

Available from: http://doi.acm.org/10.1145/2245276.2231963


This is the author’s version of the work, posted here with the permission of the publisher for your personal use. No further distribution is permitted. You may also be able to access the published version from your library. The definitive version is available at http://dl.acm.org/.
Fault Class Prioritization in Boolean Expressions

Ziyuan Wang\textsuperscript{1,2}  Zhenyu Chen\textsuperscript{1}  Tsong-Yueh Chen\textsuperscript{1}  Baowen Xu\textsuperscript{1,2}
\textsuperscript{1}State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
\textsuperscript{2}Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China
\textsuperscript{3}Centre for Software Analysis and Testing, Swinburne University of Technology, Victoria 3122, Australia
wangziyuan@nju.edu.cn, zychen@software.nju.edu.cn
tychen@groupwise.swin.edu.au, bwxu@nju.edu.cn

\section*{ABSTRACT}
A recent study has classified faults in Boolean expressions into ten classes and has proved that there are five key fault classes, namely \textit{CCF}, \textit{CDF}, \textit{ORF}, \textit{ENF} and \textit{ASF}, such that if a test suite can kill all faulty versions of these five core fault classes, it can kill all faulty versions of all fault classes. In order to generate more effective test suites, we should prioritize these five fault classes further, such that test cases with stronger fault detection capability could be generated as early as possible. Such a process is referred to as the fault class prioritization. Based on the observation in the fault class hierarchy, we divide the five fault classes into two groups \{\textit{CCF}, \textit{CDF}\} and \{\textit{ORF}, \textit{ENF}, \textit{ASF}\}. Two strategies of fault class prioritization are proposed to generate test cases efficiently. We design experiments using TCAS Boolean expressions and some randomly generated Boolean expressions. The experimental results suggest that if we generate test cases for \textit{CCF} and \textit{CDF} firstly, the final test suite always have a higher efficiency of killing faults.

Categories and Subject Descriptors
D.2.5 \textit{Software Engineering}: Testing and Debugging

General Terms
Algorithms, Experimentation, Reliability

Keywords
Boolean expression, fault-based testing, fault class hierarchy, fault class prioritization

1. INTRODUCTION
Software quality assurance is critical to the successful development and maintenance of software systems. There exist many approaches to improve software quality, including testing, inspection, and formal verification, etc. Software testing is a major means of software quality assurance in real-world software development life circles.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

\textit{SAC'12} March 26-30, 2012, Riva del Garda, Italy.
Copyright 2011 ACM 978-1-4503-0853-0/12/03 ...$10.00.
example, the fault class hierarchy [2] helps us to improve fault-based testing. We only need to generate test cases for the five core fault classes, \(CCF, CDF, ORF, ENF\) and \(ASF\), and then to merge these five test suites. The final test suite guarantees the detection of all possible faults. A new challenge is how to identify fault class’s priority, namely fault class prioritization, such that we should first generate test cases for fault classes of higher possibility to kill more faulty versions. Based on the observation in the fault class hierarchy, it is natural to divide the five fault classes into two groups \(\{CCF, CDF\}\) and \(\{ORF, ENF, ASF\}\). Two strategies of fault class prioritization are proposed to generate test cases efficiently. We design experiments on TCAS Boolean expressions and some randomly generated Boolean expressions. The experimental results suggest that if we generate test cases for \(\{CCF, CDF\}\) firstly, the final test suite always has high efficiency to detect faults.

The rest of paper is organized as follows. Section 2 introduces fault class hierarchy and proposes two strategies of fault class prioritization. Section 3 gives a detailed description of experiments to evaluate two strategies empirically. Section 4 discusses our empirical study and its implication of experiments to evaluate two strategies empirically. Section 5 presents the conclusion.

2. FAULT CLASS PRIORITIZATION

A Boolean expression is a string which involves some Boolean variables \(x_i (i \geq 1)\), the logical operators \(\neg\) (NOT), \(\land\) (AND), \(\lor\) (OR), and the brackets ‘(‘, ‘)‘. Some backgrounds about fault class hierarchy are introduced in the following subsection.

2.1 Fault Class Hierarchy

Ten fault classes used in [2, 4] are defined as follows.

