On Code Coverage of Extended FSM Based Test Suites: An Initial Assessment

Khaled El-Fakih, Tariq Salameh, Nina Yevtushenko

On Code Coverage of Extended FSM Based Test Suites: An Initial Assessment

Khaled El-Fakih[1], Tariq Salameh[1], Nina Yevtushenko[2]

[1]American University of Sharjah, Sharjah, UAE
kelfakih@aus.edu, b00046306@aus.edu
[2] Tomsk State University, Tomsk, Russia
ninayevtushenko@yahoo.com

Abstract. In this paper, we present an assessment of the fault coverage of several Extended FSM (EFSM) based test suites. More precisely, EFSM specifications and their corresponding Java implementations are considered. Mutants of these implementations are derived using the standard arithmetic, conditional, and relational mutation operators. The fault coverage of test suites derived by different EFSM based strategies are evaluated and obtained results are summarized.

Keywords: Extended FSM test suites, test coverage, case study.

1 Introduction and Summary

Extended Finite State Machines (EFSMs) provide a rigorous approach for the development of functional tests for reactive systems, protocols and software. As an EFSM specification expresses both the data-flow and the control-flow behavior of a system, test derivation from an EFSM can be done using traditional software test selection criteria, such as the well-known All-Uses criterion, or using other criteria such as the derivation of test suites (TSs) that cover all EFSM mutants with single transfer faults, called STF TS, or a test suite, with one test case, called Transition Tour (TT), that traverses each transition of the specification, or a test suite, called DSTS, with tests that for each transition of the specification, the test suite includes input sequences that traverse the transition appended with a set of input sequences that distinguish the ending state of the transition from all other states of the machine. Another possibility for test derivation is to randomly derive a test suite with one test case of a particular length, for example, a test case with the same length as an All-Uses test suite [1], called Random All-Uses, or as investigated in this paper, the derivation of a test case of the same length as a DSTS test suite, called Random DSTS. In [1] it is shown that a test suite that covers All-Uses may not detect all STF mutants of the EFSM specification and a test suite that covers all STF mutants of the specification may not satisfy the All-Uses criterion of the specification, etc. Thus, an empirical study is needed to assess the coverage of these test suites.

For evaluating the code fault coverage, in this paper, four known EFSM specifications and corresponding Java code implementations are considered. Then All-Uses, STF, TT, DSTS test suites are derived and the coverage of these test suites is calculated using mutants of the Java implementations where code mutants are derived using the traditional arithmetic, logical, and conditional operators [4].

Major contributions of the conducted experiments can be summarized as follows:

- All-Uses, STF, and TT test suites provide comparable (fault) coverage.
- DSTSs outperform all other considered test suites.
- Random-All-Uses and All-Uses test suites provide comparable coverage where DSTS test suites slightly outperform Random-DSTSs.
- Test suites coverage of conditional faults is significantly higher than their coverage of mutants with arithmetic, logical, or relational faults.
- Test suites coverage of mutants with relational faults is much less than that of the coverage of mutants with arithmetic, conditional or logical faults.
- All-Uses and TT test suites both achieve comparable coverage of mutants with arithmetic faults; however, STF test suites have significantly lower coverage of arithmetic faults than All-Uses and TT test suites.
- All considered test suites provide comparable coverage of conditional faults, also all test suites provide comparable coverage of logical (or relational) faults.

Empirical assessment studies related to code based data-flow, control-flow, mutation testing, and some specification based test selection criteria are studied in many papers which are mostly summarized in [3] and [5]. For specifications modeled as EFSMs, a preliminary assessment of STF, TT, All-Uses, All-Predicates, double transfer faults and some random test suites has been recently presented in [1]. The assessment is done based on the coverage of the test suites of EFSM mutants of the specification with single and double transfer faults. In this paper, unlike [1], the considered test suites are assessed in terms of their coverage of code mutants of implementations of these specifications. Thus, the work allows us to compare the coverage of considered test suites w.r.t. known code based types of mutants. Further, in this paper, DSTS test suites are also considered in the assessment and the results show that such test suites outperform all other considered test suites. Finally, it is worth stating that the results obtained here are rather different than those in [1]. For example, in [1] it is found that All-Uses, TT, and STF test suites do not provide comparable coverage of EFSM mutants with single or double transfer faults while in this paper these test suites achieve comparable coverage of code mutants.

2 Preliminaries

The EFSM Model: Let X and Y be finite sets of inputs and outputs, R and V be finite disjoint sets of parameters and context variables, respectively. For $x \in X$, $R_x \subseteq R$ denotes the set of input parameters and D_{Rx} denotes the set of valuations of the parameters over the set R_x. Similarly, for $y \in Y$, $R_y \subseteq R$ denotes the set of output parameters and D_{Ry} denotes the set of valuations of the parameters over the set R_y. The set D_V denotes the set of context variable valuations.

