Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 2750 – 2767
© TÜBİTAK
doi:10.3906/elk-1909-90

Mutant selection by using Fourier expansion

Savas TAKAN, Tolga AYAV*
Department of Computer Engineering, Faculty of Engineering, İzmir Institute of Technology, İzmir, Turkey

Received: 17.09.2019 • **Accepted/Published Online:** 11.05.2020 • **Final Version:** 25.09.2020

Abstract: Mutation analysis is a widely used technique to evaluate the effectiveness of test cases in both hardware and software testing. The original model is mutated systematically under certain fault assumptions and test cases are checked against the mutants created to see whether the test cases can detect the faults or not. Mutation analysis is usually a computationally intensive task, particularly in finite state machine (FSM) testing due to a possibly huge amount of mutants. Random selection could be a practical reduction method under the assumption that each mutant is identical in terms of the probability of occurrence of its associating fault. The present study proposes a mutant selection method based on Fourier analysis of Boolean functions. Fourier helps to identify the most effective transitions on the output so that the mutants related to those transitions can be selected. Such mutants are considered more important since they are more likely to be killed. To evaluate the method, test cases are generated by the well-known W method, which has the capability of detecting every potential fault. The original and reduced sets of mutants are compared with respect to their importance values. Evaluations show that the mutants selected by the proposed technique are more effective, which reduces the cost of mutation analysis without sacrificing the performance of the mutation analysis.

Key words: Mutation analysis, finite state machine, Fourier transformation, W method

1. Introduction

Testing provides a means to check whether prespecified requirements are met and correct outputs are produced. We benefit from software testing to eliminate the potential faults, to ensure error-free operation of the software [1]. To this end, mostly a large number of test cases need to be generated, thanks to several test strategies.

Formal methods in computer science, particularly in software and hardware engineering, are a type of mathematical technique for specification, development, and verification of software and hardware systems [1]. Formal method-based testing is one of the active research areas in software testing [2, 3]. Formal techniques allow one to prove that the system or its model meets the requirements. There exist various models to express software systems like finite state machines (FSMs), Petri nets, and UML. A FSM is a mathematical model with discrete inputs and outputs. It is mostly used for modeling hardware and software systems [4–6]. There are several methods proposed in the literature to generate test cases for FSMs. The most common ones are transition tour [7], W [8], Wp [9], UIO [10], UIOv [11], DS [12], HSI [13, 14], and H [15]. Among all, the W method has the capability to reveal all potential faults [1] and we benefit from W in the experimental part of this article. The preliminaries of FSM testing are given in Section 3.

Mutation analysis is a useful tool in software testing. The software is systematically mutated under certain fault assumptions. The idea is to make a small change (mutation) in the source code in order to

*Correspondence: tolgaayav@iyte.edu.tr

represent a potential programmer fault. The following shows an example mutation operation:

$$\texttt{if(a<b)\{...\}} \quad \rightsquigarrow \quad \texttt{if(a>b)\{...\}}$$

, where the conditional statement is altered to inject a common programming fault. The modified version of a program is called *mutant*. One can create as many mutants as needed and these mutants are used either in generating test cases or in evaluating a test suite. When it is used to evaluate a test suite, test cases are applied to the mutants to check whether the injected faults can be detected or not [1, 16, 17]. If a test case is able to detect a particular mutant, the mutant is said to be killed by that test case. Mutation analysis is often used to measure the effectiveness of tests for formal models (FSM, Petri net, and UML) [16]. Mutation analysis is powerful, yet it lacks performance due to a possibly large number of mutants produced [17].

Even though mutation analysis is a widely used technique in computer science, the number of studies addressing FSM-based mutation analysis is quite limited [18–24]. These studies mostly discuss mutation operators needed to generate mutants. Despite its many advantages, the analysis requires time and space for large models, which makes the mutation analysis impractical [17].

Mutant reduction is a method to limit the number of mutants so that the analysis can complete in a reasonable time and space. Random selection, as the name implies, selects an arbitrary subset of the mutants. This is quite a practical solution under the assumption that underlying faults are identical in terms of their probability of occurrence. In fact, this assumption is invalid. The aim of the present study is to propose a new mutant reduction method that makes the mutation analysis more efficient. To that end, the important parts of a model are selected to generate mutants. We assume that the parts having a higher influence on the model's output are the parts that are most likely to cause errors.

The method is straightforward. First, the most influential parts of a model need to be identified. This is done by exploiting Fourier analysis of Boolean functions. Then mutants are generated using the mutation operators like reverse of transition, missing transition, and change of input and output. The effectiveness of the resulting set of mutants is calculated. For comparison purposes, we also generate a set of mutants covering the whole FSM. In the evaluations we compare the effectiveness values of these two sets. The results are presented by graphical comparisons and t-tests. The proposed mutant selection method is shown to be able to provide a significant performance improvement.

The organization of the article is as follows: Section 2 provides information about the existing literature. In Section 3, FSM testing is explained. Section 4 gives details of the existing testing method, W. Section 5 introduces the principles of mutation analysis. The proposed method, Fourier analysis-based mutant selection, is detailed in Section 6. Section 7 presents the experimental results. Section 8 provides information about the discussion. The final section contains the conclusion.

2. Related work

Recent literature reviews have shown that the research in mutation testing has been steadily increasing over the last two decades [16, 17, 25–27]. Mutation analysis suffers from the mutant explosion problem, particularly in large FSMs [28–31].

Mathur and Wong proposed methods relying on random selections from within mutants in order to reduce the mutant sets [32]. Later, Zhang developed the so-called double random selection strategy and demonstrated its effectiveness [33]. There also exist several other articles. Derezinska proposed different mutant sampling criteria based on equivalence partitioning in respect to object-oriented program features [34]. In order to alleviate the computational issues, Petrovic presents a diff-based probabilistic approach to mutation analysis [28].

Mathur's study was the first attempting to reduce mutants [35]. Offutt conducted an experimental work in which some mutation operators can be discarded without reducing the effectiveness of mutation tests [36]. Wong and Mathur investigated 22 mutant operators with the Mothra tool and found that some operators had greater effectiveness [37]. Offutt showed that mutation testing can be performed with five mutants without a significant reduction in success [38]. Namin suggested to use statistical methods to select mutation operators [39]. Vincenzi developed a method similar to the incremental techniques in the selection of mutation operators for unit and integration tests [40]. It was shown by Just and Schweiggert that redundant mutants can affect mutation testing in terms of quality and efficiency [41]. They also proposed a method to find redundant mutants. Delamaro found that the most cost-effective mutation operator for C programs could be the statement deletion operator (SDL) [42].

The use of high-order mutation is a relatively new approach. Jia and Harman introduced the concept of higher-order mutants, which are more difficult to kill than first-order mutants [43]. Polo et al. introduced methods for combining first-order mutants with second-order mutants [44]. Papadakis et al. conducted an empirical study in relation to mutant selection [45]. In their study, they focused on comparing first- and second-order mutation techniques. As a result, Papadakis et al. observed that first-order mutation is usually more effective than second-order mutation. Harman et al. [46] focused on the effectiveness of higher-order mutants. The high-order mutant technique significantly reduces the number of mutants. Harman et al. transformed easily killed first-order mutants into a high-order mutant by combining them.

The equivalent mutant problem is also important in mutation testing, because it is costly to find an answer to the question "is a mutant equivalent to a basic program or not?" In his proposed methods, Offutt focused on identifying equivalent mutants based on compiler optimization and data flow analysis [47]. Hierons et al. used the program slicing technique to find equivalent mutants [48]. Yao realized that some mutant operators could produce equal or hard-to-kill mutants, and proposed that mutation operators should be prioritized accordingly [49]. Papadakis focused on mutant classification techniques and observed that high effectiveness in mutant classification may be associated with the use of low quality test suites [50]. Kintis attempted to isolate first-order equivalent mutants using a second-order mutation [51]. Papadakis developed an algorithm called trivial compiler equivalence to find equal mutants [52]. The equivalent mutant problem has been discussed in many ways in the literature. Kintis studied trivial compiler equivalence (TCE) as a simple, fast, and readily applicable technique for identifying equivalent mutants for C programs [53].

