2012 IEEE Sixth International Symposium on Theoretical Aspects of Software Engineering

A Path-oriented Approach to Generating Executable Test Sequences for Extended Finite State Machines*

Tianyong Wu[1,3], Jun Yan[1] and Jian Zhang[2]
1. *Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences*
2. *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences*
3. *Graduate University of Chinese Academy of Sciences*
Email: {wutianyong11, yanjun}@otcaix.iscas.ac.cn, zj@ios.ac.cn

***Abstract*—The Extended Finite State Machine (EFSM) is a commonly used model for specifying software systems. A test sequence for an EFSM is a sequence composed of values of input variables, which can make the EFSM "execute" along a complete path from entry to exit. Traditional test sequence generation methods for EFSM almost imitate those FSM-based approaches and focus on states identification. Most of them impose significant restrictions on the EFSM. This paper proposes a path-oriented approach to generating test cases for EFSM and presents a tool for test data generation. The experiments show that our tool can generate executable test sequences for EFSM models of software systems automatically in acceptable time.**

***Keywords*-EFSM; path feasibility; test sequence generation; coverage criterion; symbolic execution; searching**

I. Introduction

The Extended Finite State Machine (EFSM) is an extension of Finite State Machines (FSM) augmented with additional variables, operations and predicates. Since EFSM is more expressive than FSM, it is widely used for modeling state-based systems in different areas like object-oriented software development and communication protocols. However, the test sequence generation problem also becomes more complex than FSM. At present, there are three main points considered for this problem.

- The first one is test adequacy criterion. Traditional methods for test sequence generation almost focus on state identification. The test sequences are regarded to be sufficient if these sequences can distinguish all states from each other. It can not be denied that state identification is a good criterion, but it is difficult to satisfy, and there has been no good solution to this problem during the past decades. So researchers begin to find some new adequacy criteria.
- Since there are a number of transitions and predicates in an EFSM model, some paths will be infeasible. Thus path feasibility checking and test sequence generation for each path are two unavoidable problems. A series of techniques have been proposed to solve them, such as genetic algorithms, interval narrowing, and so on.
- Path searching is quite important for path-oriented testing. As we know, there are often an infinite number of paths when the EFSM has loops. Thus, a major issue is how to quickly find the path set that we need. Some heuristics are helpful to reduce the search time.

In this paper, we are mainly concerned with the first and the third points. For the first point, we propose to use the basis path coverage as a test adequacy criterion; while for the last point, we present a new evaluation metric method and some techniques about reducing the number of feasibility checks to improve the efficiency.

II. Background

We will give some basic concepts in this section, mainly about EFSM and path-oriented testing.

A. EFSM model

We define an EFSM model M to be a 5-tuple $\langle S, T, V, s_0, S_T \rangle$, where S is a nonempty set of states; T is a nonempty set of transitions; V is a set of variables, including context variables and input variables; s_0 is the entry state of M; S_T is a nonempty subset of S and its elements are the exit states of M.

A transition t in T is a 4-tuple $\langle s, d, pre, act \rangle$, where s and d are the starting state and tail state of t, pre is the predicate of t, which must be satisfied when t is triggered, and act is the action of t, which will be executed when t is triggered.

B. Path-oriented testing

Path-oriented testing is an important class of testing that analyzes various paths of the program. As we know, there are many or usually an infinite number of paths in an EFSM model M when it has loops. It is time-consuming to generate test sequences corresponding to all paths. Thus we need some coverage criteria. The criteria are used to measure the sufficiency of a given test suite and they can also be used to compare the quality of different test suites. We employ three commonly used coverage criteria, including state coverage, transition coverage and basis path coverage. More details about these criteria can be found in [1].

*This work is supported by NSFC (grant No. 61070039, 60903049).

III. The proposed method

Our method for generating test sequences for EFSMs can be divided into two steps:

- Find a number of feasible paths that satisfy given coverage criteria.
- Generate test sequences to trigger each path.

For the first step, we use common algorithms of graph searching, such as depth-first search (DFS) or breadth-first search (BFS), to generate a number of paths. Since we also need to decide the feasibility of paths and feasibility checking might be quite time-consuming, we modify the search algorithms to meet our needs. For the second step, symbolic execution and constraint solving are employed. In addition, we also hope that the number of test sequences is as few as possible so that the cost of testing is lower.

A. Find Feasible Paths

This subsection describes our approach to finding feasible paths satisfying given coverage criteria. The basic idea is using DFS and BFS to extend partial paths and check path feasibility. In addition, we use some effective techniques in the search procedure to improve its efficiency.

