Event-Based Mutation Testing vs. State-Based Mutation Testing – Comparison Using a Web-based System

Mutlu Beyazıt[1], Timon Sebastian Deistler[1], Nida Gökçe[2]

[1]Faculty of Computer Science, Electrical Engineering and Mathematics,
University of Paderborn, Germany
beyazit@adt.upb.de, timond@teleos-web.de

[2]Faculty of Arts and Science, Muğla University, Turkey
on the leave at University of Paderborn, Germany
goekce@adt.upb.de

Abstract: The idea of using models in mutation analysis dates back to mid 1980s. Recently, it is shown that, with some proper adaptations, mutation testing can also be utilized as a black-box testing approach. For this purpose, a modeling structure, a set of mutation operators, coverage criteria and test case generation methods are required. This paper compares event-based and state-based realizations of this testing approach over a case study and achieves experimental and comparative results, while validating the applicability of the discussed notions.

1 Introduction

Mutation testing [DLS78] can be defined as a white-box testing technique based on generating faulty versions (mutants) of software by introducing small but typical faults using mutation operators. It aims at evaluating adequacy of test sets by their ability to reveal these mutants (or faults). On the other hand, using mutated models of the system [BBW06] can be deployed as a black-box testing technique, and test generation process can also be evaluated. For example, the original model can be mutated to generate test cases to be executed on the system. Doing so, a mutant is said to be *live* if all its test cases yield the expected output when executed on the system, and it is said to be *killed* if these outputs differ. Failures are also expected to be found during test execution. However, typically, many mutants can be generated, resulting in a massive testing effort, and, mutants, in terms of behavior, can be equivalent to each other.

The objective of this paper is to compare state-based and event-based mutation testing approaches. The state-based approach, discussed here, makes use of finite state automata [HMU01], more specifically Moore finite state machines (MFSMs) [Mo56]. They focus on states, events and outputs of these events in a modeled process. On the other hand, the event-based approach, modeled here by event sequence graphs (ESGs) [Be01], focus only on events of a process. When compared to MFSMs, ESGs are easier to learn and apply, therefore it is intuitive and common to think that MFSMs are more powerful in

terms of detecting the failures/faults. We will check the validity of this argument whether it does necessarily hold by making use of, and with respect to, the following aspects:

(1) *Modeling structure.* As already mentioned, the present paper makes use of ESGs [Be01] and MFSMs [Mo56].

(2) *Mutation operators.* Here, different, but structural, mutation operators are selected for ESGs and MFSMs. Only (a subset of) first order mutants are created, inserting small changes, i.e., not every mutant is generated or used.

(3) *Coverage criteria.* Coverage criteria are used to measure how well the system is exercised by a test suite [Bi00]. In present paper, two different coverage criteria, arc coverage and transition coverage, are used for ESGs and MFSMs, respectively.

(4) *Test case generation.* Minimum set of test cases, more specifically test sequences achieving the above mentioned coverage, are generated by solving slightly different variants of Chinese Postman Problem [EJ73] over the respective underlying graphs of ESG and MFSM models.

The rest of the paper is outlined as follows: Sections 2 and 3 explain the elements used in event (ESG) and state (MFSM) based approaches respectively. Then, in Section 4, the introduced concepts are applied over a case study and the results are given. Finally, in Section 5, the paper is concluded, also making some comparative remarks.

2 Event Sequence Graphs

Definition. An *event sequence graph (ESG)* can be used to represent a system's behavior interacting with user's actions. Nodes of an ESG are behavioral events of the system and arcs are sequences of these events. Thus, an ESG is a tuple $G = (N, A, [,])$ where

- N is a finite set of *nodes* or *events*,
- $A \subseteq N \times N$ is a finite set of *arcs* or *event sequences*,
- $[\in \mathrm{N}$ is a unique pseudo *start* node and, for $n \in N$, $(n, [) \notin A$, and
- $] \in \mathrm{N}$ is a unique pseudo *finish* node and, for $n \in N$, $(], n) \notin A$.
- each node is *useful*, i.e., for $n \in N$, n is reachable from start node and finish node is reachable from n.

Note that the start node has no incoming arcs and the finish node has no outgoing arcs. They are simply used to mark entry and exit events in a system. For that matter, the arcs connected to these nodes are often referred to as pseudo arcs, whose number is ≥ 2.

Mutation Operators. *3* mutation operators are defined for ESGs as follows: (1) *Insert arc (iA) operator* adds a new non-pseudo arc to the model, (2) *delete arc (dA) operator* removes an existing arc from the model, and (3) *reverse arc (rA) operator* reverses the direction of an existing non-pseudo arc in the model (See Figure 1). Application of dA and rA operators may invalidate the ESG model or the usefulness of some nodes. Thus, following formulae give only some upper bounds on the number of first order mutants:

- #iA_mutants = (#nodes - 2)^2 + 2 (#nodes - 2) - #arcs
- #dA_mutants = #arcs
- #rA_mutants = #arcs - #pseduoarcs - #loops - #symmetricarcs

where a symmetric arc is an arc whose reversion is also an arc.