FSM-related mutation analysis research is limited [20–24] and these works mostly study the mutation operators that are used to generate mutants. Mutant reduction, however, has been addressed in many studies [16, 17, 25–27]. Nevertheless, there is no study on mutant reduction for FSMs to the best of our knowledge. The aim of the present study is to make the mutation analysis for FSMs more practical and effective by reducing the mutant set. The method we propose exploits Fourier analysis for Boolean functions and relies on selecting the most influential regions of a given FSM and generating mutants for those regions. We assume that the important regions are the parts that are most likely to cause errors. Reducing the size of the model decreases the number of mutants while the fault detection rate per mutant increases. The proposed mutation analysis shows significant performance improvements compared to the standard mutation analyses.

3. Testing finite state machines

A FSM is a mathematical model with discrete inputs and outputs. It is mostly used for modeling hardware and software systems. Examples of software and hardware systems are text editor, compilers, and synchronous sequential circuits. Digital computers can also be seen as systems suitable for this model. Therefore, FSM is quite a common model in computer science and engineering [4–6].

A finite state machine is usually expressed with the tuple M = $\langle$I, O, S, f, g, s_0 $\rangle$, where I is a finite set of input symbols, O is a finite set of output symbols, S is a set of finite states, f: S $\times$ I $\rightarrow$ S is the transition function that determines the next state, g: S $\times$ I $\rightarrow$ O is the output function, and s_0 is the initial state of the machine. Figure 1 shows a 2-input and 1-output FSM with 4 states and 7 transitions. We will use it as the running example in the rest of the article.

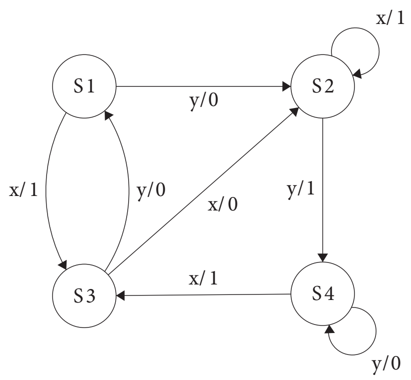

Figure 1. Example FSM.

As a formal modeling technique representing both circuits and software, FSM is widely used in testing. Therefore, there exist several methods for test case generation. Some examples from the literature are W, Wp, UIO, UIOv, DS, HSI, and H. Before introducing the techniques for FSM testing, the following two definitions should be distinguished:

Test case: A set of input values developed for a specific purpose or test condition, such as running a specific program path or verifying its compatibility with a requirement, a set of prerequisites to be run before the test, the expected results, and conditions after the test.

Test suite: A set of test cases that are collected to test a system or component. It is possible that postcondition is for one test case and precondition is for another test case.

On the other hand, the increase in software diversity and size requires better testing tools taking into account several quality metrics like failure detection rate, failure detection capability, test suite size, test case lengths, and time spent in testing.

In the following section, the W method is explained. W produces a test suite that is able to detect all potential faults, in other words, to kill all potential mutants. Therefore, the test suite is expected to be too large. In the present study, we compare the initial set of mutants and the set reduced by Fourier analysis against the test suite generated by the W method.

4. W method

W is a common method to generate test cases for FSMs. It is able to detect all the potential faults [1, 54]. Therefore, we use W to demonstrate the effectiveness of the proposed mutant selection method. Table 1 compares the fault detection capabilities of the common test generation methods. Figure 2 is the comparison of the same methods in terms of the fault detection and test suite lengths.

Table 1. Fault detection capability of the presented methods [54].

	Output faults	Transition faults	Extra state faults	Missing state faults
Transition tour	always	-	-	-
DS-method	always	always	-	always
W-method	always	always	always	always
UIO-method	always	almost	almost	almost

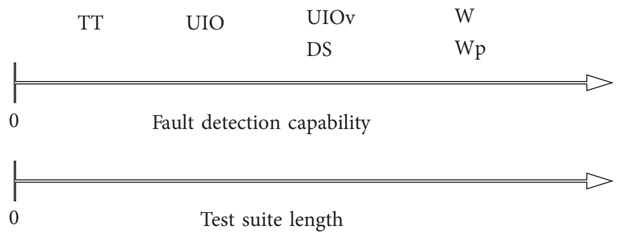

Figure 2. Fault detection capability versus test suite length of the presented methods [54].

The W method consists of two parts. These are the W-set section for testing the next state and the transition coverage set (TCS) for testing the transition (output) between the two states. The transition coverage set is created to visit all transitions. To do this, the tree is obtained from the graph. Paths leading to all ends of this tree are found. The set of input values for these paths creates the transition coverage set. Refer to Chows's study [8] for detailed information.

In the present study, the TCS created from the sample FSM shown in Figure 1 is as follows:

$$\{x, y, xx, xy, yx, yy, yyx, yyy\}$$

The W-set created from this FSM is

$$\{yy, y, x\}.$$

By the W method, the test suite is the product of the TCS and W-set:

$$W\,TestSuite = TCS \times W\,set = \{x, y, xx, xy, yx, yy, yyx, yyy\} \times \{yy, y, x\} =$$
$$\{xyy, xy, xx, yyy, yy, yx, xxyy, xxy, xxx, xyyy, xyy, xyx, yxyy, yxy, yxx,$$
$$yyyy, yyy, yyx, yyxyy, yyxy, yyxx, yyyyy, yyyy, yyyx\}$$

5. Mutation analysis

Mutation analysis is based on the idea of using artificial errors to support testing activities [55]. The mutation is typically used to evaluate the adequacy of test sets, guide the creation of test scenarios, and support experiments [56]. Mutation analysis generates simple syntactic modifications (mutants) in the original program, which represents typical programming errors [57]. For example, a mutation operator replaces an arithmetic operator in the original program with another operator; if a test team can distinguish a mutant from the original program, the mutant is said to have been killed. Otherwise, the mutant is called a living mutant. A mutant can survive since it might be equivalent to the original program or the test case might be insufficient to kill the mutant. The adequacy of a test suite is determined by the ratio of the number of killed mutants to the number of nonequivalent mutants. This ratio is also called the mutation score.

Mutation analysis assumes that small changes in programs are mostly capable of revealing more complicated errors. Mutation analysis was used not only in program code, but also in design models, system properties, and system interfaces.

In FSM testing, mutation analysis is often performed for the assessment of test suites. However, there are few publications in the literature regarding the use of mutation analysis for FSMs. In the case of large FSMs, however, the number of generated mutants becomes too high. This problem is overcome by reducing the mutant set. In the present study, we aim to analyze the regions with the highest error rate in output and state values. Therefore, mutation analysis with a reduced mutant set that contains the important mutants becomes practical to be implemented.

As stated by Fabbri et al. [20], potential mutation operators for FSMs are arc-missing, wrong-starting-state (default state), event-missing, event-exchanged, event-extra, state-extra, output-exchanged, output-missing, and output-extra. According to another article in 2009 [22], mutation operators are reverse of transition (ROT), missing of transition (MOT), redundant of transition (DOT), change of input (COI), missing of input (MOI), change of output (COO), missing of output (MOO), end state changed (ESC), end state redundant (ESR), start state changed (SSC), and start state redundant (SSR). To the best of our knowledge, there is no study on prioritizing mutants for reduction.

