Mutation Analysis Testing for Finite State Machines

Sandra C. Pinto Ferraz Fabbri - UFSCar/IFSC-USP
Márcio Eduardo Delamaro - IFSC-USP

UFSCar - Universidade Federal de São Carlos
Departamento de Computação
P.B. 676, 13.565-905, Phone (0162)748232
São Carlos, SP, Brazil
e.mail - sfabbri@icmsc.sc.usp.br

José Carlos Maldonado - ICMSC-USP
Paulo Cesar Masiero - ICMSC-USP
ICMSC-USP - Instituto de Ciências Matemáticas de São
Carlos, Universidade de São Paulo
P.B. 668, 13.560-970, Phone (0162)749136
São Carlos, SP, Brazil
e.mail - jcmaldon@icmsc.sc.usp.br

Abstract

The application of the Mutation Analysis criterion in the context of specification based on Finite State Machine is proposed. The main concepts of Finite State Machine and of Mutation Analysis are briefly introduced. An experiment is reported which manually applied Mutation Analysis to a finite state machine modeling a Class 0 ISO Transport Protocol Specification, using two test sequence generator criteria - the W Method and the TT Method. The results obtained are presented and evidences are given that the use of Mutation Analysis is effective in this context. Finally, the lines of evolution of the work presented in this paper are briefly discussed.

1: Introduction

Specification and validation of software are very important software development activities, and Software quality depends on the use of suitable techniques and tools that provide support for these activities. Some of the formal techniques proposed to specify the dynamic aspects of real time systems are Finite State Machines (FSM) [GIL62], Petri Nets [PET81] and Statecharts [HAR87], which all have a visual notation.

Many techniques for analysis, testing and validation have been proposed to assist the software development process [LEV87, PET81, MAS94, CHO78]. Testing is a relevant and extremely expensive software quality assurance activity [HOW87, LIN90], being usually responsible for 50% of the costs of software development. Testing is needed due to our inability to guarantee that all other software development activities have been adequately developed [SOM89]. The main objective of testing is to reveal errors in a system, i.e., to show that the system is incorrect. Thus, a well succeeded test is one that reveals errors and a good test case is one that has a high probability of showing the presence of not yet uncovered errors. It is important to note that there is not a general purpose testing procedure which can be used to prove program correctness.

There are two main research issues in the testing area: how to select test cases and how to assure that a program P was sufficiently tested. The success of the testing activity depends on the quality of a test case set T. In general, there are three techniques to design test cases: functional, structural and error based. These techniques are complementary to each other as they exercise different characteristics of the software being tested [PRE92]; they are also the source of several testing criteria. One way to evaluate the quality of T is to use coverage measures derived from the required elements of a given testing criteria [LIN90].

Mutation Analysis is one of the error-based criteria. It is used to evaluate the quality of a test case set T, and also to generate test case sets [DEM91]. It consists basically of generating mutant programs M, based on a set of mutation operators which are themselves based on common, typical errors made by a programmer. Many testing criteria have been proposed to validate the behavior of systems specified by Finite State Machines [CHO78, GON70, NAI81, SAB88, FUJ91]. In this paper we propose the use of Mutation Analysis for this purpose. We also discuss the adequacy of Mutation Analysis for this task and compare the results obtained with those obtained from the application of other criteria.

The structure of this paper is as follows. In Sections 2 and 3 Finite State Machines and Mutation Analysis, respectively, are briefly introduced. A set of
mutation operators for FSMs is proposed in Section 4. An example illustrating our proposal is presented in Section 5 which also contains an analysis of the results obtained. In the concluding Section we summarize our findings on the use of Mutation Analysis applied to FSM and comment about a tool being developed to support this criterion, as well as the evolution of the present work.

