. It enables engineers to define the expected behavior of software from the aspect of not only state transitions but also data processing. The part of state transitions, which corresponds to FSM, is drawn in the form of simple table or directed graph. On the other hand, the part of data processing, which corresponds to the actions on state transitions, is written in natural languages or TBFML (Text-Based Formal Modeling Language). Test cases are usually created from EFSM models so as to satisfy a test coverage criterion called *N*-switch[4], that is, cover all the successive state transition sequences of length *N*+1 ($N \geq 0$). However, *N*-switch is originally designed for FSM, and actions are not taken into account in it. The behavior of software is determined by the combination of state transitions and actions in EFSM models, and thus it should be comprehensively tested.

To address this problem, we propose a new test coverage criterion for EFSM, named *N*-SAP$_L$ (*N*-Switch for state transitions and All-Path with loop frequencies *L* for actions) in this paper. It is a combination of *N*-switch and AP (All-Path test coverage criterion). AP is usually used in structural testing[5] in order to cover all the paths on control flow graphs of programs under test. *N*-SAP$_L$ requires that (i) test cases cover all the successive state transition sequences of length *N*+1, and also (ii) the test cases cover all the paths on control flow graphs of actions that accompany each of the successive state transition sequences. (i) and (ii) are the characteristics of *N*-switch and AP, respectively. Note that *L* expresses a set of loop frequencies to be taken into account in the control flow graphs of actions. When there are any loop structures in the actions, *L* makes the number of paths that should be covered finite. To avoid the ambiguity and allow the code coverage analysis for AP, all the actions in our EFSM models are written only in TBFML, such as VDM++[6,7].

This paper is organized as follows. Section 2 shows the overview of traditional model-based testing using EFSM. In section 3, we describe the details of N-SAP$_L$ and a simple example. Section 4 gives discussion based on preliminary experiments. Finally, section 5 shows conclusion and our future work.

2. Traditional Model-Based Testing Using EFSM

Fig. 1 shows an example of an abstracted EFSM model. An EFSM model in this study mainly consists of states, events, and actions. It illustrates possible state transitions with data processing in software, and each of them is identified by a tuple in the form of <*from-state*, *event*, *action*, *to-state*>. For example, the state transition β in Fig. 1 is identified by <s_2, e_1, a_2, s_4>. Each state transition can optionally have event parameters and a guard. Each of actions consists of codes written in TBFML, and can define and refer variables that characterize the behavior of software. Some actions may be shared by different state transitions. For example, a_2 is shared by β and δ in Fig. 1.

Test cases are created from an EFSM model in the form of sequences of successive state transitions and expected values of variables. The quality of the test cases is usually evaluated by N-switch, and they are created so as to satisfy it. Software implemented based on the EFSM model can be closely tested by a larger value of N. For example, a technique using orthogonal arrays is used to reduce testing effort[8].

In general, coverage of given test cases is calculated by $|M'|/|M|$. M expresses a set of measuring objects that should be covered, and is determined by a selected test coverage criterion. For example, M for 1-switch is a set of all the successive state transition sequences of length 2. In N-switch, a larger value of N results in M that consists of a larger number of measuring objects. On the other hand, M' expresses a set of measuring objects that have actually been covered by the given test cases, and satisfies $M' \subseteq M$. When M' is equal to M, the test coverage criterion is satisfied. Therefore, a larger value of N results in a larger number of test cases to satisfy N-switch.

3. N-SAP$_L$ Test Coverage Criterion for EFSM

As is discussed above, test coverage criteria are used to determine measuring objects. In this section, we propose a procedure to systematically get measuring objects for N-SAP$_L$ from a given EFSM model. The procedure

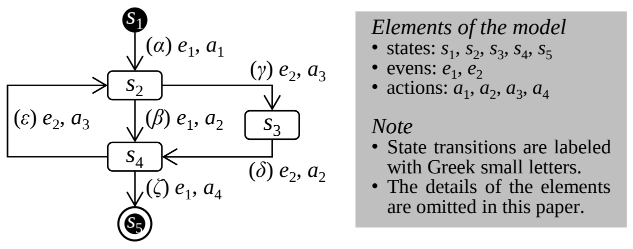
s_1
(α) e_1, a_1
(γ) e_2, a_3
s_2
(ε) e_2, a_3
(β) e_1, a_2
s_3
s_4
(δ) e_2, a_2
(ζ) e_1, a_4
s_5
Elements of the model
• states: s_1, s_2, s_3, s_4, s_5
• evens: e_1, e_2
• actions: a_1, a_2, a_3, a_4
Note
• State transitions are labeled with Greek small letters.
• The details of the elements are omitted in this paper.