1) Search heuristics: When we want to extend the current partial path p to a new path, we will have a few choices if there are more than one transitions starting with the tail state of p. This is an important issue when searching massive space and it determines how quickly the algorithm can find the solutions that we need. Evaluation metrics are often used to solve this problem. In this subsection we give some metric for achieving transition coverage, and it can be easily extended to the other two criteria.

Let p denote the current path, $\mathcal{P}$ denote the selected paths. Let t = (s, d, pre, act), t_1 = (s, d_1, pre_1, act_1) and t_2 = (s, d_2, pre_2, act_2) represent three transitions, where s is the tail state of p. We assume that the given coverage criterion is transition coverage. Then we make the following experimental observations.

- Let $tta(t)$ denote the times that t appears in $\mathcal{P}$. If $tta(t_1)$ is greater than $tta(t_2)$, then we think t_2 may be better than t_1.
- Let $tsa(d)$ denote the times that state d appears in $\mathcal{P}$. If $tsa(d_1)$ is greater than $tsa(d_2)$, then we think t_2 may be better than t_1.
- Let $tnta(t)$ denote the number of transitions whose starting state is d and which do not appear in $\mathcal{P}$. If $tnta(t_1)$ is less than $tnta(t_2)$, then we think t_2 may be better than t_1.
- Let $tnsa(d)$ denote the number of states which are neighbors of state d and which do not appear in $\mathcal{P}$. If $tnsa(d_1)$ is less than $tnsa(d_2)$, then we think t_2 may be better than t_1.

Based on the above observations, we define the penalty values in Table I and give the formula to calculate the total penalty of t as follows.

Table I
The penalty values for guiding the search

pe(tta)	pe(tsa)	pe(tnta)	pe(tnsa)
2	1	-1	-2

$$\begin{aligned} totalPe &= tta(t) * pe(tta) + sta(d) * pe(tsa) \\ &\quad + tnta(t) * pe(tnta) + tnsa(d) * pe(tnsa) \end{aligned}$$

2) Reducing the number of feasibility checks: For the basis path criterion, an algorithm is presented in our previous work [2]. The main idea of this method is selecting basis paths on the fly. When we find a new path, we decide whether it is linearly dependent of all other selected paths, instead of deciding path feasibility right away. If the new path is linearly dependent of the selected paths, we do not need to decide its feasibility. Thus a lot of paths are pruned. Furthermore, deciding linear dependence has low time complexity.

In addition, in this paper, we employ a property about path p and p_{new} which extends transition t from p: if t has no predicate, then the feasibility of p and p_{new} are equivalent. Based on this property, we can also prune lots of paths to reduce the number of feasibility checks.

Fig. 1 shows the modified DFS algorithm to find feasible paths. $MaxLen$ is the upper bound on path length, $\mathcal{P}$ indicates the set of selected paths, path p represents the transitions in the current path that is being searched, $depth$ is the size of path p. Line 8 and 9 implement the techniques mentioned above.

```
$DFS$(int $depth$, path $p$)
 1: if $\mathcal{P}$ satisfies given criterion then
 2:     return
 3: end if
 4: if $depth > MaxLen$ then
 5:     return
 6: end if
 7: for all transition $t$ in $T$ do
 8:     extend $p$ with $t$ according to the ascending penalty value order;
 9:     if it is necessary to decide feasibility of $p$ then
10:         if $p$ is feasible then
11:             if $p$ is a complete path then
12:                 add $p$ to $\mathcal{P}$;
13:             end if
14:             $DFS(depth + 1, p)$;
15:         end if
16:     else
17:         $DFS(depth + 1, p)$;
18:     end if
19: end for
```

Figure 1. Algorithm for Generating Feasible Paths

B. Generate Test Sequences with Data

A path of EFSM can be regarded as a program fragment and the goal of generating test sequence for a path is to find an input sequence such that it can "trigger" the path, or in other words, all of the predicates in this path are true under this input sequence. In our previous work [3], we have used an approach based on symbolic execution and constraint solving, to generate the test sequences.

C. Reducing the Path Set

Considering two path sets of an EFSM model M, $\mathcal{P}_1 = \{p_1, p_2, p_3\}$ and $\mathcal{P}_2 = \{p_1, p_2\}$. Assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ cover all transitions of M. We will select $\mathcal{P}_2$ to be our path set because of the smaller number of paths in $\mathcal{P}_2$, and we call p_3 a redundant path in $\mathcal{P}_1$.

definition** 1:* A path p is a ***redundant path in path set $\mathcal{P}$ ($p \in \mathcal{P}$) if and only if the coverage of $\mathcal{P}$ is the same as that of $\mathcal{P}^*$, where $\mathcal{P}^* = \mathcal{P}$ - $\{p\}$.

definition** 2:* A path set $\mathcal{P}$ is a ***concise path set if and only if there is no redundant path in $\mathcal{P}$.