6. Mutant selection by using Fourier expansion

The proposed mutant selection method aims to create mutants from important hypothetical parts of a FSM so that the mutation analysis can be faster and more effective with a relatively small set of mutants. These parts of FSMs are selected by Fourier analysis. To this end, the transitions in the model are prioritized with the assumption that the important hypothetical regions are the areas with the highest fault probability.

To evaluate the proposed method, two sets of mutants are formed. In the first set, mutants are generated from the whole model. The second set is a subset of the first one and it includes mutants generated from the important hypothetical transitions selected by Fourier transform. Finally, these two sets are analyzed with a test suite constructed by W method. In Subsection 6.1, first the fundamentals of Fourier analysis are given. The proposed method is detailed in Subsection 6.2, followed by its complexity analysis with Subsection 6.3.

6.1. The fundamentals of Fourier analysis of binary functions

In this section, the basic concepts of Fourier analysis of binary functions are introduced. Although Fourier analysis of binary functions has been an interesting topic in the field of mathematics and engineering over the last decade, its implementations are quite limited. Binary functions are often defined as $f : \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B} = \{\text{True}, \text{False}\}$. As a requirement of Fourier analysis, *True* and *False* are replaced with -1 and 1, respectively. The basic definitions and theorems for the analysis are given below without proof. For other theorems, proofs, and examples, refer to O'Donnell [58] and De Wolf [59].

Theorem 1 (Fourier expansion) *Each function $f : \{-1,1\}^n \to \mathbb{R}$ can be uniquely expressed with the Fourier expansion*

$$f(x) = \textstyle\sum_{S \subseteq [n]} \hat{f}(S) \chi_S ,$$

where $\hat{f}(S)$ represents the Fourier coefficients and $\chi_S = \prod_{i \in S} x_i$ refers to the parity function.

Definition 1 (Inner product) *The inner product is found by the following formula:*

$$< f, g >= \tfrac{1}{2^n} \textstyle\sum_{x \in \{-1,1\}^n} f(x)g(x)$$

Fourier coefficients are calculated as follows:

$$\hat{f}(S) =< f, \chi_S >$$

We show below how the Fourier expansion of a simple function is calculated. Let the function be as follows:

$$f = a \vee b \wedge c$$

It is straightforward to write the following truth vector where F and T denote False and True:

[F F F T T T T T]

The Fourier expansion of the function can be written as follows:

$$f = \hat{f}(\emptyset) + \hat{f}(1)a + \hat{f}(2)b + \hat{f}(3)ab + \hat{f}(4)c + \hat{f}(5)ac + \hat{f}(6)bc + \hat{f}(7)abc$$

According to Definition 1, the first Fourier coefficient $\hat{f}(\emptyset)$ is calculated as follows:

$$\hat{f}(\emptyset) = \frac{1}{2^3}(1 + 1 + 1 - 1 - 1 - 1 - 1 - 1) = -0.25$$

Similarly, the second coefficient is found as follows:

$$\hat{f}(1) = \frac{1}{2^3}(1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 - 1 \cdot 1 - 1 \cdot -1 - 1 \cdot -1 - 1 \cdot -1 - 1 \cdot -1) = 0.75$$

When all coefficients are calculated, Fourier expansion is obtained as follows:

$$f = 0.25 + 0.75a + 0.25b + 0.25ab + 0.25c + 0.25ac - 0.25bc + 0.0abc$$

6.2. The proposed mutant selection algorithm

Since the Fourier transform is defined on binary functions, FSMs need to be represented in binary form. For example, S_1 and S_2 can be encoded as 00 and 01, respectively. Input values x and y are represented as 1 and 0, respectively.

The output function is given with Equation 1. The output function consists of states and inputs.

$$O = (\bar{S_1} \wedge \bar{S_2} \wedge X) \vee (\bar{S_1} \wedge S_2 \wedge X) \vee (\bar{S_1} \wedge S_2 \wedge \bar{X}) \vee (S_1 \wedge S_2 \wedge X) \quad (1)$$

Inputs and states are used to obtain Fourier coefficients. Fourier expansion Equation 2 is given:

$$O = 0.5S_1 + 0.5S_2 + 0.5X + 0.5S_1S_2X \tag{2}$$

Note that in Function 2 some Fourier terms are missing. This means that these components have no effect on the function. Deleting test cases that relate to these components reduces the test suite size and cost of testing. Therefore, the components below a predefined threshold are ignored. The generation and the reduction of the number of test cases are determined by two parameters. The first parameter refers to how many terms to take. The second parameter refers to how many transitions are selected.

By applying the parameters to the Fourier expansion, some components are omitted, and as a result the function O becomes as seen in Equation 3.

$$O = 0.5S_1 + 0.5S_2 + 0.5X + 0.5S_1S_2X \tag{3}$$

As shown in Equation 3, the suitable transitions are chosen after the coefficients are selected. Table 2 shows the suitable transitions after the algorithms are executed.

Table 2. After all coefficients are applied.

S1	S2	X	S1	S2	Output
0	0	0	1	0	0
0	1	0	1	1	1
1	1	1	0	1	1
0	0	1	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0

Two algorithms have been developed to select transitions associated with Fourier terms. In the first algorithm, there are two parameters. The first parameter enables one to select the most important terms of the Fourier expansion. The second parameter defines how many transitions will be taken from each selected term. Depending on these parameters, the number of selected transitions can be adjusted. If the parameters are assigned their highest possible values, all transitions are taken. This makes the proposed analysis the same as normal mutation analysis.

The other algorithm takes Fourier variables as a template. Whether the transition is appropriate is checked from this template. If the pattern is one and there are no changes in values, the program returns false. On the other hand, if the pattern is zero and there is a change in values, the program again returns false. It turns true at the end of the program.

Significant reduction in the number of mutants is achieved only by mutating the selected transitions. Thus, the computational burden of mutation analysis is expected to decrease. One can access the tool developed in this article in the code repository[1].

6.3. Complexity analysis

Binary encoded S states result in m bits such that $S \leq 2^m$. A Fourier expansion consists of 2^m terms and the maximum number of transitions in a FSM can be $S(S-1)$. The complexity of Fourier transformation is

[1]https://github.com/savastakan/mutant_selection

given as $O(2^m)$. Therefore, the complexity of the whole algorithm can be expressed as $O(2^m + 2^m \cdot S(S-1) \cdot m + S(S-1)) = O(S + S \cdot S(S-1) \cdot \log_2 S + S(S-1)) = O(S^3)$.

Complexity is polynomial in terms of the number of states. However, for large systems, the performance of Fourier computation can be enhanced using on-the-fly computation techniques. In the evaluations, the method is verified against FSMs that are up to about 32,350 states. We use a machine having an Intel i7 processor and 16 GB memory. We do not present the computation times in the evaluations since they are insignificant.

7. Evaluations

This section demonstrates the performance of the proposed mutant reduction method. Test cases are generated using the W method. As stated before, W is capable of detecting all potential faults, i.e. killing all mutants. We use the following mutation operators to create the mutants: ROT, MOT, COI, and COO. Test cases are checked against the mutants whether they are killed or not. Mutation analysis is generally benefited from to assess the effectiveness of test cases. Mutant kill ratio, that is the number of killed mutants over the entire mutant set, is a common performance metric. Since the W method provides the most comprehensive test suite for a given FSM, the expected mutant kill ratio is 100%. In fact, our mutant reduction method aims to select the most influential mutants. Those mutants are more likely to be killed by a test suite and they show a better performance in revealing the faults. Therefore, we define a mutant importance metric as given in Definition 2.

Definition 2 (Mutant importance metric) *Let $M = \{m_1, m_2, m_3 ... m_n\}$ be a set of mutants. The performance metric of mutant i is*

$$\mathcal{I}(m_i) = \textit{Number of test cases killing } m_i$$

The performance of the entire mutant set can be expressed as the average importance as defined in Definition 3.