Fig. 1. Example of an abstracted EFSM model.

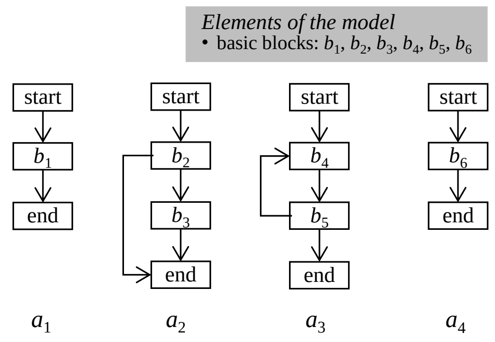

Fig. 2. Abstracted control flow graphs of actions in Fig. 1.

consists of the following five steps. At least Step 2, 3, 4 and a part of Step 5 can be automated.

Step 1. N and L are determined by test engineers according to a given test plan. The former is for the length of successive state transition sequences to be covered, and the latter is the set of loop frequencies to be taken into account in actions.

Step 2. A set of all the measuring objects for N-switch is generated by using a common graph search algorithm. The set is hereinafter referred to as MS_N. The measuring objects as the elements of MS_N are successive state transition sequences of length N+1, and each of them is hereinafter referred to as sts_x ($1 \le x \le |MS_N|$). For example, MS_1 for Fig. 1 is {$\alpha\beta$, $\alpha\gamma$, $\beta\varepsilon$, $\beta\zeta$, $\gamma\delta$, $\delta\varepsilon$, $\delta\zeta$, $\varepsilon\beta$, $\varepsilon\gamma$}.

Additionally, MS_N should include special measuring objects that satisfy all of the following conditions:

- Their length are less than N+1.
- They start with the initial state, and end with a final state.

For example, MS_3 for Fig. 1 should include $\alpha\beta\zeta$. The special measuring objects ensure that the satisfaction of N-SAP$_L$ results in the satisfaction of N'-SAP$_L$ ($N' < N$).

Step 3. A set of all the paths in each action is derived by using common path analysis based on control flow graphs. When there are any loop structures in actions, only the loops of frequencies specified by L are taken into account. Fig. 2 shows abstracted control flow graphs of the actions in Fig. 1. For example, when L is {0, 1}, the sets for a_1, a_2, a_3 and a_4 are $\{b_1\}$, $\{b_2, b_2b_3\}$, $\{b_4b_5, b_4b_5b_4b_5\}$ and $\{b_6\}$, respectively. Note that in a_3 there are infinite paths but L makes the number of paths that should be tested finite.

Step 4. A set of all the paths in the actions that accompany sts_x, which is hereinafter referred to as $PSTS_x$, is derived by the following equation:

$$PSTS_x = \prod_{y=1}^{\#sts_x} PST_{x,y} \tag{1}$$

$\#sts_x$ means the length of sts_x, and $PST_{x,y}$ expresses a set of all the paths in the action that accompanies the yth state transition in sts_x. For example, when L is {0, 1}, *PSTS* for $\beta\varepsilon$ in Fig. 1 is $\{<b_2, b_4b_5>, <b_2, b_4b_5b_4b_5>, <b_2b_3, b_4b_5>, <b_2b_3, b_4b_5b_4b_5>\}$.

