A Search-Based Approach for Automatic Test Generation from Extended Finite State Machine (EFSM)

AbdulSalam Kalaji, Robert M Hierons and Stephen Swift
School of Information Systems, Math and Computing
Brunel University
Uxbridge, UB83PH, UK
e-mail: {abdulsalam.kalaji,rob.hierons,stephen.swift}@brunel.ac.uk

Abstract— **The extended finite state machine is a powerful model that can capture almost all the aspects of a system. However, testing from an EFSM is yet a challenging task due to two main problems: path feasibility and path test data generation. Although optimization algorithms are efficient, their applications to EFSM testing have received very little attention. The aim of this paper is to develop a novel approach that utilizes optimization algorithms to test from EFSM models.**

I. Introduction and Problem Area

Errors in a system, if not eliminated, can lead to unexpected outcomes and testing is therefore an important process. However, manual testing is slow, expensive and error-prone, thus automation is desirable.

Model based testing represents the system specification in a form of a model which is then used to derive tests that can be applied to the system implementation. Finite state machine (FSM) and Extended-FSM are commonly used for the purpose of model based-testing. An FSM can represent the control part of a system such as a traffic light system. However, an EFSM is needed to represent more complex systems usually with control and data parts such as communication protocols. An EFSM essentially comprises states, variables and transitions among states. Generally, a transition has guards (*g*) and actions (*op*) over the machine variables where guards must be satisfied in order for this transition to be taken and so for the associated actions to be executed. For example, Fig. 1 shows an EFSM for a simple flight safety system.

The EFSM is a powerful approach for modeling and has been widely applied to many areas in order to derive test sequences from systems of interest. Despite its popularity, testing from this model is still a challenging task due to two main problems: the path feasibility and path test data generation.

Due to the presence of guards and actions, a conflict may occur among two or more transitions in a given transition path (TP). For example, a transition's action may set a variable $x = 0$ and the next transition guard checks if $x>0$. Such a path is infeasible and so it is impossible to find test data that can trigger it. However, determining if a given path is feasible in advance is undecidable. In addition, the development of good methods for this problem is an open

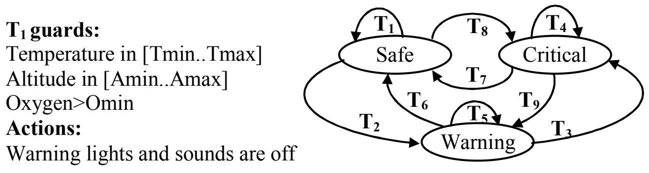


Figure 1. An EFSM example of a simple flight safety system

research problem [1]. If the path is feasible, then a set of input values is required to trigger (exercise) this path. Nevertheless, finding such a set is a difficult task since the input domain is usually quite large but the required values are just a small subset of this domain [2] e.g., in Fig. 1, the parameter Temperature is of Integer type, however, the required value to test transition T_1 must be in [Tmin..Tmax].

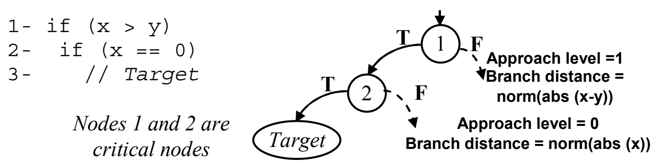
There are many approaches that study these two problems (path feasibility and path test data generation) by converting EFSM to FSM for which many test techniques are available. The conversion is conducted by either abstracting the data away from an EFSM so it is FSM or expanding the EFSM to become FSM [3]. While the first approach does not guarantee that the paths taken from the FSM will be feasible in the corresponding EFSM, the other approach can easily lead to the number of the states in the resultant FSM being prohibitively large.

Although optimization algorithms are recognized to be efficient for testing purposes [4], very little attention has been paid towards investigating their application for testing from EFSM. Therefore, the aim of this paper is to tackle these two problems and develop a novel approach to test from EFSM models by employing optimization algorithms.

II. The Proposed Approach

A. *The Problem of Selecting Feasible Paths*

Since the path feasibility problem is undecidable, we propose a fitness metric to estimate the likelihood of a given path being feasible. The fitness metric is based on analyzing all the relations among the transitions of a path in order to estimate its feasibility. This requires categorizing the machine transitions to two types: affecting and affected-by. A transition is an affecting within a TP if it has an operation which can affect the guards of a later transition, the affected-by. For example, a transition that assigns a value to a variable *x* can affect later transitions that have guards over *x* in a given TP. Then, each pair of (affecting, affected-by) is



```
1- if (x > y)
2-  if (x == 0)
3-    // Target
```

T
1
F
Approach level =1
Branch distance = norm(abs (x-y))
T
2
F
Approach level = 0
Branch distance = norm(abs (x))
Target
Nodes 1 and 2 are critical nodes