Definition 3 (Total importance metric) *Let $M = \{m_1, m_2, m_3 ... m_n\}$ be a set of mutants. The performance of the mutant set M is*

$$\mathcal{I}(M) = \mathbf{E}_i[\mathcal{I}(m_i)] = \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}(m_i)$$

FSMs are randomly generated by the KISS Generator v0.8 software tool[2], changing the states, output, and input values separately. Each FSM is produced 5 times and the mean values of outcomes are calculated. This procedure is repeated 300 times, which results in 1500 FSMs in total. According to the existing similar studies, this amount seems sufficient and cannot be considered a serious threat to validity. Similarly, random generation of values eliminates another validation threat. FSMs are *reduced*, *deterministic*, *fully correlated*, and *fully specified* as required by the W method.

In the evaluations, $\mathcal{I}(M)$ is measured against three different FSM parameters: i) number of states, ii) number of inputs, and iii) number of outputs. We compare the original set of mutants and its subset selected by the proposed Fourier method. Let M_0 denote the original set of mutants and $M_f \subset M_0$ denote the set of mutants selected by the Fourier method. Each figure compares $\mathcal{I}(M_0)$ and $\mathcal{I}(M_f)$ against the number of states, inputs, and outputs separately.

[2]https://ddd.fit.cvut.cz/prj/Circ_Gen/index.php?page=kiss

Figure 3 shows the comparisons for one mutation operator, ROT. According to the leftmost plot, $\mathcal{I}(M_f)$ is slightly better than $\mathcal{I}(M_0)$. According to Levene's test for equality of variances, the variances are homogeneous (sig.: 0.055883), yet the t-test for equality of means produces a significant difference between them (sig. (2-tailed): 0.012760). In the middle plot of Figure 3, the proposed reduction method shows a significant advantage. The t-test for equality of means reveals a difference between them (sig. (2-tailed): 0.000498). The rightmost plot similarly proves the success of the proposed mutant reduction method. According to Levene's test for equality of variances, the variances are not homogeneous (sig.: 0.03908). According to the t-test for equality of means, there is a difference between $\mathcal{I}(M_f)$ (sig. (2-tailed): 0) and $\mathcal{I}(M_0)$. In conclusion, there seem to be significant differences between $\mathcal{I}(M_0)$ and $\mathcal{I}(M_f)$, which proves that the proposed mutant reduction method can successfully select the appropriate mutants.

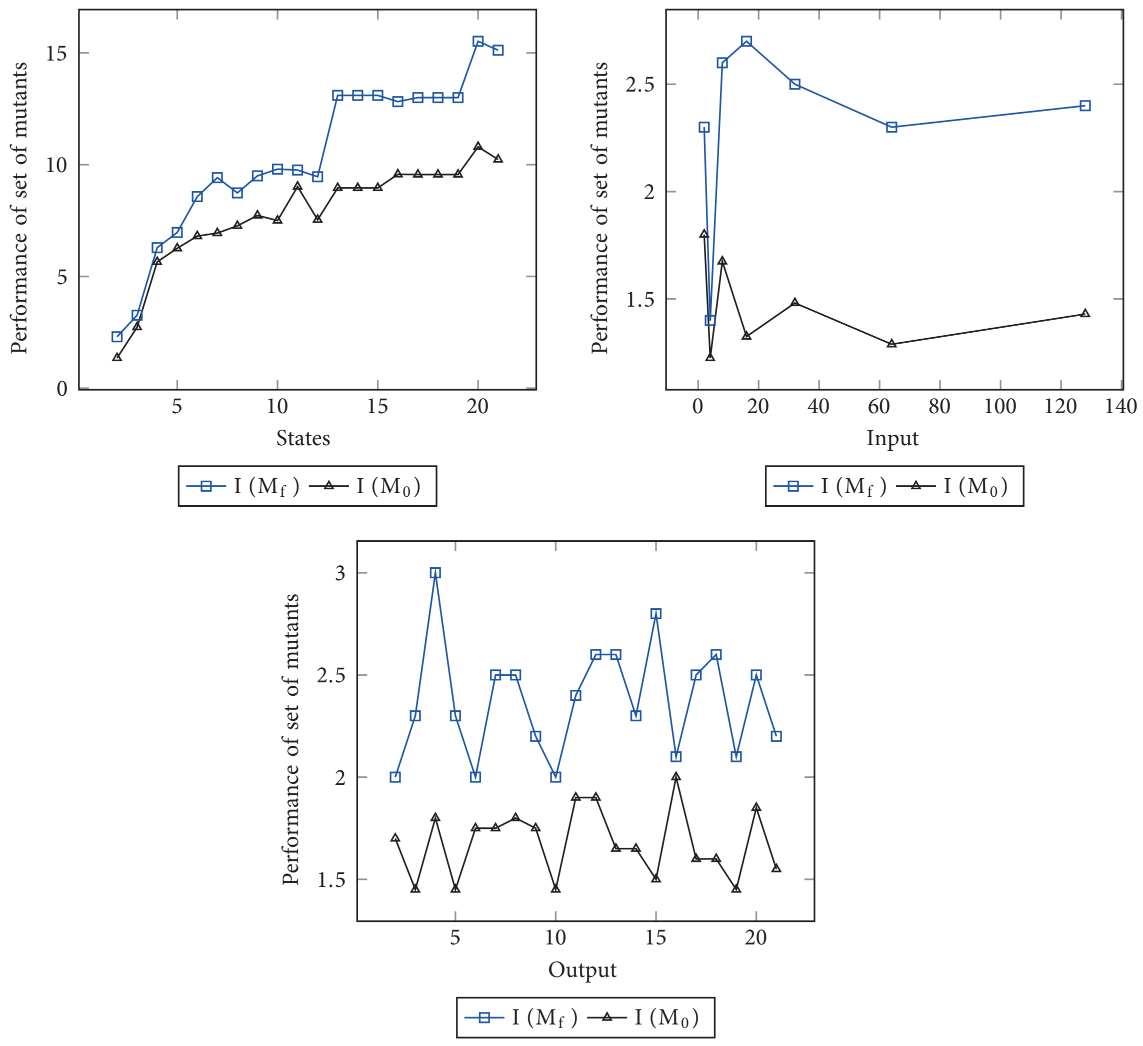

Figure 3. Comparisons for reverse of transition (ROT) operator.

The second comparison is performed for MOT. In the leftmost plot of Figure 4, $\mathcal{I}(M_f)$ and $\mathcal{I}(M_0)$ look quite similar. According to Levene's test for equality of variances, the variances are not homogeneous (sig.: 0.362126). According to the t-test for equality of means, there is no significant difference (sig. (2-tailed): 0.16931). In the second plot, $\mathcal{I}(M_f)$ is significantly higher than $\mathcal{I}(M_0)$. This is proved by the statistical tests. Levene's test shows that the variances are not homogeneous (sig.: 0.002164) and the t-test concludes that there is a significant difference between the means of $\mathcal{I}(M_f)$ and $\mathcal{I}(M_0)$ (sig. (2-tailed): 0.002564) and GMA. Hence, the proposed mutant reduction method is shown to be distinctively advantageous. In the rightmost plot of Figure 4, the variances are not homogeneous (sig.: 0.000007) and the t-test for equality of means concludes that there is a difference between $\mathcal{I}(M_f)$ (sig. (2-tailed): 0.000862) and $\mathcal{I}(M_0)$.