Step 5. A set of all the measuring objects for N-SAP$_L$, which is hereinafter referred to as $MSAP_{N,L}$, is derived by the following equation:

$$MSAP_{N,L} = \bigcup_{x=1}^{|MS_N|} F(\{sts_x\} \times PSTS_x) \tag{2}$$

$F(S)$ is a feasibility evaluation function, and it removes infeasible elements from a given set S. The feasibility in EFSM models can be evaluated by using metaheuristics[3], symbolic execution, and manpower. For example, $|MSAP_{1,\{0,1\}}|$ for Fig. 1 is 28 as shown in Table 1, if there are no infeasible ones. When given test cases cover 14 elements in the $MSAP_{1,\{0,1\}}$, 1-SAP$_{\{0,1\}}$ test coverage achieves 50% (14/28). Note that the same paths often appear in different state transition sequences, but they are distinguished in $MSAP_{N,L}$. For example, $<\beta\zeta, <b_2b_3, b_6>>$ and $<\delta\zeta, <b_2b_3, b_6>>$ are strictly distinguished. Of course, the same state transition sequences with different paths are also distinguished in $MSAP_{N,L}$. For example, $<\alpha\beta, <b_1, b_2>>$ and $<\alpha\beta, <b_1, b_2b_3>>$ are strictly distinguished. Additionally, the same basic blocks that appear in different state transitions are also distinguished. For example, $<\varepsilon\gamma, <b_4b_5, b_4b_5b_4b_5>>$ and $<\varepsilon\gamma, <b_4b_5b_4b_5, b_4b_5>>$ are strictly distinguished.

4. Discussion

We developed a tool for preliminary experiments, and tried to automatically generate the measuring objects of N-SAP$_L$ from Fig. 1. The overview of its result is shown in Table 2. Note that we added special measuring objects in Step 2, and assumed that there are no infeasible ones in Step 5. This result indicates that the number of measuring objects rapidly becomes larger when a larger number and set are given to N and L respectively. For example, the following ways can be adopted to address this problem:

- Remove redundant measuring objects according to the results of fault-proneness prediction using bug-fixing record[9], identifiers in source code[10], and so on.
- Limit L to typical loop frequencies that should be tested, such as 0, 1, and multiple times.
- Automate the generation and execution of test cases that satisfy N-SAP$_L$.

Test engineers will need a tool especially to generate the test cases from their EFSM models. The difficulty of constructing the tool is in (a) solving feasibility problems on state transitions and actions, and (b) minimizing the number of the test cases. One of effective techniques for addressing (a) and (b) is metaheuristics[3]. For example, the following algorithm will enable to generate test cases that satisfy N-SAP$_L$:

Step 1. Search (execute) a given EFSM model randomly to find test case candidates. This step ensures the feasibility of test cases.

Step 2. Evaluate each candidate by using a fitness function. Good candidates include many new measuring objects without redundancy. If there are not good ones, modify the search policy as necessary, and then go to Step 1.

Step 3. Select the best candidates as a subset of final test cases. If the set of final test cases does not cover

Table 1. Measuring objects of 1-SAP$_{\{0,1\}}$ in Fig. 1.

state trans. seq.	**all paths in actions that accompany the state trans. seq.**
$\alpha\beta$	$<b_1, b_2>, <b_1, b_2b_3>$
$\alpha\gamma$	$<b_1, b_4b_5>, <b_1, b_4b_5b_4b_5>$
$\beta\varepsilon$	$<b_2, b_4b_5>, <b_2, b_4b_5b_4b_5>, <b_2b_3, b_4b_5>, <b_2b_3, b_4b_5b_4b_5>$
$\beta\zeta$	$<b_2, b_6>, <b_2b_3, b_6>$
$\gamma\delta$	$<b_4b_5, b_2>, <b_4b_5b_4b_5, b_2>, <b_4b_5, b_2b_3>, <b_4b_5b_4b_5, b_2b_3>$
$\delta\varepsilon$	$<b_2, b_4b_5>, <b_2, b_4b_5b_4b_5>, <b_2b_3, b_4b_5>, <b_2b_3, b_4b_5b_4b_5>$
$\delta\zeta$	$<b_2, b_6>, <b_2b_3, b_6>$
$\varepsilon\beta$	$<b_4b_5, b_2>, <b_4b_5b_4b_5, b_2>, <b_4b_5, b_2b_3>, <b_4b_5b_4b_5, b_2b_3>$
$\varepsilon\gamma$	$<b_4b_5, b_4b_5>, <b_4b_5, b_4b_5b_4b_5>, <b_4b_5b_4b_5, b_4b_5>, <b_4b_5b_4b_5, b_4b_5b_4b_5>$

Table 2. Number of measuring objects for N-SAP$_L$ in Fig. 1.