(a) Original ESG (b) iA Mutant (c) dA Mutant (d) rA Mutant

Figure 1: Sample ESG Mutants

Test Generation. While generating test sequences from ESG models, the arc coverage criterion is quite a sensible choice. This criterion entails coverage of all event sequences of length *2*. In order to generate a test set achieving arc coverage from a given ESG, a variant of Chinese Postman Problem is solved. Thus, a minimum test set is constructed.

3 Moore Finite-State Machines

Definition. *Moore finite state machines (MFSMs)*, in contrast to ESGs, focus on states of the system and represent the events by including them in transitions. They also include outputs in the states. A MFSM is a tuple $M = (S, S0, E, O, T, G)$ where

- S is a finite set of *states*,
- $S0 \in S$ is a *start state* or *initial state*,
- E is a finite set of *input symbols* or *events*, called the *input alphabet*,
- O is a finite set of *output symbols*, called the *output alphabet*,
- $T: S \times E \rightarrow S$ is a *transition function*, and
- $G: S \rightarrow O$ is an *output function*.

Although, a MFSM does not have a set of final states, here, it is assumed that they have a non-empty set of final states F (to mark exit events or final states) and each state in the machine is useful. Also, note that, by definition, MFSMs are deterministic.

Mutation Operators. MFSMs are manipulated by the following *3* mutation operators: (1) *Insert transition (iT) operator* adds a new transition to the model, (2) *delete transition (dT) operator* deletes a transition from the model, and (3) *reverse transition (rT) operator* reverses an existing transition in the model (See Figure 2). Note that when compared to their ESG counterparts, MSFM mutation operators use state information to some level. Also, application of these operators may produce an invalid MFSM in the sense that the resulting FSM may be non-deterministic or some states may lose their usefulness. Thus, following formulae can be used to determine only some upper bounds on the number of first order mutants produced using the defined operators:

- #iA mutants = (#events) (#states)^2 - #transitions
- #dA mutants = #transitions

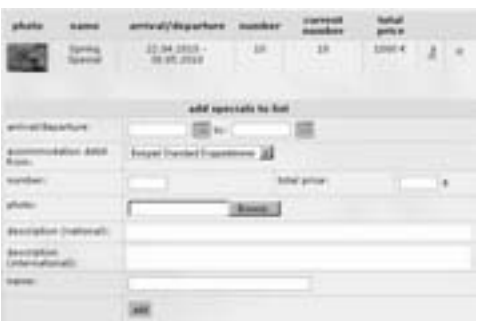
- #rA mutants = #transitions - #loops - #symmetrictransitions

where a symmetric transition is a transition whose reversion is also a transition.

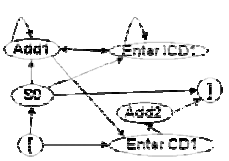
(a) Original FSM (b) iT Mutant (c) dT Mutant (d) rT Mutant

Figure 2: Sample MFSM Mutants

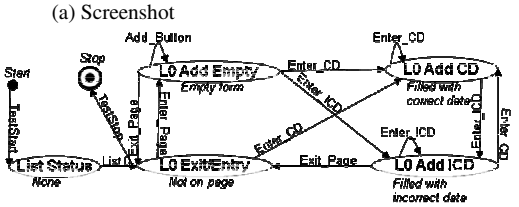
Test Generation. Transition coverage criterion is selected for generation of test sequences. Consequently, the problem of producing minimized test sets satisfying this criterion turns out to be another variant of Chinese Postman Problem. The additional feature of MFSMs is that, using the output function, it is possible to provide extra information such as suspected output in order to aid the test oracle during test execution.



(a) Screenshot



(b) Partial ESG Model



(c) Partial MFSM Model

Figure 3: ISELTA Specials Module – Main Interface and Simplified Models

4 Case Study

ISELTA is an online reservation system for hotel providers and agents. It is a cooperative product of the work between a mid-size travel agency (ISIK Touristik Ltd.) and University of Paderborn. For our case study, we will consider "Specials Module" of ISELTA (Figure 3). Through this module, in the given hotel, one is able to add special prices to the specified number of rooms of certain type for the determined period of time. Consequently, one can edit existing specials and also remove them. In our case study, testing of this module is carried out by two students separately.