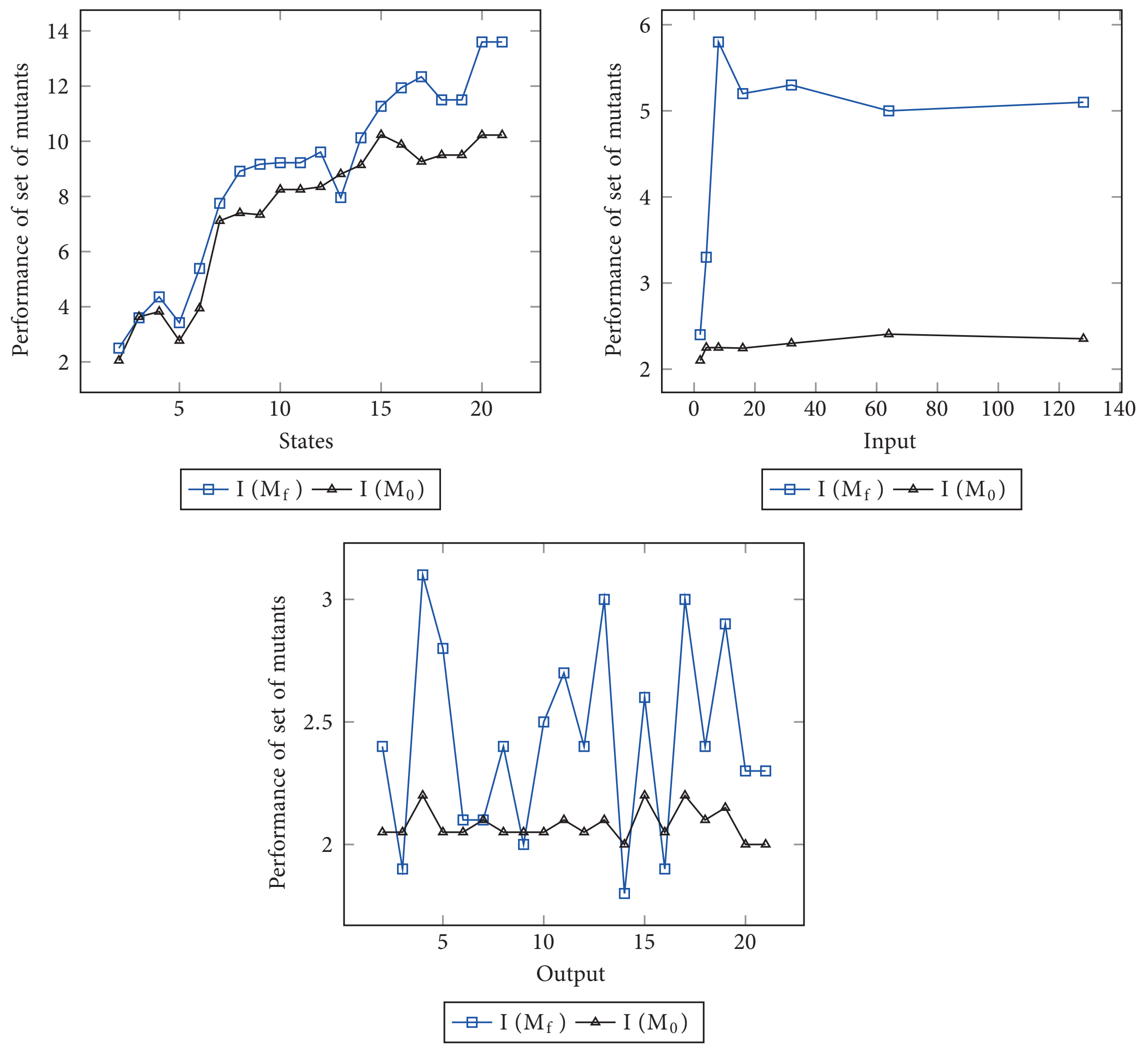

Figure 4. Comparisons for missing of transition (MOT) operator.

Figure 5 presents the evaluations for the mutation operator *COI*. In the leftmost plot, there is no significant difference as proved by Levene's test for equality of variances and t-test. The variances are homogeneous (sig.: 0.17891) and the means are equal (sig. (2-tailed): 0.127798). In the second plot, $\mathcal{I}(M_f)$ is higher than $\mathcal{I}(M_0)$. According to the test results, the variances are homogeneous (sig.: 0.074305) and the means are not identical (sig. (2-tailed): 0.048076). The last plot of Figure 5 does not exhibit a significant difference. According to Levene's test for equality of variances, the variances are homogeneous (sig.: 0.945665). According to the t-test for equality of means, there is no difference between the means (sig. (2-tailed): 1). Hence, we can conclude that there is not a sufficient difference for this type of mutation operator.

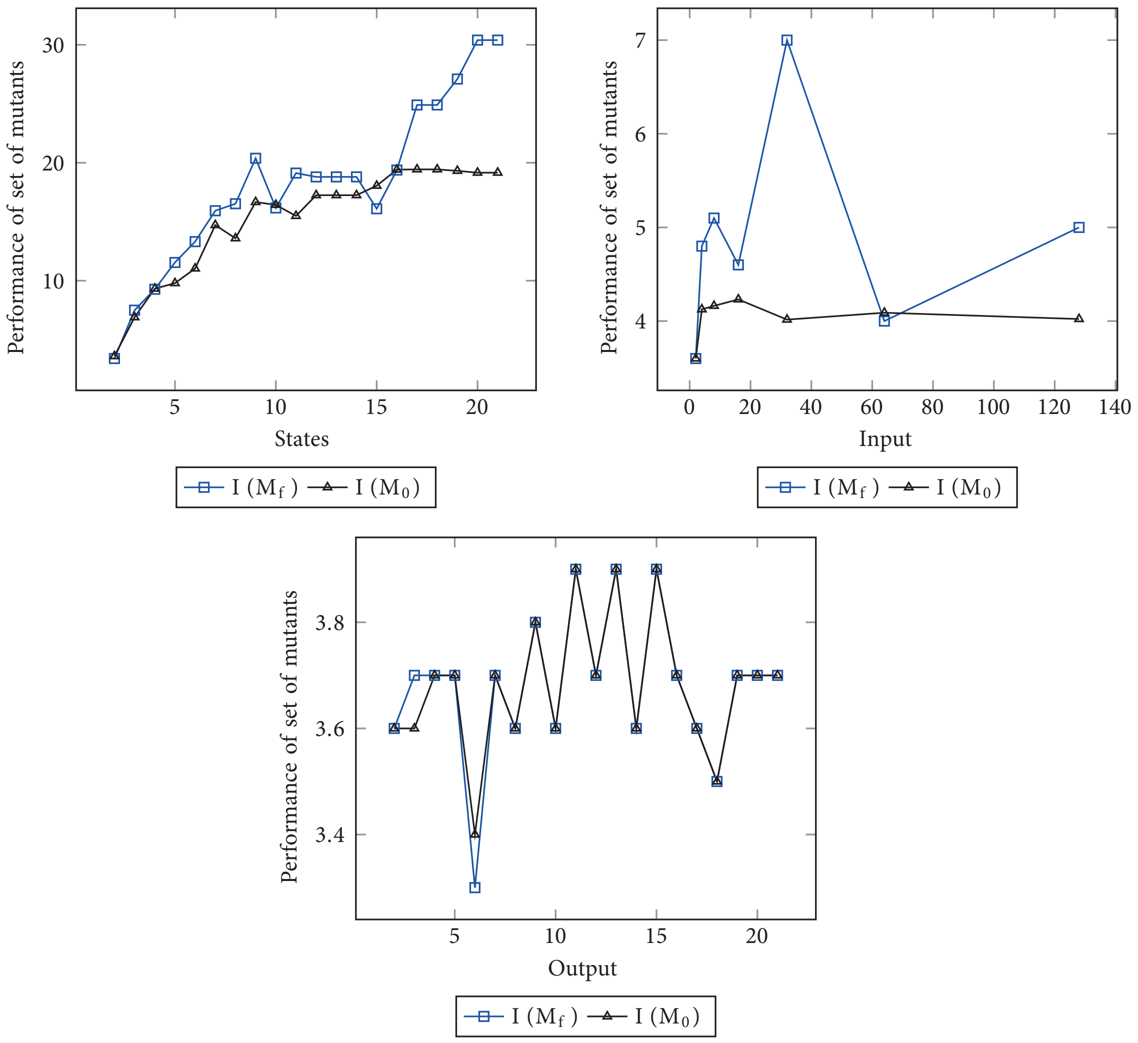

Figure 5. Comparisons for change of input (COI) operator.