In this subsection, we briefly describe a method for translating a path set to a concise path set. For more details, see [4]. The path set reduction problem can be simply reduced to the set cover problem (SCP, http://en.wikipedia.org/wiki/Set_cover_problem). For instance, we just consider the transition set of the EFSM as the universal set and regard each set of transitions appearing in each path of the path set as subsets, when the given criterion is transition coverage. The SCP can be formulated as integer linear programming (ILP) as follows:

$$\begin{array}{ll} minimize: & \sum_{i=1}^{n} x_i \\ subject\ to: & \sum_{i=1}^{n} x_i \delta_{ij} \geq 1 \end{array}$$

for $1 \leq j \leq m$, where x_i indicates whether S_i is selected or not and δ_{ij} denotes whether j belongs to S_i or not. In summary, the path set reduction problem can be formulated as ILP. There are many sophisticated tools to solve ILP, such as lp_solve (http://lpsolve.sourceforge.net/5.5/).

Discussion

Although McCabe [5] gives a formula to calculate cyclomatic complexity, it has some premises: the tested graph must have only one entry and only one exit, and it must be structural (planar graph). Since sometimes EFSM models are allowed to be unstructured in order to represent software systems more concisely, we do some experiments about using the cyclomatic complexity directly when the model is unstructured and the given coverage criterion is basis path criterion, and the results show that the path sets we generate may not satisfy transition coverage criterion. Thus, in this article, we restrict our EFSM models to be structural.

IV. Implementation and Experiments

We developed a tool called ***Jesmat*** based on the techniques discussed in the previous sections. Its frontend (See Fig. 2) uses GEF (Graph Editor Framework, http://www.eclipse.org/gef/) to support user interaction. The user can edit his or her EFSM model using ***Jesmat*** easily through dragging controls. In the backend, we implemented the algorithm described in Fig. 1. It can give the test sequences satisfying a given criterion.

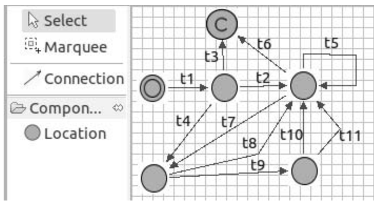

Figure 2. The Interface of Jesmat Tool

Table II
DEFINITIONS OF TRANSITIONS IN FIG. 2

Transition	Predicate	Action
t_1		$input(inLine)$; $prevSum$ = 0; $curSum$ = 0; c = $inLine[i]$; $lastdelimit$ = $true$;
t_2	(c != INF) && (c != EOF)	$i = i + 1$; c = $inLine[i]$; $lastdelimit$ = $false$; $curSum$ = $curSum$ + c;
t_3	c == EOF	
t_4	c == INF	
t_5	the same as t_2	the same as t_2
t_6	c == EOF	
t_7	c == INF	
t_8	$lastdelimit$ == $true$	i = i + 1; c = $inLine[i]$;
t_9	$lastdelimit$ == $false$	$lastdelimit$ = $true$;
t_{10}	$prevSum$ == $curSum$	$curSum$ = 0; i = i + 1; c = $inLine[i]$;
t_{11}	$prevSum$!= $curSum$	$prevSum$ = $curSum$; $curSum$ = 0; i = i + 1; c = $inLine[i]$;

A. An example

P. Ammann and J. Offutt use EFSM to represent the specification of Java class $Stuteer$ in [6]. We modify it slightly to adapt this model to our definition of EFSM. In this instance, there is only one input variable denoted by $inLine$ and it has several positive integer separated by "INF" and ends up with "EOF". The function of this class is to find out all adjacent pairs whose sum are equal. For example, if $inLine$ = {1, 4, INF, 2, 3, INF}, then the pairs $\langle(1, 4), (2, 3)\rangle$ will be found. The model can be seen in Fig. 2 and the predicates and actions for each transition are given in Table II. The situations in this instance are denoted by a triple $\langle curSum, prevSum, lastdelimit\rangle$ and this book alleges that there are only six feasible situations in this instance.

We apply our tool to this instance, and it finds one path to achieve state coverage, three paths to achieve transition coverage and seven basis paths showed in Table III. They cover all of feasible situations about the combinations of values mentioned in the book. It indicates that our approach can generate useful test sequences for EFSMs.