An EFSM [6] M over X, Y, R, V with the associated valuation domains is a pair (S, T) where S is a finite non-empty set of states and T is the set of transitions between states in S, such that each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where, s and s' are the *start* and *final* states of the transition t, $x \in X$ is the *input* of transition t, $y \in Y$ is the *output* of transition t, P and up, op are functions defined over context varia-

bles V and input parameters as follows: P: $D_{Rx} \times D_V \rightarrow \{True, False\}$ is the *predicate* of the transition t; $up : D_{Rx} \times D_V \rightarrow D_V$ is the *context update* function of transition t; $op : D_{Rx} \times D_V \rightarrow D_{Ry}$ is the *output parameter update* function of transition t. A context variable valuation $\mathbf{v} \in D_V$ is called a *context* of M. A *configuration* of M is a tuple (s, $\mathbf{v}$) where s is a state and $\mathbf{v}$ is a context. Given input x and the input parameter valuations, a (*parameterized*) *input* is a tuple (x, $\mathbf{p}_x$), where $\mathbf{p}_x \in D_{Rx}$. A sequence of inputs is also called *an input sequence*. An *output sequence* can be defined in a similar way. In this paper, hereafter, we consider deterministic complete EFSM specifications, i.e. for each input sequence there exists exactly one corresponding output sequence. If an EFSM is partial then a complete version of the machine is obtained by appropriately augmenting the machine will transitions with the null output.

Assume that EFSM M is at a current configuration (s, $\mathbf{v}$) and the machine receives an input ($\boldsymbol{x}$, $\mathbf{p}_x$) such that ($\mathbf{v}$, $\mathbf{p}_x$) satisfies the guard P of an outgoing transition t = (s, x, P, op, y, up, s'). Then when receiving the input ($\boldsymbol{x}$, $\mathbf{p}_x$), the machine executes the update statements of t; produces the (parameterized) output where parameter values are provided by the output parameter function op, and moves to configuration (s', $\mathbf{v}'$), where $\mathbf{v}' = up(\mathbf{p}_x, \mathbf{v})$. Thus, such an execution of transition t can be represented as (s, $\mathbf{v}$) - (x, $\mathbf{p}_x$)/(y, $\mathbf{p}_y$) -> (s', $\mathbf{v}'$), where $op(\mathbf{p}_x, \mathbf{v}) = \mathbf{p}_y$. Such a transition can also be written as ((s, $\mathbf{v}$), (x, $\mathbf{p}_x$), (y, $\mathbf{p}_y$), (s', $\mathbf{v}'$)).

It is well known that not each transition with the start state s is executable at any configuration (s, $\mathbf{v}$). A *path* is a sequence of consecutive transitions (s_1, x_1, P_1, op_1, y_1, up_1, s_2)…. (s_l, x_l, P_l, op_l, y_l, up_l, s_l). A path is *feasible* or *executable* if there is a sequence of (executable) transitions (s_1, $\mathbf{v}_1$) - (x_1, $\mathbf{p}_{x1}$)/(y_1, $\mathbf{p}_{y1}$) -- (x_l, $\mathbf{p}_{xl}$)/ (y_l, $\mathbf{p}_{yl}$) -> (s_l, $\mathbf{v}_l$) in EFSM M starting from configuration (s_1, $\mathbf{v}_1$). The *input/output projection* of such an executable path is the *sequence of input/output pairs* (x_1, $\mathbf{p}_{x1}$)/(y_1, $\mathbf{p}_{y1}$) .. (x_l, $\mathbf{p}_{xl}$)/ (y_l, $\mathbf{p}_{yl}$) and is called a *trace* of M starting from configuration (s_1, $\mathbf{v}_1$).

A test case is a (parameterized) input sequence of the EFSM specification. A *test suite* (*TS*) is a finite set of test cases. The length of a test suite *TS* is the total length of its corresponding test cases.

EFSM Based Test Suites:

Single Transfer Faults (STF) Test Suites: Given an EFSM M, a transition t = (s, x, P, op, y, up, s') of an EFSM IUT M' has a *transfer fault* if its final state is different from that specified by M, i.e., M' has a transition (s, x, P, op, y, up, s''), $s'' \neq s'$, $s'' \in S$. Such M' is a *mutant of M with a transfer fault*. EFSMs M and M' are distinguishable if their initial configurations are distinguishable by an input sequence (or a test case) α. In this case, we say that α *kills* M'. A test suite, called a STF test suite, covers single transfer faults of M, if for each mutant of M with a single transfer fault that is distinguishable from M, the test suite has at least one test case that kills such a mutant.

Transition Tour (TT) Test Suites: A TT test suite of M is an input sequence that starts at the initial configuration of M and traverses each transition of M.