N	***L***			
	{0}	**{0, 1}**	**{0, 1, 2}**	**{0, 1, 2, 3}**
0	8	10	12	14
1	16	28	42	58
2	28	74	140	226
3	52	198	476	922
4	92	526	1604	3674
5	164	1390	5384	14618

enough measuring objects, modify the search policy, and then go to Step 1.

5. Conclusion and Future Work

In this paper, we have proposed N-SAP$_L$ test coverage criterion for EFSM to comprehensively test the combination of state transitions and accompanying actions in software. A characteristic of this study is to blend a functional state-based testing technique and a structural path testing technique. The latter plays an important role in testing of the actions.

The satisfaction of N-SAP$_L$ results in the satisfaction of N'-SAP$_{L'}$, if $N'<N \wedge L' \subset L$ is satisfied. When test engineers give a larger number to N and a larger set to L in order to achieve higher software reliability, they will need to create a larger number of test cases to cover the measuring objects of N-SAP$_L$. It will not be so easy for test engineers to manually calculate N-SAP$_L$ coverage and create test cases that satisfy N-SAP$_L$. Therefore we will plan to develop a prototype tool to support such tasks.

One of challenges in future work is to develop a technique to automatically find infeasible measuring objects in EFSM models, which will make it possible to get precise N-SAP$_L$ coverage.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP17K00103.

References

1. M. Utting, A. Pretschner, B. Legeard, "A taxonomy of model-based testing approaches", Journal of Software Testing, Verification and Reliability, Vol.22, No.5, pp.297-312, Aug. 2012.
2. Object Management Group, "Unified Modeling Language", https://www.uml.org/.
3. A. Kalaji, R.M. Hierons, S. Swift, "Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)", Proc. of International Conference on Software Testing Verification and Validation, IEEE, Denver, CO, United States, pp.230-239, April 2009.
4. T.S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering, Vol.SE-4, No.3, pp.178-187, May 1978.
5. B. Beizer, "Software Testing Techniques", 2nd edition, Van Nostrand Reinhold, New York, NY, United States, 1990.
6. J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, "Validated Designs for Object-Oriented Systems", Springer-Verlag London, 2005.
7. T. Takagi, K. Sakata, "Test-First for Abstracted Behavior of Software Using Extended Finite State Machine", Proc. of International Symposium on Software Reliability Engineering, IEEE, Coimbra, Portugal, pp.159-160, Oct. 2020.
8. K. Akiyama, "A Problem of N-Switch Coverage Testing and Its Solution", Proc. of Software Symposium, Gifu, Japan, 7 pages, July 2013 (in Japanese).
9. J.A. Whittaker, J. Arbon, J. Carollo, "How Google Tests Software", Addison-Wesley Professional, Boston, MA, United States, 2012.
10. O. Mizuno, N. Kawashima, K. Kawamoto, "Fault-Prone Module Prediction Approaches Using Identifiers in Source Code", International Journal of Software Innovation, Vol.3, Issue 1, pp.36-49, 2015.

Authors Introduction

Dr. Tomohiko Takagi

He received the B.S., M.S. and Ph.D. degrees from Kagawa University in 2002, 2004 and 2007, respectively. He became an assistant professor in 2008, and a lecturer in 2013 in the Faculty of Engineering at Kagawa University. Since 2018 he has been an associate professor in the Faculty of Engineering and Design at Kagawa University. His research interests are in software engineering, particularly software testing.

Mr. Koichiro Sakata

He received the B.S. degree from Kagawa University in 2020. He is a master's student in the Graduate School of Engineering at Kagawa University. His research interests are in software engineering, particularly software testing.

Dr. Kouichi Akiyama

He received the Ph.D. degree from Kagawa University in 2013. Since 2021 he has been an internal appraiser of CMMI. His research interests are in software engineering, particularly software testing.