2: Finite State Machines and Test Sequence Generation Methods

According to Fujiwara [FUJ91], a deterministic Finite State Machine (FSM) can be represented by a quintuple (X,E,Y,T,O), where:

X: is a set of inputs, x
E: is a set of states $S_i$, including a special state $S_0$ called the initial state
Y: is a set of outputs, y, including the null output ($\cdot$)
T: transition function, $X \times E \rightarrow E$
O: output function, $X \times E \rightarrow Y$

The notation $S_i \rightarrow x/y \rightarrow S_j$ indicates that the FSM M in state $S_i$ responds to the application of the input x with the generation of an output y and a transition to the state $S_j$. The machine M is completely specified if from each state of M there exists a transition for each input symbol in X. The machine M is strongly connected if for each pair of states ($S_i$, $S_j$) there exists an input sequence which takes M from $S_i$ to $S_j$. The machine M is minimal if the number of states in M is less than or equal to the number of states for any machine $M'$ equivalent to M, i.e., $M'$ responds with identical output sequences to each input sequence.

Considering M a correct machine and $M'$ the machine available, both minimal, with the same input alphabet, Chow [CHO78] classifies the errors in the control structure (called sequencing errors) in three types: operation errors (when $M'$ is not equivalent to M, but can be modified to become equivalent to M by changing only the output function of $M'$); transfer errors (when $M'$ is not equivalent to M, but can be modified to become equivalent to M by changing only the transition function of $M'$); and extra or missing states (when, if in order to make $M'$ equivalent to M the number of states in $M'$ must be reduced or increased; as $M'$ and M are minimal, an unequal number of states implies that $M'$ and M are not equivalent).

Many criteria have been proposed in the literature for FSM test sequences generation, differing essentially with respect to the properties and characteristics required of the machine being tested [FUJ91, NA81, GON70, CHO78, SAB88]. A comparison of the main features of these methods is presented in [FUJ91]. Two of these methods are TT (Transition Tour) and W (Automata Theoretic). The objective of TT is the construction of a test sequence that exercises all transitions and, according to [CHO78], it is not efficient to reveal transfer errors.

The W method is both valid and reliable for control structures modeled by FSM [CHO78]. It requires that the next operation and the next state depend solely on the current state and on the input, i.e., there are no control variables or counters handled by the operations which affect the sequencing of operations. It is also assumed that the machine is completely specified, minimal, starts with a fixed initial state and every state is reachable. This method consists of three main steps: 1) estimation of the maximum number of states in the correct design; 2) generation of test sequences based on the design; and 3) verification of the responses to the test sequences generated in Step 2. The test sequences generated by this method are guaranteed to reveal any error in the control structure as long as the initial assumptions are satisfied. In Section 5, the Mutation Analysis adequacy of test sequences generated by the W and TT methods is explored.

3: Mutation Analysis

Mutation Analysis was created around the year of 1970 at Yale University and Georgia Institute of Technology [DEM78]; it has been shown, by empirical and theoretical work, to be an attractive criterion for program testing [DEM80, BUD80, HOR92]. Recently, with the advances in hardware technology and computer architecture, it is being observed an intensive activity of tool construction to support this criterion aiming at minimizing its application cost [CHO89, KRA88, MAT88].

Mutation Analysis tries to create the confidence that a program P is correct producing, through small syntactic changes, a program set — the mutants — that are similar to P, and creating test cases that are capable of causing behavioral differences between P and each of its mutants. These changes are based on a operator set called mutation operators. To each operator its is associated an error type or an error class that we want to reveal in the program. Mutation Analysis consists of four steps: mutant generation, P execution based on a defined test case set T, mutant execution based on T and adequacy analysis.

The choice and definition of the mutation operators is a key factor for the success of Mutation Analysis. Very simple operators are usually defined based on the competent programmer hypothesis, which states that a program produced by a competent programmer is either correct or near correct. The tester must construct
test cases that show that these transformations lead to incorrect programs. Another hypothesis considered by Mutation Analysis is the coupling effect that, according to DeMillo [DEM78], can be described as: "test data that distinguishes all programs differing from a correct one by only simple errors is so sensitive that it would also implicitly distinguish more complex errors." Some empirical studies have validated this hypothesis [BUD80, ACR79].

All mutants are executed using the input test cases T. If a mutant P_i presents different results from P, it is said to be dead; in this case T identified the error in the mutant or in other words, T revealed the difference between P and P_i. On the other hand, if P_i presents responses identical to P, it is said to be alive. This fact can occur due to two reasons: either there are no test cases in T that are capable to distinguish P_i from P or both P_i and P execute the same function, i.e., they are equivalent. In the first case, new test cases must be inserted in T aiming at killing the mutant. In the latter case there are no test cases capable of killing the mutant because the result will always be the same. Verification of two equivalent programs is, in general, an undecidable problem; this theoretical limitation can be circumvented in some cases. In fact, methods and heuristics have been proposed to determine program equivalence in more restrictive cases [BUD81].