Figure 2. A fitness calculation example

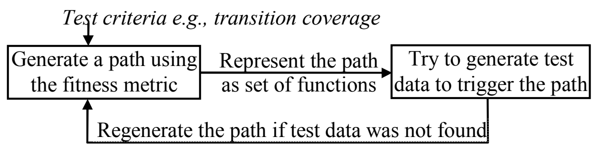
assigned a numeric value depending on the type of the assignments and guards found in this pair respectively. For example, consider a pair of (affecting, affected-by) with (*op*: $x = 10$) and (*g*: $x != 0$) respectively. The fitness metric value assigned to such a pair represents how easy it is to satisfy this guard. The latter value is certainly much less than a value given to a pair with (*op*: $x = 10$) and (*g*: $x <= 0$) since this corresponds to infeasible case. Similarly, a pair with (*op*: $x = p$) and (*g*: x != 0), where p is a parameter, is better than a pair with (*op*: $x =p$) and (*g*: $x== 0$) since in the first case many values of p can achieve the guard, however only one value of p (zero) in the second case satisfies the guard. In this way, a given TP can be examined for all existed pairs of (affecting, affected-by) and assigned an integer value which estimates its feasibility. The lower the fitness metric value the more likely it is a feasible path. The fitness metric and the algorithm that calculates it are described in [5].

B. Path Test Data Generation

Any EFSM transition can be considered as a function where the function name and input parameters are derived from the corresponding transition name and input parameters. Therefore, a path test data is a set of inputs to be applied to a set of functions which are called sequentially. Given this description, a fitness function is required to guide the search for a suitable set of inputs. Wegener et al. [6] described a fitness function (Equation 1) in the presence of nested IF statements that comprises two components: a *branch distance* [7] and the *approach level* (Equation 2) to measure how close a particular input was to executing the target branch that is missed and how many critical nodes are away from the target respectively. The critical node is a branching node (see Fig. 2) at which the path control flow may divert. Since it is necessary to contrast how many conditions were achieved by a specific input, the branch distance of each IF statement is normalized, using *norm* function, to a value in the range of [0..1] (Equation 3).

$$fitness = approach\ level + norm\ (branch\_distance) \qquad (1)$$
$$approach\_level = 1 - NumCriticalNodeFromTarget \qquad (2)$$
$$norm\ (branch\_distance)\ = 1 - 1.05^{-branch\_distance} \qquad (3)$$

The latter fitness calculation can be applied to one function at a time. The work of Wegener et al. [6] was found to be more efficient when compared to Tracey et al. work [7] since considering only the branch distance to calculate the fitness can lead to plateaux in the fitness function landscape and these can cause the search to be stuck in local minima [4]. However, for a set of functions, a new fitness function is required. We propose a fitness function which consists of two components: *function distance* and *function approach level*: a *function distance* is calculated by using the approach of Wegener et al. [6] and thus a *function distance* is zero



Test criteria e.g., transition coverage
Generate a path using the fitness metric
Represent the path as set of functions
Try to generate test data to trigger the path
Regenerate the path if test data was not found

Figure 3. The proposed framework to test from EFSM

when the corresponding transition is triggered otherwise a *function distance* reflects how close a given input was to executing the subject function. Then each guarded function in a path is considered as a critical node and so we compute the *function approach level* which reflects how many critical functions (transitions) are away from the target (Equation 4).

$$fitness = fun\_approach\_level + norm(fun\_distance) \qquad (4)$$

A similar notion of calculating a path fitness is introduced in [8] however they calculate the fitness for each function in a path by using Tracey et al. [7] approach. Since, as argued in [4], the Tracey et al. approach can experience problems in the presence of nesting; therefore, we propose using the approach of Wegener et al. [6] to calculate the *function distance* which is more efficient.

Fig. 3 shows the proposed framework for testing from EFSMs which comprises two phases: firstly, the fitness metric is used by an evolutionary approach to generate a transition path that is likely to be feasible. Then, another evolutionary approach is used to try to trigger the path using the proposed path fitness calculation. If the second phase is not successful, we iterate.

REFERENCES

[1] A. Y. Duale and M. U. Uyar, "A method enabling feasible conformance test sequence generation for EFSM models," Computers, IEEE Transactions on, vol. 53, pp. 614-627, 2004.

[2] H. Ural and B. Yang, "A test sequence selection method for protocol testing," Communications, IEEE Transactions on, vol. 39, pp. 514-523, 1991.

[3] R. M. Hierons and M. Harman, "Testing conformance of a deterministic implementation against a non-deterministic stream X-machine," Theoretical Computer Science, vol. 323, pp. 191-233, 2004.

[4] P. McMinn, "Search-based software test data generation: a survey: Research Articles," Software Testing, Verification & Reliability, vol. 14, pp. 105-156, 2004.

[5] A. S. Kalaji, R. M. Hierons, and S. Swift, "Generating Feasible Transition Paths for Testing from an Extended Finite State Machine (EFSM)," in Proceeding of the 2nd IEEE International Conference on Software Testing, Verification, and Validation, pp. 230-239, 2009.

[6] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," Information and Software Technology, vol. 43, pp. 841-854, 2001.

[7] N. Tracey, J. Clark, K. Mander, and J. McDermid, "An automated framework for structural test-data generation," in Proceeding 13th IEEE International Conference on Automated Software Engineering, pp.285-288, 1998.

[8] R. Lefticaru and F. Ipate, "Functional Search-based Testing from State Machines," in Proceedings of the 1st IEEE International Conference on Software Testing, Verification and Validation, pp. 525-528, 2008.