- Operator Reference Fault (ORF). An occurrence of a logical connective \(\land\) replaced by \(\lor\) or vice versa.
- Expression Negation Fault (ENF). A sub-expression (except conditions) is replaced by its negation.
- Variable Negation Fault (VNF). An occurrence of a condition is replaced by its negation.
- Associative Shift Fault (ASF). \(ASF\) is caused by omission of the brackets.
- Missing Variable Fault (MVF). An occurrence of a condition is omitted in the expression.
- Variable Reference Fault (VRF). An occurrence of a condition is replaced by another possible condition. A condition is said to be possible if its variable has already appeared in the expression.
- Clause Conjunction Fault (CCF). An occurrence of condition \(c\) is replaced by \(c \land c’\), where \(c’\) is a possible condition.
- Clause Disjunction Fault (CDF). An occurrence of condition \(c\) is replaced by \(c \lor c’\), where \(c’\) is a possible condition.
- Stuck-At-0 Fault (SA0). An occurrence of a condition is replaced by 0 in the expression.

2.2 Prioritization Strategies

Based on the fault class hierarchy in Figure 1, the five core fault classes could be divided into two groups: (1) the first group, \(\{CCF, CDF\}\), is co-stronger than five fault classes including \(MVF, VNF, SA0, SA1, \) and \(VRF\); (2) the second group, \(\{ORF, ENF, ASF\}\), does not have any relationships with other seven fault classes. Following the division, two fault class prioritization strategies are proposed as follows.

Given a Boolean expression, we generate all possible faulty versions in \(CCF\) and \(CDF\). For each faulty version, we find all test cases that could kill it. All these test cases for each faulty version are collected into a set \(T_a\). Similarly, a set \(T_b\) is generated for \(ORF, ENF\) and \(ASF\).

Strategy 1: \(CCF\) and \(CDF\) have higher priority.

1. Select test cases from \(T_a\) to kill all faulty versions in \(CCF\) and \(CDF\), and to generate a First-Round Test Suite. Here we could utilize some existing test case selection and reduction algorithms, including greedy algorithm, Harrold’s [13] and Chen’s [14] algorithms, etc.
2. Pick up all faulty versions in \(ORF, ENF\) and \(ASF\), which have not been killed by the First-Round Test Suite. Then select test cases from \(T_b\) to kill all these remaining faulty versions, and to form a Second-Round Test Suite.
3. Finally, merge the First-Round Test Suite and the Second-Round Test Suite to form a Final Test Suite.

Figure 1: A fault class hierarchy in general Boolean expressions

- Stuck-At-1 Fault (SA1). An occurrence of a condition is replaced by 1 in the expression.

A fault class hierarchy based on the subsumption relation is proposed by Chen et al [2], which is shown in Figure 1. In Figure 1, \(A \rightarrow B\) indicates a test suite that kills all possible faults in \(A\) can kill all possible faults in \(B\). The fault class hierarchy shows that five fault classes, \(CCF, CDF, ORF, ENF\) and \(ASF\), are core fault classes. That is, a test suite that can kill all faulty versions of these five core fault types, can kill faulty versions of all fault classes.

\[\text{CCF, CDF, ORF, ENF, ASF}\]
Strategy 2: ORF, ENF and ASF have higher priority.

1. Select test cases from $T_b$ to kill all faulty versions in ORF, ENF and ASF, and to generate a First-Round Test Suite.

2. Pick up all faulty versions in CCF and CDF, which have not been killed by the First-Round Test Suite. Then select test cases from $T_b$ to kill all these remaining faulty versions, and to form a Second-Round Test Suite.

3. Finally, merge the First-Round Test Suite and the Second-Round Test Suite, and to form a Final Test Suite.

In the following section, we will study the problem of prioritization of fault classes in fault-based Boolean-specification testing by comparing the two strategies in various aspects empirically.

3. EXPERIMENT
In order to study which fault classes prioritization strategies are better, we will design experiments in this section to answer following concrete questions.

1. Cost: Which strategy will yield a smaller Final Test Suite?

2. Fault detection capability: Which strategy will yield a smaller First-Round Test Suite. And which strategy will get a higher percentage of killing all faulty versions in all 10 fault classes after the First-Round Test Suite has been designed?

3. Fault detection efficiency: Which strategy will yield a Final Test Suite with better efficiency, which is usually measured by the APFD metric [12]? 

In our experiments, we will use the 20 general form expressions in TCAS system [11] as the experiment objects, which have been extensively used in the research of Boolean specification-based testing. And to make our results to be more convicive, we will also supplement 4 groups of randomly generated expressions as the experiment objects. In each expression group, there are 20 randomly generated general form expressions, where the number of variables in these expressions ranges from 10 to 20, and the number of logical connective $\land$ or $\lor$ ranges from 20 to 50.