Figure 6 shows the results for the mutation operator *COO*. In the leftmost plot, $\mathcal{I}(M_f)$ seems slightly better than $\mathcal{I}(M_0)$ but the tests did not find a meaningful difference. The variances are homogeneous (sig.: 0.077204) and means are equal (sig. (2-tailed): 0.035075). The second and third plots, however, exhibit a meaningful difference. For the second plot, Levene's test for equality of variances reports that the variances are not homogeneous (sig.: 0.004758) and the t-test for equality of means reports that there is a significant difference between $\mathcal{I}(M_f)$ and $\mathcal{I}(M_0)$ (sig. (2-tailed): 0.014156). For the rightmost plot, the variances are not homogeneous (sig.: 0) and the means are not equal, i.e. $\mathcal{I}(M_f)$ is significantly higher than $\mathcal{I}(M_0)$ (sig. (2-tailed): 0.000192).

The last figure aims to repeat the same evaluation for all types of mutation operators, ROT, MOT, COI, and COO. In the leftmost plot of Figure 7, $\mathcal{I}(M_f)$ seems slightly higher than $\mathcal{I}(M_0)$. In the other plots, the

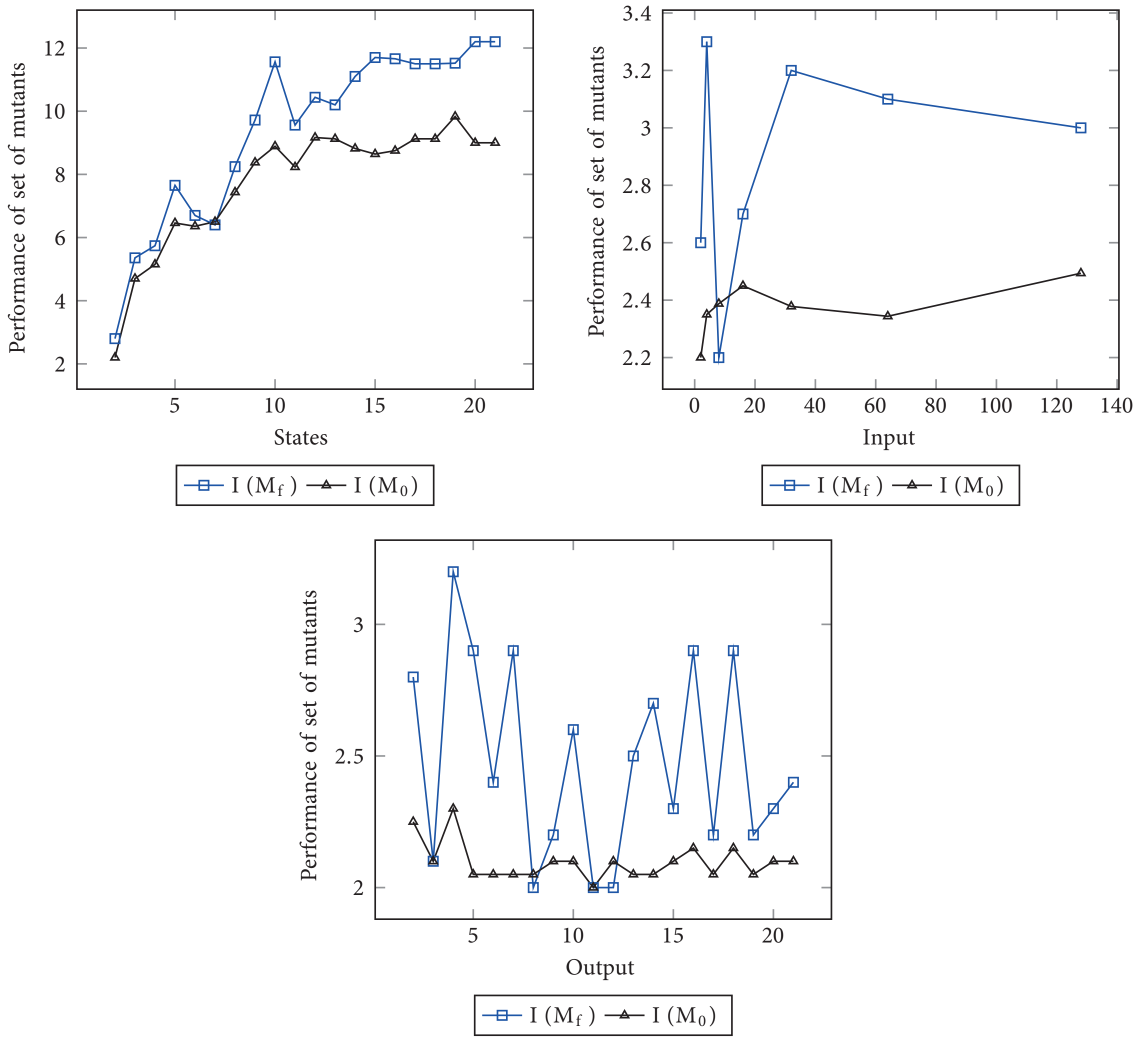

Figure 6. Comparisons for change of output (COO) operator.

proposed technique shows quite a better performance. The results of the presented evaluations so far allow us to prove that the proposed method provides a significant improvement in mutant reduction.

8. Discussion

FSM-based mutant reduction research is quite limited in the literature. In addition, the existing studies are rather focused on mutation operators. Alternatively, in the present study, a new FSM-based mutation reduction method was developed. In the evaluations, large amounts of FSMs were produced in order to achieve more stable results. Compared to similar studies, it can be inferred that the amount of generated mutants is sufficient to make a conclusion about the success of the proposed method. The performance of the proposed mutant reduction method was verified by the comparisons. In the study, t-test and graphical comparison were preferred to demonstrate the significance of performance. According to the results of the assessment, it is possible to conclude that the proposed mutant reduction method plays an active role in reducing the amount of mutants produced by a FSM, without a significant reduction in the ability to catch errors in the FSM.