EFSM Flow-Graph Based All-Uses Test Suites: All-Uses test suite is a set of test cases of EFSM M that cover the All-Uses of each context variable and every parameterized input of M. Such a test suite can be derived directly from M as illustrated in [2] or from a flow-graph representation of M as illustrated in [7].

Distinguishing Set Test Suites (DSTS): An input sequence α_{ij} is a *distinguishing sequence* for states s_i and s_j of M if α_{ij} distinguishes each pair of configurations (s_i, **v**) and (s_j, **v**′), **v**, **v**′ $\in D_V$, of M. M is *state reduced* if each two different states of M are distinguishable. Given state $s_j \in S$ of a state reduced EFSM M with n states, a set W_j of input sequences is called a *distinguishing set* of state s_j, if for any other state s_i there exists a sequence $\alpha \in W_j$ that distinguishes states s_i and s_j. Given distinguishing sets $W = \{W_0, W_1, .. , W_{n-1}\}$ of states of M, a Distinguishing Sets Test Suite (DSTS) is a set of test cases that satisfies the following property. For every transition $t = (s, x, P, op, y, up, s')$ of M and each $\alpha \in W_j$, the TS has the input sequence $\gamma.(x, \mathbf{p}_x).\alpha$, where γ is the input sequence that takes M from the initial configuration to a configuration (s, **v**) such that (**v**, $\mathbf{p}_x$) satisfies P of t.

3 Assessment Methodology, Results, and Further Work

We consider four known EFSM specifications, namely, complete EFSM versions the Initiator and Responder of the INRES protocol, the SCP, and the Cruise Control. The method has three steps. In **Step 1**, for each considered EFSM specification, all EFSM mutants of M with STF faults are derived and a corresponding STF test suite (with optimal or near optimal length) is derived as illustrated in [1]. Moreover, for every specification, a corresponding flow-graph representation annotated with definitions and uses of variables is constructed and then a corresponding All-Uses test suite is derived from the obtained flow-graph as illustrated in [1] based on related work [7]. In **Step 2**, three corresponding Java code implementations are developed by three different software engineers, based on the EFSM specification and its textual description, under the following coding rules. State variables cannot be explicitly or implicitly introduced in an implementation, for example, no state variables and no flags indicating state variables can be used; moreover, no labels and no Go-to statements can be used. In addition, names of context variables, inputs and outputs with their parameters of the EFSM specification should be preserved in a code implementation. Each implementation should be implemented as one function that inputs a string separated by a delimiter "," representing an input sequence to the function and returns as an output a string representing the output response of the implementation to the input sequence. A Reader/Writer class is used in all implementations that handles reading/writing the input and the output strings in order to separate reading and writing outputs from the function that implements the specification and thus, code mutants are only derived from the function that implements the specification. We note that before deriving mutants, each Java implementation is thoroughly tested using all the considered test suites written in JUnit. In **Step 3**, 1-Order Java code mutants are derived using the Java arithmetic, relational, conditional, logical-shift, and assignment operators shown in List 1. As usual, 1-Order code mutants are considered to alleviate problems related to the coupling effect of using N-order mutants, when $N > 1$. Afterwards, the fault detection capabilities (*fault coverage*) of each considered test suite of a given EFSM specification is measured as (J_{killed} / $J_{Mutants}$)×100, where $J_{Mutants}$ denotes the number of derived mutants of the Java implementation and J_{killed} is the number of these mutants killed by the given test suite. MuClips [8], MuJava [4], and JUnit are used for the

automatic derivation of mutants, execution of test suites, and for determining fault coverage.

List 1. Selected Mutation Operators [4]:
(AORB) Arithmetic Operator Replacement, (AORS) Arithmetic Operator Replacement– Shortcut, (AOIS) Arithmetic Operator Insertion – Shortcut, (AOIU) Arithmetic Operator Insertion – Unary, (COI) Conditional Operator Insertion, (COD) (Conditional Operator Deletion, (LOI) Logical Operator Insertion.

Results: This part contains the results of all conducted experiments. Fig. 1 includes the coverage of the All-Uses, STF, TT, and DSTS test suites. These results clearly show that the All-Uses and STF have almost the same coverage, TTs slightly outperforms (approximately by 2%) All-Uses and STF test suites. However, DSTSs outperform all other test suites by at least 5%. This pattern is almost the same for each considered example. These results clearly show the importance of using state identifiers based test suites in EFSM based test derivation.