In following sub-sections, we will use the experimental data on 20 TCAS expressions and 4 groups of randomly generated expressions to answer the above three questions.

3.1 Cost
Firstly, for TCAS expressions and 4 groups of randomly generated expressions, we compare the number of test cases in Final Test Suites generated by Strategy 1 and Strategy 2. And for comparison, we also examine the size of random test suites, which are obtained by randomly selecting test cases from the set $T_b$ until all faulty versions in 10 fault classes have been killed.

![Figure 2: Number of test cases in Final Test Suites for 20 TCAS expressions](image)

![Figure 3: Number of test cases in Final Test Suites for 4 groups of randomly generated expressions](image)

The experimental results about the total number of test cases of Final Test Suites for 20 TCAS expressions are shown in Figure 2, in which the vertical axis denotes the number of test cases in Final Test Suite for all 20 expressions, the horizontal axis denotes the fault classes prioritization strategies including Strategy 1, Strategy 2, and random selection. It is evident that the random selection method is much worse that Strategy 1 and Strategy 2 since it requires much more test cases to kill all possible faulty versions. And we can conclude from Figure 2 that, for most expressions, the sizes of Final Test Suites generated by Strategy 1 are slightly larger than the sizes of Final Test Suites generated by Strategy 2. It is also shown that the gaps between two strategies are very narrow.

Besides the 20 TCAS expressions, we investigate 4 groups of randomly generated Boolean expressions too, and display the results in Figure 3. Similarly, for each expression group, we count the number of test cases in Final Test Suite for all 20 expressions, and list the total number of test cases for 4 groups in Figure 3. The results about randomly generated expressions are very similar to the results about TCAS expressions.

Therefore, we can conclude from the above results that the Strategy 1 is a slightly worse than Strategy 2 with respect to the number of test cases in the Final Test Suite.

3.2 Fault detection capability
In this sub-section, for TCAS expressions and 4 groups of randomly generated expressions, we compare the number of test cases in First-Round Test Suites generated by Strategy 1 and Strategy 2. The fault detection ability of the First-Round Test Suites are also compared.
Firstly, the experimental results for TCAS expressions show that, for all 20 expressions, the First-Round Test Suites generated by Strategy 1 kill all possible faulty versions in 10 fault classes. Another observation is that, for each TCAS expression, the size of First-Round Test Suite is equal to the size of corresponding Final Test Suite (so the Second-Round Test Suite will be empty) if Strategy 1 is utilized. The total number of test cases in the corresponding Final Test Suites generated by Strategy 1 kill all possible faulty versions in 10 fault classes. Further, the test cases in the Final Test Suites generated by Strategy 1 nearly contain all test cases, until the its size is equal to another Final Test Suite. Here for each expression, we make a modification on the larger Final Test Suite by removing some latest test cases. In order to make the comparison to be more fair, before the comparison, a pretreatment of Final Test Suites for each expression is needed to make two corresponding Final Test Suites contains the same number of test cases. In such two figures, the results of Strategy 1 and Strategy 2 are listed from left to right respectively for each expression group.

Secondly, we analyze the experimental results for randomly generated expressions. Different from the TCAS expressions, for most randomly generated expressions, the First-Round Test Suites of Strategy 1 cannot guarantee 100% fault detection (though very closed to 100%), and the sizes of these First-Round Test Suites are smaller than the sizes of corresponding Final Test Suites (but the gaps are very narrow). Here we report the total percentage of faults killed by First-Round Test Suites for 4 groups of randomly generated expressions in Figure 6. And in Figure 7, we report the total percentage of the sizes of First-Round Test Suites to the sizes of corresponding Final Test Suites. In such two figures, the results of Strategy 1 and Strategy 2 are listed from left to right respectively for each expression group.

Therefore, we can conclude that, for both TCAS expressions and randomly generated expressions, First-Round Test Suites generated by Strategy 1, which nearly contain all test cases in the corresponding Final Test Suites, could kill almost all faulty versions of all 10 fault classes. But the First-Round tests suite generated by Strategy 2 can kill only 40% - 60% (TCAS) or 50% - 70% (randomly generated expressions) faulty versions by utilizing 10 – 25% (TCAS) or 20 – 35% (randomly generated expressions) test cases in the corresponding Final Test Suite.

3.3 Fault detection efficiency

Finally, we will compare the Final Test Suites that generated by Strategy 1 and 2 respectively, with respect to fault detection efficiency of killing faults.