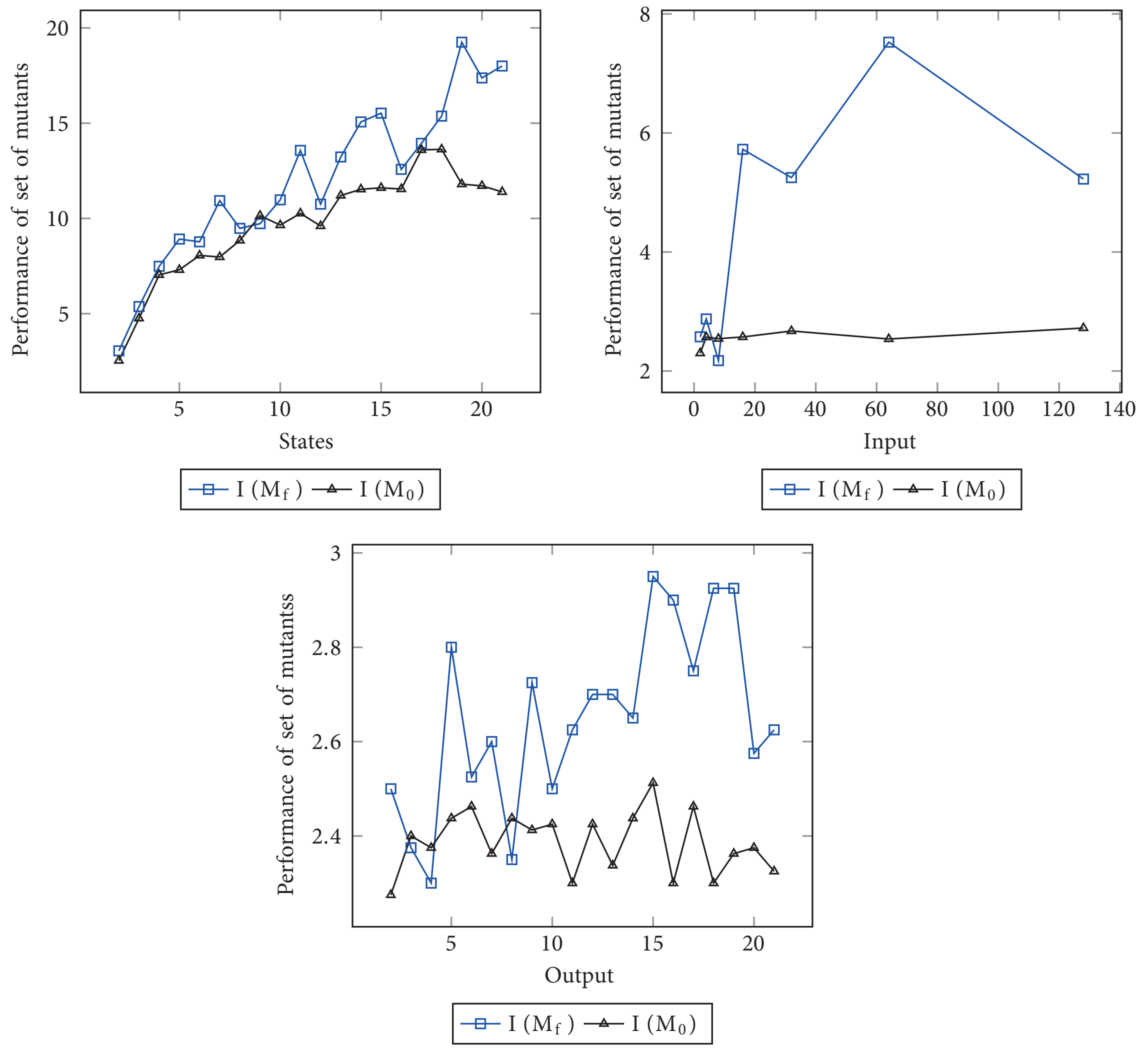

Figure 7. Comparisons for all mutation operators, ROT, MOT, COI, and COO.

The Fourier-based mutant reduction method has two main parameters. These parameters are selected intuitively. It is also possible to find more suitable values by further researching these parameters in subsequent studies. In addition, the lack of much work in the literature on mutant reduction in FSMs allows this issue to be developed with different approaches. Fourier expansion is used in the mutant reduction method but it can also be used in different studies in the software field.

9. Conclusion

This article developed a method for mutant selection as a part of mutation analysis, a widely used technique in circuit and software testing. The reason behind this is that mutation analysis frequently involves large amounts of mutants, particularly in FSM testing. To this end, we defined the FSM parts that may potentially expose more faults by exploiting Fourier analysis of Boolean functions. Mutants are then generated for those parts only. By doing so, the mutant set is reduced while the selected mutants have more potential to reveal the faults. As demonstrated by the evaluations, the proposed method of reduction performs better than a possible random selection. As one of the recent research topics, Fourier analysis of Boolean functions offers many advantages since it is entirely based on mathematics and is quite powerful and also practical to implement. The method presented in this paper provides useful insight about how Fourier analysis can be applied to software testing and related problems.

References

[1] Mathur AP. Foundations of Software Testing, 2/e. India: Pearson Education, 2013.

[2] Lee D, Yannakakis M. Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE. 1996; 84 (8): 1090-1123.

[3] Endo AT, Simao A. Evaluating test suite characteristics, cost, and effectiveness of FSM-based testing methods. Information and Software Technology 2013; 55 (6): 1045-1062.

[4] Belli F, Beyazıt M, Endo AT, Mathur A, Simao A. Fault domain-based testing in imperfect situations: a heuristic approach and case studies. Software Quality Journal 2015; 23 (3): 423-452.

[5] Fragal VH, Simao A, Mousavi MR, Turker UC. Extending hsi test generation method for software product lines. The Computer Journal 2019; 62 (1): 109-129.

[6] Damasceno CDN, Masiero PC, Simao A. Evaluating test characteristics and effectiveness of fsm-based testing methods on rbac systems. In: Proceedings of the 30th Brazilian Symposium on Software Engineering; Rio de Janerio, Brazil; 2016. pp. 83-92.

[7] Naito S. Fault detection for sequential machines by transition tours. In: Processing of 11th IEEE International Sympiosium on Fault Tolerant Computing (FTCS-11); New York, NY, USA; 1981. pp. 238-243.

[8] Chow TS. Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering 1978; 4 (3): 178-187.

[9] Fujiwara S, Bochmann Gv, Khendek F, Amalou M, Ghedamsi A. Test selection based on finite state models. IEEE Transactions on Software Engineering 1991; 17 (6): 591-603.

[10] Sabnani K, Dahbura A. A protocol test generation procedure. Computer Networks and ISDN Systems 1988; 15 (4): 285-297.

[11] Vuong ST. The UIOv-method for protocol test sequence generation. In: Processing of 2nd IFIP International Workshop on Protocol Test Systems (IWPTS'89); Berlin, Germany; 1989. pp. 161-175.

[12] Gonenc G. A method for the design of fault detection experiments. IEEE Transactions on Computers 1970; 100 (6): 551-558.

[13] Petrenko A, Yevtushenko N. Testing from partial deterministic FSM specifications. IEEE Transactions on Computers 2005; 54 (9): 1154-1165.

[14] Petrenko A. Nondeterministic state machine in protocol conformance testing. In: Processing of the 6th International Workshop on Protocol Test systems (IWPTS); Pau, France; 1993. pp. 363-378.

[15] Dorofeeva M, Koufareva I. Novel modification of the W-method. Bulletin of the Novosibirsk Computing Center Series: Computer Science 2002; 1 (18): 69-80.

[16] Papadakis M, Kintis M, Zhang J, Jia Y, Le Traon Y et al. Mutation testing advances: An analysis and survey. In: Memon AM (editor). Advances in Computers. Amsterdam, Netherlands: Elsevier Science Press, 2019, pp. 275-378.

[17] Pizzoleto AV, Ferrari FC, Offutt J, Fernandes L, Ribeiro M. A systematic literature review of techniques and metrics to reduce the cost of mutation testing. Journal of Systems and Software 2019; 157: 110388.

[18] Fabbri SCPF, Maldonado JC, Delamaro M. Proteum/FSM: A tool to support finite state machine validation based on mutation testing. In: Processing of the 19th International Conference of the Chilean Computer Science Society IEEE (SCCC'99); Talca, Chile; 1999. pp. 96-104.

[19] Maldonado JC, Delamaro ME, Fabbri SC, Da Silva Simao A, Sugeta T et al. Proteum: A family of tools to support specification and program testing based on mutation. In: Wong WE (editor). Mutation Testing for the New Century. Boston, MA, USA: Springer, 2001, pp. 113-116.

[20] Fabbri SPF, Delamaro ME, Maldonado JC, Masiero PC. Mutation analysis testing for finite state machines. In: Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering; Monterey, CA, USA; 1994. pp. 220-229.