Fig. 2 shows the coverage of all considered test suites using arithmetic, conditional, logical, and relational operators. According to these results, there is a significant difference between the coverage per mutation operator, for example, the coverage of conditional faults is 15.3, 23.5, and 32.5 percent higher than the coverage of mutants with arithmetic, logical, or relational faults. In addition, the coverage of relational faults is much less than the coverage of mutants with arithmetic, conditional and logical faults. Fig. 3 depicts the coverage of each test suite per each mutation operator. According to these results, it is clear that there is a huge difference, approximately by 16 percent, of the coverage of arithmetic faults by the STF test suites in comparison with the All-Uses and TT test suites. As expected, the All-Uses (data-flow based) test suites coverage of arithmetic faults is higher than that of STF (control-flow based) test suites; however, we were surprised that TT test suites provide comparable (even slightly higher) coverage of arithmetic faults as the All-Uses test suites. All considered test suites provide comparable coverage of conditional faults. Also, all test suites achieve comparable coverage of logical (or relational) faults.

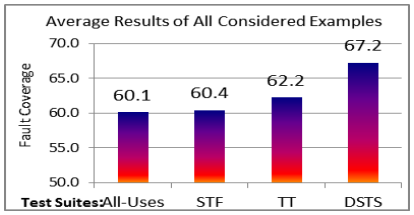

Fig. 1. Coverage of test suites

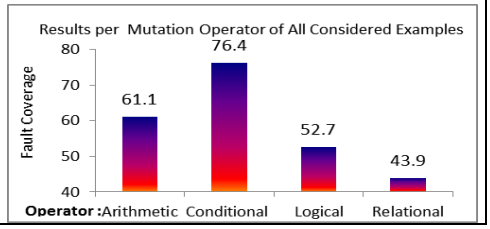

Fig. 2. Results per mutation operator

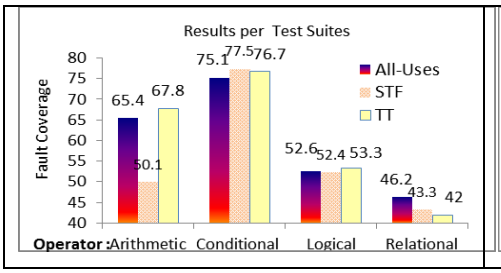

Fig. 3. Results per mutation operator of test suites

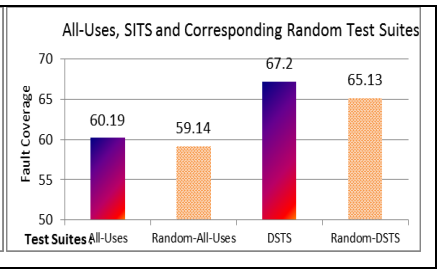

Fig.4. results with random test suites

Experiments with Random Test Suites: As All-Uses and TT and STF test suites provide comparable coverage and are all outperformed by the DSTS test suites, we were curious to determine if All-Uses and DSTS test suites would outperform random test suites derived as (executable) random paths of the EFSM transitions. To this end, for each considered example, we considered the All-Uses (DSTS) test suite and derive three random test suites, called Random-All-Uses (Random-DSTS); each random test suite has one test case that has the same length as the corresponding All-Uses (DSTS) test suite. Obtained results, shown in Fig. 4, show that All-Uses and Random-All Uses provide comparable coverage; however, DSTSs slightly outperform (by 2%) Random DSTSs. These results clearly indicate the good performance of such random test suites and further experiments are needed to verify these important results.

Future Work: Though a clear pattern of the fault coverage of the considered test suites is indicated using the four conducted experiments, yet, experiments with more application examples would be useful to clearly confirm the obtained pattern. Assessing the coverage of more types of EFSM based, especially random, test suites would also be useful as future work on EFSM based testing. It would also be useful to consider length of obtained test suites and assess the relationship between length of a generated random test suite and its fault coverage.

References

1. El-Fakih, K., Simao, A., Jadoon N., Maldonado J.C.: On studying the effectiveness of extended finite state machine based test selection criteria. IEEE ICST Workshops, Mutation 2014 (2014)
2. Bourhfir, C., Abdoulhamid, E., Khendek, F., Dssouli, R.: Test cases selection from SDL specifications, *Computer Networks*, 35(6), 693-708 (2001)
3. Jia, Y., Harman, M.: An anaylsis an survey of the development of mutation testing, *IEEE TSE*, 37(5), 649-678 (2011)
4. Ma, Y.-S., Offutt, J., Kwon, Y. R.: MuJava: An automated class mutation system. *JSTVR*, 15(2), 97–133 (2005)
5. Mathur, A.: *Foundations of Software Testing*, Addison-Wesley, (2008)
6. Petrenko, A., Boroday, S., Groz, R.:Confirming configurations in EFSM Testing, *IEEE TSE*, 30(1), 29-42 (2004)
7. Ural, H., Williams, A.: Test generation by exposing control and data dependencies within system specifications in SDL, In *Formal Description Techniques*, pp.335-350 (1993)
8. MuClipse Team. (2011, September) MuClipse - Mutation Testing tool for Eclipse. [Online]. http://muclipse.sourceforge.net/