Specials module is modeled using ESGs and MFSMs (See Figure 3 for partial models). The ESG model has *3* levels and it is a regular approximation to the system. In order to guarantee a valid approximation, some system-attributed events, e.g., S0 or "there are no existing specials" event in Figure 3(b), are not interpreted in test execution. They are used to differentiate between contexts, like "existing number of specials is *0*, *1*, or *≥2*" or "a special is being edited". On the other hand, MFSM model turns out to be another regular approximation to the system. It has *2* levels and, unlike ESGs, states are used to differentiate between contexts. In addition, thanks to the output function, corresponding test sequences also provide suspected outputs to guide the tester during test execution.

In order to test the system, aforementioned mutation operators are applied to generate mutants and these mutants together with original models are used to generate test sequences to be executed on the system to detect failures. Table 1 gives data on the number of mutants, which are selected differently based on the individual preferences (Numbers with respect to different mutants are not included in order to save space).

Table 1: Number of Mutants

Model	Upper Bound	Selected	Live	Killed	Equivalent
ESG	1507	120	44	76	0
MFSM	30040	126	24	102	8

Test execution results show that the majority of live mutants are dA mutants (for ESGs) and dT mutants (for MFSMs). Since greater number of dA mutants is used for ESGs, fewer mutants were killed. This can be explained by the fact that, in our case, dA or dT mutants are just correct sub models of the system. Thus, based on the used test generation methods, these mutants tend to yield test sequences which are relatively less effective in revealing failures or mutants. Furthermore, it is observed that the upper bound on the number of 1st order MFSM mutants is quite larger than that of ESG mutants. This stems from the fact that application of iT operator results in, as expected, a really great deal of mutants (approximately 96% of all mutants) because the operator includes states, increasing the number of possible operator applications and, in general, the testing effort.

Table 2: Detected Failures

No.	Brief Explanation
1	A special can be inserted with missing start date.
2	A previous date can be inserted for a special.
3	Departure date is not autocorrected.
4	Departure date is falsely autocorrected.
5	Today button does not work.

During test execution, killed mutants are able to detect the failures given in Table 2. Except for failure 5, both ESG-based and MFSM-based approaches manage to reveal the same failures (Failure 5 could not be detected using MFSMs, because today button was not included in the model). The results imply that, despite being simpler in structure, ESGs are able to achieve the same failure/fault-detection capability as MFSMs.

5 Conclusion

In this paper, mutation testing of a web-based system is carried out using an event-based and a state-based approach, i.e., using ESGs and MFSMs, respectively. To do this, regular approximations to the system are created, while taking particular measures to handle the execution of test sequences properly.

The results indicate that both approaches, contrary to the common belief, have similar failure-detection power when appropriately used. The advantage of ESGs is that they focus on only system events. Therefore, in an application with relatively large number of states, ESGs are easier to build and handle. However, direct use of state information is rendered impossible. On the other hand, FSMs include the states explicitly. Thus, they can easily be used to deliver auxiliary information like suspected outputs and provide assistance to the test oracle. Still, due to the inclusion of states, the length of the produced test sequences tend to be longer and mutation operators may easily lead to production of quite a large number of mutants.

In our case study, for both approaches and, naturally, with respect to the selected mutation operators, coverage criteria and test generation methods: (1) Repeated test sequences are observed in the produced test sets, and (2) the mutants produced by the applications of delete arc or delete transition operators are found to be less effective, or even ineffective, in failure/fault detection. Nevertheless, it still seems possible to obtain relatively less redundant test sequences by using higher order or different mutants, and, furthermore, the deletion operators can still prove to be more effective if they are used in combination with different coverage criteria and/or test sequence generation methods. In addition, it is naturally possible that the results may vary for other (type of) systems.

As a final remark, the testing approach used in this paper is a work in progress and being actively studied in our research group.

References

[BBW06] Belli, F.; Budnik, Ch., J.; Wong, E.: Basic Operations for Generating Behavioral Mutants, In: Proc. 2nd Workshop on Mutation Analysis in conjunction with ISSRE'06. IEEE CS, 2006; S. 10-18.

[Be01] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces, Proc. of Int. Symp. on Softw. Reliability and Eng., IEEE Comp. Press, 2001; S. 34-43.

[Bi00] Binder, R.V: Testing Object-Oriented Systems. Addison-Wesley; 2000.

[DLS78] DeMillo R. A.; Lipton R. J.; Sayward F. G.: Hints on Test Data Selection: Help for the Practicing Programmer, Computer, v.11, no. 4, April 1978; S. 34-41.

[EJ73] Edmonds, J.; Johnson, e.L.: Matching: Euler Tours and the Chinese Postman, Math. Programming, 1973; S. 88-124.

[HMU01] Hopcroft, J. E.; Motwani. R.; Ullman, J. D.: Introduction to Automata Theory, Languages and Computation, 2nd Edition, Addison-Wesley, Massachusetts, 2001.

[Mo56] Moore, E. F.: Gedanken-experiments on Sequential Machines. Automata Studies, Annals of Mathematical Studies, Princeton University Press, 1956; 34, S. 129-153.