[21] Da Silva Simao A, Ambrosio AM, Fabbri SCPF, Do Amaral ASMS, Martins E et al. Plavis/FSM: an environment to integrate FSM-based testing tools. In: Processing of Tool Session of 19th Brazilian Symposium on Software Engineering; Uberlândia, MG, Brazil; 2005. pp. 1-6.

[22] Li Jh, Dai Gx, Li Hh. Mutation analysis for testing finite state machines. In: Proceedings of The 2nd IEEE International Symposium on Electronic Commerce and Security (ISECS); Nanchang City, China; 2009. pp. 620-624.

[23] Petrenko A, Timo ON, Ramesh S. Multiple mutation testing from FSM. In: Proceedings of the International Conference on Formal Techniques for Distributed Objects, Components, and Systems; Heraklion, Greece; 2016. pp. 222-238.

[24] Timo ON, Petrenko A, Ramesh S. Multiple mutation testing from finite state machines with symbolic inputs. In: Proceedings of the 19th IFIP International Conference on Testing Software and Systems; St. Petersburg, Russia; 2017. pp. 108-125.

[25] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 2010; 37 (5): 649-678.

[26] Silva RA, De Souza SdRS, De Souza PSL. A systematic review on search based mutation testing. Information and Software Technology 2017; 81: 19-35.

[27] Ferrari FC, Pizzoleto AV, Offutt J. A systematic review of cost reduction techniques for mutation testing: Preliminary results. In: Proceedings of the IEEE 11th International Conference on Software Testing, Verification and Validation Workshops (ICST); Vasteras, Sweden; 2018. pp. 1-10.

[28] Petrovic G, Ivankovic M. State of mutation testing at Google. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice; Gothenburg, Sweden; 2018. pp. 163-171.

[29] Zhu Q, Panichella A, Zaidman A. An investigation of compression techniques to speed up mutation testing. In: Proceedings of the IEEE 11th International Conference on Software Testing, Verification and Validation (ICST); Vasteras, Sweden; 2018. pp. 274-284.

[30] McMinn P, Wright CJ, McCurdy CJ, Kapfhammer GM. Automatic detection and removal of ineffective mutants for the mutation analysis of relational database schemas. IEEE Transactions on Software Engineering 2017; 45 (5): 427-463.

[31] Acree Jr AT. On mutation. PhD, Georgia Institute of Technology, North Avenue, Atlanta, USA, 1980.

[32] Mathur AP, Wong WE. An empirical comparison of data ow and mutation-based test adequacy criteria. Software Testing, Verification and Reliability 1994; 4 (1): 9-31.

[33] Zhang L, Hou SS, Hu JJ, Xie T, Mei H. Is operator-based mutant selection superior to random mutant selection? In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering; Cape Town, South Africa; 2010. pp. 435-444.

[34] Derezinska A, Rudnik M. Evaluation of mutant sampling criteria in object-oriented mutation testing. In: Proceedings of the 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE; Prague, Czech Republic; 2017. pp. 1315-1324.

[35] Mathur AP. Performance, effectiveness, and reliability issues in software testing. In: Proceedings of the 15th Annual International Computer Software & Applications Conference, IEEE; Tokio, Japan; 1991. pp. 604-605.

[36] Offutt AJ, Rothermel G, Zapf C. An experimental evaluation of selective mutation. In: Proceedings of the 15th International Conference on Software Engineering, IEEE; Baltimore, MD, USA; 1993. pp. 100-107.

[37] Wong WE, Mathur AP. Reducing the cost of mutation testing: An empirical study. Journal of Systems and Software 1995; 31 (3): 185-196.

[38] Offutt AJ, Lee A, Rothermel G, Untch RH, Zapf C. An experimental determination of sufficient mutant operators. ACM Transactions on Software Engineering and Methodology (TOSEM) 1996; 5 (2): 99-118.

[39] Namin AS, Andrews J, Murdoch D. Sufficient mutation operators for measuring test effectiveness. In: Proceedings of the ACM/IEEE 30th International Conference on Software Engineering. IEEE; Leipzig, Germany; 2008. pp. 351-360.

[40] Vincenzi AMR, Maldonado JC, Barbosa EF, Delamaro ME. Unit and integration testing strategies for C programs using mutation. Software Testing, Verification and Reliability 2001; 11 (4): 249-268.

[41] Just R, Schweiggert F. Higher accuracy and lower run time: efficient mutation analysis using non-redundant mutation operators. Software Testing, Verification and Reliability 2015; 25 (5-7): 490-507.

[42] Delamaro ME, Deng L, Durelli VHS, Li N, Offutt J. Experimental evaluation of SDL and one-op mutation for C. In: Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation; Ohio, USA; 2014. pp. 203-212.

[43] Jia Y, Harman M. Constructing subtle faults using higher order mutation testing. In: Proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation; Beijing, China; 2008. pp. 249-258.

[44] Polo M, Piattini M, Garcia-Rodriguez I. Decreasing the cost of mutation testing with second-order mutants. Software Testing, Verification and Reliability 2009; 19 (2): 111-131.

[45] Papadakis M, Malevris N. An empirical evaluation of the first and second order mutation testing strategies. In: Proceedings of the 3th IEEE International Conference on Software Testing, Verification, and Validation Workshops; Paris, France; 2010. pp. 90-99.

[46] Harman M, Jia Y, Reales Mateo P, Polo M. Angels and monsters: An empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation. In: Proceedings of the 29th ACM/IEEE international Conference on Automated Software Engineering; Vasteras, Sweden; 2014. pp. 397-408.

[47] Offutt AJ. A practical system for mutation testing: Help for the common programmer. In: Proceedings of the International Test Conference, IEEE; Washington, DC, USA; 1994. pp. 824-830.

[48] Hierons R, Harman M, Danicic S. Using program slicing to assist in the detection of equivalent mutants. Software Testing, Verification and Reliability 1999; 9 (4): 233-262.

[49] Yao X, Harman M, Jia Y. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In: Proceedings of the 36th International Conference on Software Engineering; Hyderabad, India; 2014. pp. 919-930.

[50] Papadakis M, Delamaro M, Le Traon Y. Mitigating the effects of equivalent mutants with mutant classification strategies. Science of Computer Programming 2014; 95: 298-319.

[51] Kintis M, Papadakis M, Malevris N. Employing second-order mutation for isolating first-order equivalent mutants. Software Testing, Verification and Reliability 2015; 25 (5-7): 508-535.

[52] Papadakis M, Jia Y, Harman M, Le Traon Y. Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In: Proceedings of the 37th IEEE International Conference on Software Engineering; Florence, Italy; 2015. pp. 936-946.

[53] Kintis M, Papadakis M, Jia Y, Malevris N, Le Traon Y et al. Detecting trivial mutant equivalences via compiler optimisations. IEEE Transactions on Software Engineering 2017; 44 (4): 308-333.

[54] König H. Protocol Engineering. Heidelberg, Germany: Springer, 2012.

[55] Papadakis M, Just R. Special issue on Mutation Testing. Information and Software Technology 2017; 81:1-2.

[56] Zhang J, Zhang L, Harman M, Hao D, Jia Y et al. Predictive mutation testing. IEEE Transactions on Software Engineering 2018; 45 (9): 898-918.

[57] Aichernig BK, Tappler M. Learning from faults: Mutation testing in active automata learning. In: Proceedings of the NASA Formal Methods Symposium; Moffett Field, CA, USA; 2017. pp. 19-34.

[58] O'Donnell R. Analysis of boolean functions. Cambridge, UK: Cambridge University Press, 2014.

[59] De Wolf R. A brief introduction to Fourier analysis on the Boolean cube. Theory of Computing 2008; 1-20.