Table III
EXPERIMENTAL RESULTS ABOUT STUTTER

Path Set	Test Sequences
t_1t_3	$\{EOF\}$
$t_1t_2t_6$	$\{1, EOF\}$
$t_1t_4t_8t_6$	$\{INF, EOF\}$
$t_1t_2t_5t_6$	$\{1, 1, EOF\}$
$t_1t_2t_5t_7t_9t_{11}t_6$	$\{1, 1, INF, EOF\}$
$t_1t_2t_5t_7t_9t_{11}t_5t_7t_9t_{10}t_6$	$\{1, 1, INF, 2, INF, EOF\}$
$t_1t_2t_5t_7t_9t_{11}t_5t_7t_9t_{10}t_7t_8t_6$	$\{1, 1, INF, 2, INF, INF, EOF\}$

Table IV
EXPERIMENTAL RESULTS ABOUT THREE SOFTWARE SYSTEMS

System	Size	Criterion	Original /Reduced	Heuristics /No Heuristics
ATM	10/20	State	16/5	16/282
		Transition	84/12	88/1165
		Basis path	22/-	22/1039
Lift	4/29	State	45/3	81/299
		Transition	190/7	468/1758
		Basis path	26/-	26/1503
Flight	4/32	State	5/1	5/27
		Transition	126/21	126/1515
		Basis path	30/-	30/1277

B. Some experiments

Table IV shows the experimental results on three different software systems. They are adapted from the case studies in A.S. Kalaji *et al* [7]. The second column shows the size of each EFSM model, including the number of states and transitions. The 4th column compares the number of paths generated initially and the number of paths after reduction, and the last column compares the searching and feasibility checking times when the heuristics are used and when they are not used.

V. RELATED WORKS

In [8], Ramalingom *et al.* proposed a method called CIUS (Context Independent Unique Sequence) to generate test sequences for EFSMs. A CIUS of each state is a test sequence that can identify itself with other states and whose feasibility of the corresponding path is independent of the context. Unfortunately, not all EFSMs have CIUSes. There are many similar works based on the FSM approach, like this method. They all focus on state identification and can not be applied to general EFSM models. Most of the EFSM models in modern software systems have easily distinguished states and we find that a path set satisfying transition or basis path coverage criterion is often able to identify all states of the EFSM model. Kalaji *et al.* [7] use genetic algorithms to generate test sequences for EFSMs. Feasibility metric is used in the evolutionary procedure, but it is just an experimental metric. For example, it considers that the predicate containing "$==$" is more infeasible than the predicate containing "$\geq$". Derderian *et al.* [9] also employ this feasibility metric, but his experiments show that this metric does not always work. Compared with these approaches, our method can almost always find a small set of executable paths and the corresponding test data.

VI. CONCLUSION

There are two contributions of this paper. The first one is that we first apply the basis path coverage on test sequence generation for EFSMs and we discuss the problem of the cyclomatic complexity of EFSMs. Previous works based on path-oriented testing mostly use the transition coverage criterion and as we know, basis path coverage is a more sufficient criterion than transition coverage. The second contribution is that we propose a series of evaluation metrics to improve the efficiency of feasible path searching. Our experiments show that this method is efficient for generating test sequences for EFSM models, in that the results have high coverage and they are small in size.

In the future, we are going to improve our evaluation metrics by analyzing deep relationship between penalty values and the EFSM structure. In additon, we will try to experiment with some EFSM models of real software systems and find out defects in them.

REFERENCES

[1] H. Zhu, P. Hall, and J. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, no. 4, pp. 366–427, 1997.

[2] J. Yan and J. Zhang, "An efficient method to generate feasible paths for basis path testing," *Information Processing Letters*, vol. 107, pp. 87–92, 2008.

[3] J. Zhang, C. Xu, and X. Wang, "Path-oriented test data generation using symbolic execution and constraint solving techniques," in *Proceeding of the Second International Conference*, 2004, pp. 242–250.

[4] H. Wang, S. Hsu, and J. Lin, "A generalized optimal path-selection model for structural program testing," *Journal of Systems and Software*, vol. 10, pp. 55–63, 1989.

[5] T. J. McCabe, "A complexity measure," *IEEE Transactions On Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[6] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge University Press, 2008, pp. 80–84.

[7] A. Kalaji, R. Hierons, and S. Swift, "An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models," *Information and Software Technology*, vol. 53, no. 12, pp. 1297–1318, 2011.

[8] T. Ramalingom, K. Thulasiraman, and A. Das, "Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines," *Computer Communications*, vol. 26, pp. 1622–1633, 2003.

[9] K. Derderian, R. Hierons, M. Harman, and Q. Guo, "Estimating the feasibility of transition paths in extended finite state machines," *Automated Software Engineering*, vol. 17, pp. 33–56, 2010.