Note that for many expressions, the sizes of Final Test Suites generated by Strategy 1 and 2 are different. So it is unfair to compare their efficiency when they contain different numbers of test cases. In order to make the comparison to be more fair, the corresponding Final Test Suites for each expression is needed to make two corresponding Final Test Suites contains the same number of test cases. Here for each expression, we make a modification on the larger Final Test Suite by removing some latest test cases, until the its size is equal to another Final Test Suite.

The test suite’s efficiency of fault detection could be evaluated by a popular metric which is named as Average Percentage of Fault Detection (APFD) [12]. Considering a prioritized test suite $T = \{T_1, T_2, \ldots, T_m\}$, and a prophetic fault set $FS = \{f_1, f_2, \ldots, f_k\}$ of test object. If $tf_i$ is denoted as the index of the first test case that detects the fault $f_i$, the APFD for $T$ is:

$$APFD(T) = 1 - \frac{tf_1 + tf_2 + \cdots + tf_k}{k \times m} + \frac{1}{2m} \quad (1)$$

The APFD metric values of 20 expressions in TCAS system
and 4 randomly generated groups are shown in Figure 8 and Figure 9-12. In these figures, for each expressions index, the results of Strategy 1 and 2 are listed from left to right. It is clear that, for most expressions, the Final Test Suites that generated by Strategy 1 have a higher APFD metric value than the Final Test Suites that generated by Strategy 2.

Therefore, we could conclude that Strategy 1 is better than Strategy 2 with respect to fault detection efficiency, which is measured by the extensively used APFD metric.

4. DISCUSSION

By analyzing all experimental results given in above section, we could reach the following conclusions.

(1) For most Boolean expressions, Strategy 1 needs slightly more test cases than Strategy 2 to kill all possible faulty versions in 10 fault classes.

(2) For Strategy 1, the First-Round Test Suite and the Final Test Suite are almost identical, and hence the First-Round Test Suite kills almost all faults. And for Strategy 2, the First-Round Test Suite uses 10 – 35% test cases in Final Test Suite to kill 40 – 70% faults.

(3) The Final Test Suites generated by Strategy 1 usually have a higher efficiency of killing all possible faulty classes in ten fault classes.

These results suggest that problem of prioritization of fault classes in fault-based Boolean-Specification testing may have different solutions depending on the different requirements of testing. The prioritization strategy that has a higher efficiency of killing faults is needed when the testing resource is limited. So fault classes $CCF$, $CDF$, $ORF$ should have a higher priority when generating test cases in this circumstance. However, if there is not any risk that testing may be terminated exceptionally, the higher efficiency of killing faults is not important yet. So the prioritization strategy that requires a smaller Final Test Suite is better, and fault classes $ORF$, $ENF$ and $ASF$ should have a higher priority.

5. CONCLUSION

Fault-based approach has been widely used in generating test cases for testing Boolean expressions. The fault class hierarchy has a close relationship with the fault-based testing, since generating test cases for different fault classes results in a different testing performance. For general form Boolean expressions, it has been proven that $CCF$, $CDF$, $ORF$, $ENF$ and $ASF$ are core fault classes. These 5 fault classes could be further divided into two groups (\{$CCF$, $CDF$\} and \{ORF, ENF, ASF\}, where the former fault classes are co-stronger than all the other fault classes. So, there is a challenge of fault class prioritization that which group of fault classes should be considered firstly. By designing some experiments on TCAS expressions and some randomly generated expressions, experimental results suggest that different fault prioritization strategies should be selected depending on the particular testing constraints. In particular, when test resource is limited and there is a risk that testing may be terminated exceptionally, the strategy that $CCF$ and $CDF$ have higher priority, should be adopted since it may yield a higher fault detection efficiency.

6. ACKNOWLEDGEMENT

The work described in this paper was partially supported by the National Natural Science Foundation of China (90818027, 91018005, 60803007, and 61003020), the Major State Basic Research Development Program of China (973 Program: 2009CB320703), and the Australian Research Council (ARC DP0771733).

7. REFERENCES


Figure 8: APFD of Final Test Suites for 20 TCAS expressions

Figure 9: APFD of Final Test Suites for 20 randomly generated expressions in 1st Group

Figure 10: APFD of Final Test Suites for 20 randomly generated expressions in 2nd Group

Figure 11: APFD of Final Test Suites for 20 randomly generated expressions in 3rd Group

Figure 12: APFD of Final Test Suites for 20 randomly generated expressions in 4th Group