Our objective must be to find a test case set able to kill all non-equivalent mutants; such a test case is considered adequate to test P. Then, either P is correct or contains a subtle, not expected error. This situation should be rare if the mutation operators used to create the mutants are carefully designed.

Mutants generated from k simultaneous changes in the program under test P are called k-order mutants. Previous experiences [BUD80] show that higher-order mutants (k>1) do not contribute significantly to create better test cases and have a high generation and execution cost. Therefore, Mutation Analysis has applied mainly first-order mutants.

DeMillo [DEM80] notes that Mutation Analysis provides an objective measure for the confidence level of the adequacy of the test cases. The Mutation Score, computed from the number of mutants generated, the number of equivalent mutants and the number of mutants killed, allows the evaluation of the adequacy of the test case set used and therefore of the program tested. Given a program P and a test case set T the mutation score ms(P,T) is computed as:

\[ ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)} \]

where:

- DM(P,T): number of mutants killed by the test case set T
- M(P): total number of mutants generated
- EM(P): number of equivalent mutants

It should be noticed that DM(P,T) is the only information dependent of the test cases used. The number of equivalent mutants generated is not known automatically; EM(P) is, in general, obtained interactively as an entry from the tester who plays a relevant role in this decision.

4: Mutation Operators for FSM

To apply Mutation Analysis in the context of FSMs we assume valid the competent programmer hypothesis; in our case it means that a model produced by a designer is either correct or very close to the correct version. Based on the error classes defined by Chow [CHO78] and on heuristics that we have devised about typical errors made by designers during the creation of FSMs, as well as trying to guarantee minimal testing requirements, e.g. covering all transitions, we have created a set of mutation operators for FSMs, namely:

- arc-missing
- wrong-starting-state (default state)
- event-missing
- event-exchanged
- event-extra
- state-extra
- output-exchanged
- output-missing
- output-extra

Based on this set of operators we have applied Mutation Analysis to the FSM depicted in Figure 4.1 which is the design of a Class 0 ISO Transport Protocol [GAB90]. This figure also shows the test sequences generated by the method W [CHO78] and method TT [NAI81], taken from [GAB90]. It should be noticed that this machine is not completely specified, property that is required by method W. Many practical applications do not have the minimum requirements needed to successfully apply test sequence generation methods. We show in Figure 4.2 examples of 1-mutants for some of the designed operators.

W Sequence = (DR, T_Creq,DR, CR,DR, CC,DR, DT,DR, DR,DR, CR, (T_Dreq, T_Cresp, CR, CC, DT, DR), DR, T_Creq, (CC, DT, DR), DR, CR.T_Cresp, (T_Dreq, T_DTreq, CR, DT, DR, N_Dind, N_Rind), DR)

Figure 4.1 - State Machine, State Table, TT and W Sequences.
Figure 4.2 - Example of Mutants Generated by some of the FSM Mutation Operators.

5: Test Sequences Based on FSMs Versus Mutation Analysis

Based on the initial mutation operators set and on the FSM of Figure 4.1, both presented above, Mutation Analysis was manually applied. We present in Table 5.1 a summary of the results obtained from this case study.

If all the restrictions (pre-conditions) required by W were satisfied it would generate a mutation score of 100%, considering only 1-mutants. All sequencing errors of a machine would be revealed if we had a machine satisfying all pre-conditions [CHO78]. Therefore, all mutants would be distinguished from the machine M as they are derived from the error classes considered: operation error, transfer error and missing/extra states.

Although the machine considered in our example does not satisfy W pre-conditions, it obtained a high mutation score as it can be seen in Table 5.1. Two results have been used to distinguish the behavior of the original machine M and its mutants: output produced and sequence of states; in this paper we use the results obtained from output analysis. The sequences generated by W and TT had almost the same degree of adequacy with respect to Mutation Analysis. The general case is not expected to be similar, as sequences generated by TT are not as efficient to reveal certain error classes as are the ones generated by W.
### Table 5.1 - Results of Mutation Analysis Applied to the Example of Figure 4.1.

<table>
<thead>
<tr>
<th>MUTATION OPERATORS</th>
<th>CRITERIA</th>
<th>TT METHOD</th>
<th>W METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>State Seq.</td>
<td>Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Killed</td>
<td>Alive</td>
</tr>
<tr>
<td>OP I - Arc-Missing</td>
<td></td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Total = 9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP II - Wrong-Initial-State</td>
<td></td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Total = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP III - Event-Missing</td>
<td></td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Total = 21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP IV - Event-Exchanged</td>
<td></td>
<td>70</td>
<td>21</td>
</tr>
<tr>
<td>Total = 139</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP V - State-Missing</td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Total = 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP VI - Output-Exchanged</td>
<td></td>
<td>0</td>
<td>193</td>
</tr>
<tr>
<td>Total = 193</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP VII - Output-Missing</td>
<td></td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>Total = 17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP VIII - State-Extra</td>
<td></td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Analyzed = 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP IX - Event-Extra</td>
<td></td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td>Analyzed = 43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equivalent = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Considering that the W method is stronger than TT [CHO78] we discuss in the rest of this section the adequacy of its sequences against Mutation Analysis. The 43 mutants that stay alive — considering the output — are the ones produced by the application of the event-extra operator; this behavior comes from the fact that the FSM is not completely specified. We have noticed that there are many equivalent mutants; their existence is also related to the fact that the FSM does not fulfill all the requirements to the use of W. The event-exchanged operator, for instance, explores the non-completeness of the machine and generates 21 equivalent mutants. Facilities to deal with equivalent mutants will be available in a testing tool being specified.

The high mutation score obtained with our running example, even not satisfying the initial conditions required by W, does not imply that this is a general situation. In real life the machines are more complex, not minimals and with state variables thus weakening the test sequences generated by W from a validation point of view. This would certainly produce a score below 100%. It should be noticed that mutations would be generated based on the state variables. At the program level this is one of the operators that generates most of the mutants. It should also be pointed out that the use of state variables introduces undecidability aspects to the point of FSM equivalence, increasing the cost of Mutation Analysis application.
It is also possible to generate higher order mutants for FSMs, obtaining k-mutants. We have found evidences in the example presented that the coupling effect in general is also valid for FSM-based Mutation Analysis. This is a hypothesis that must be validated for a significant number of models. However, we have identified that by combining the state-extra operator with other mutation operators we have not minimals, equivalent mutant machines and this generates mutants for which the test sequences are not effective, i.e., are not capable of distinguishing among the behavior of M and its mutants. Therefore, investigating 2-mutants in the way pointed out seems to be worthwhile.

Consider a machine \( M_e \) (Figure 5.1a) equivalent to the machine \( M \) created supposedly to facilitate the understanding and the maintenance of machine \( M \), regarding the issue of error messages. The \( M_e \) machine is not minimal as state 1 and Err are equivalent. If we consider now the mutation in \( M_e \) presented in Figure 5.1b the sequences generated by \( W \) do not distinguish most of these mutants e.g. considering the arc-missing operator only 2 mutants out of 5 are killed (a mutation score of 40%).

Let \((S_i,S_j)\) denote the existence of a transition \( t \in T \) for some \( x \in X \) and the set \( X(S_i,S_j) = \{x | x \in S_i \rightarrow x/y \rightarrow S_j, y \in O \} \), indicating the entries that cause a transition from \( S_i \) to \( S_j \). A strategy to generate equivalent, not minimal mutants, with an extra state using the state-extra operator is to generate a mutant for every pair of states \((S_i,S_j)\) such that \( \text{card}(X(S_i,S_j)) > 1 \), as follows: for each \( x \in X(S_i,S_j) \) one state equivalent to \( S_y \) is generated such that \( x \in X(S_y,S_j) \) and \( x \notin X(S_y,S_j) \) in the mutant generated. This is shown in Figure 5.2.

Let \( O(S_i,S_j) = \{y/y \in S_i \rightarrow x/y \rightarrow S_j, x \in X \) and \( y \in O \} \) define the set of all outputs produced taking the transitions from \( S_i \) to \( S_j \). Other strategy to generate equivalent, not minimal, with an extra state mutants is to generate \( \text{card}(O(S_i,S_j)) \) mutants for each pair of states \((S_i,S_j)\) such that \( \text{card}(O(S_i,S_j)) > 1 \). For each \( y \in O(S_i,S_j) \) a \( S_j \) is generated such that \( y \in O(S_i,S_j) \) and \( y \notin O(S_i,S_j) \) in the generated mutant and for each \( x \in X \) \( x/y \rightarrow S_j \) in the machine \( M \), \( x \notin X(S_y,S_j) \) and \( x \in X(S_y,S_j) \) in the mutant machine as is shown in Figure 5.3.

After the generation of these equivalent, not minimal mutants all other mutant operators are applied, thus generating 2-mutants. Consider for example the 2-mutant generated using the arc-missing operator over the FSM mutants generated by applying the latter strategy presented above, in special over the mutant presented in Figure 5.3, on the set of transitions including the extra-state, as depicted in Figure 5.4b. The mutation score would be approximately 25% which is a relatively low score. Another example showing that the test sequence considered in this paper is less effective for 2-mutants is shown in Figure 5.4a which is derived from the mutant of Figure 5.2 after applying the arc-missing operator. The behavior of the original machine is not distinguished by the W test sequence.

Figure 5.1 - (a) A Mutant Generated by the state-extra Operator and (b) A Mutant Generated by the arc-missing Operator over the Mutant (a).
Figure 5.2 - Other Mutant Generated by the state-extra Operator.

Figure 5.3 - A Mutant Generated by the state-extra Operator - Different Events Labeling a Transition that Produces the same Outputs are Grouped Together.

Figure 5.4 - Application of the arc-missing Operator over the state-extra Operator of Figures (a) 5.2 and (b) 5.3.
6: Conclusions

In this paper we proposed the Mutation Analysis criterion in the context of FSM. We explored the use of Mutation Analysis to evaluate the adequacy of the test sequences generated by the W and TT methods for FSMs. The criterion was manually applied to a Class 0 ISO Transport Protocol.

Our main conclusion is that Mutation Analysis can be effectively applied to FSM and can be considered a complementary criterion compared to the test sequence generation methods, as for example the W method [CHO78]. State machines (models) usually do not satisfy the pre-conditions specified by these methods which requires other techniques to improve the confidence that the model is correct. Extensions to FSM like Statecharts would benefit even more of Mutation Analysis as they do not have hitherto developed techniques for test sequence generation and are validated through various types of simulation [HAR90].

The evolution of our work on this subject is directed to three lines of research: improvement of the mutation operators, minimization of the number of mutants generated and the development of a tool to support this criterion. To improve the mutation operator set we are looking at the proposal made by Beizer [BEI90] of a program error taxonomy and trying to do the same at the FSM level; the error classes proposed by Chow [CHO78] are also being considered. To minimize the number of mutants a strategy similar to the one proposed by Acree [ACR79] — verifying how effective are the mutation operators — may be followed. It should be noticed that the generation of certain specific classes of 2-mutants seems to be very effective in the context of FSM.

The cost of applying Mutation Analysis to FSM seems to be comparable to the costs already known at the program level. Use of Mutation Analysis for machines from moderate to great size is almost impractical. Thus, the development of a tool to support the use of Mutation Analysis is mandatory. We have already developed a tool called Proteum [DEL93] to support Mutation Analysis for C programs and we have also developed a graphical editor for FSM and Statecharts with simulation capacities [MAS91]. We are now integrating these two tools in a tool called Proteum/FSM to support Mutation Analysis for FSM. Also, we are extending and exploring these ideas for specification based on Statecharts. Moreover, extending the Proteum/FSM to Statecharts is straightforward as we already have capacity for Statecharts simulation.

References

[BUD80] Budd; T.A.; DeMillo, R.A.; Lipton, R.J.; Sayward, F.G. Theoretical and Empirical Studies on UsingProg Mutation to Test the Functional Correctness of Prog., 7th ACM Symposium on Principles of Programming Languages, jan., 1980.


