The Journal of Systems and Software 210 (2024) 111933

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Do we need high-order mutation in fault-based Boolean-specification testing?☆

Ziyuan Wang [a,*], Min Yu [a], Yang Feng [b], Weifeng Zhang [a]

[a] School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

ARTICLE INFO

Keywords:
Boolean-specification testing
High-order mutation testing
Weak mutation testing
Coupling effect hypothesis
Fault detection

ABSTRACT

Fault-based Boolean-specification testing is an essential weak mutation testing technique since the executing paths of the programs are usually dependent on those Boolean expressions in predicate statements. In mutation testing, the coupling effect hypothesis assumes that a test set that detects all the simple faults in the program can detect a high percentage of complex faults. Such a hypothesis suggests that we do not need to perform high-order mutation testing in practice. Many empirical results have shown the existence of the coupling effect. However, as fault-based Boolean-specification testing is mainly used in safety-critical scenarios, the question of whether high-order mutation testing is necessary for Boolean-specifications needs to be answered more strictly. To answer whether we need high-order mutation testing in fault-based Boolean-specification testing, we conduct an empirical study on general-form Boolean expressions. Experimental results show us that: (1) Generally, it is slightly easier to kill high-order mutants (HOMs) than first-order mutants (FOMs) since fault detection probabilities for complex faults (HOMs) are marginally higher than that for simple faults (FOMs) for almost all the cases. (2) However, for any fault types in Boolean-specifications, the coupling effect hypothesis is not guaranteed to hold with absolute certainty. (3) When HOMs involve the expression negation fault (ENF) or the operator reference fault (ORF), the percent of killed HOMs is lower. Such results suggest that, if the testing resource is sufficient, high-order mutation testing could be carried out on fault types ENF and ORF in fault-based Boolean-specification testing.

1. Introduction

Mutation testing is one of the fault-based testing techniques that have been widely studied and performed in practice (Wong and Mathur, 1995; Jia and Harman, 2011; Papadakis et al., 2019; Sánchez et al., 2022). In mutation testing, mutants are obtained by seeding some faults into an original program to simulate the real mistakes that programmers often make. If a test case detects a fault seeded in a mutant, we say that the test case kills the mutant. The mutation score, which indicates the proportion of seeded faults that are detected by the given test set, can be utilized as a testing criterion to evaluate the effectiveness of the test set. On this basis, the evaluation results of test cases' effectiveness can also guide the generation and selection of high-quality test cases.

Though mutation testing has been proven to be a powerful technique, it may be computationally expensive. This is due to the execution of a large number of mutants to simulate all the potential faults in an original program. The computational cost of mutation testing is mainly the total cost of the executions of each mutant. So, there may be two approaches that can reduce the cost of mutation testing: reducing the number of mutants, or reducing the cost of the execution of each mutant.

In order to provide the theoretical basis for reducing the number of mutants, the competent programmer hypothesis and the coupling effect hypothesis were introduced (DeMillo et al., 1978; Acree et al., 1979). Both two hypotheses suggest focusing only on single and simple faults in mutation testing. The competent programmer hypothesis assumes that faults introduced by qualified programmers are merely simple so that these faults can be fixed by a small syntactic change. It means that only faults constructed from some simple syntactical changes should be considered in mutation testing. Therefore, many mutation operators have been proposed to simulate such simple syntactic changes in mutation testing. Meanwhile, the coupling effect hypothesis focuses on the issue of whether multiple faults need to be seeded in a single mutant. Programmers may make more than one fault in the practical

☆ Editor: Yan Cai.

* Corresponding author.
E-mail address: wangziyuan@njupt.edu.cn (Z. Wang).

https://doi.org/10.1016/j.jss.2023.111933
Received 26 April 2023; Received in revised form 3 November 2023; Accepted 14 December 2023
Available online 18 December 2023

programming process. Therefore, there was a viewpoint that double faults or even more faults should also be taken into consideration in mutation testing. However, compared to the first-order mutants (FOMs) that contain only a single fault, the high-order mutants (HOMs), which contain double or more faults, are too numerous to be practical as a source of simulated faults. Such a contradiction leads out the question: *is it necessary to perform high-order mutation testing at the expense of higher costs*? The coupling effect hypothesis states that "complex faults are coupled to simple faults in such a way that a test set that detects all the simple faults will detect a high percentage of the complex faults" (Jia and Harman, 2011). Many people believe in this hypothesis, so they believe that HOMs are most likely to be unimportant. Therefore, to answer whether high-order mutation testing is required in practice, we should answer the question firstly: *does the coupling effect hypothesis hold*? People have carried out many empirical studies for the problem of the existence of the coupling effect. Most of the experimental results have shown that test cases that kill FOMs can often kill a large percentage of HOMs (Offutt, 1992, 1989; Lipton and Sayward, 1978; Wah, 2000, 2003).

In order to reduce the cost of the execution of each mutant, the weak mutation testing technique was proposed. In weak mutation testing, instead of checking the result of the mutant after the execution of the entire program, only the internal state of the mutant needs to be checked after the execution of the mutated point in the program. If the specific internal state in the mutant is inconsistent with the internal state in the original program, we say that the mutant is killed. Many studies indicate that the mutation score obtained in weak mutation testing can also reflect the effectiveness of the test set well. Moreover, some studies indicate that test cases generated by the weak mutation testing can be as effective as mutation testing under certain conditions (Horgan and Mathur, 1990; Offutt and Lee, 1991; Offutt and Lee, 1994). The implementation of weak mutation testing depends on the specification of components and a set of associated component mutation transformations (Howden, 1982), where the components refer to elementary computational structures in the program, such as variables, relations, arithmetic, logic expressions, etc. Boolean logic expressions in the program are usually utilized to make a decision about which path is executed. Due to the importance of logic expressions in the program, fault-based Boolean-specification testing, which focuses on mistakes in logic expressions in the program, has been studied for a long time. Faults in a logic expression induced by the same mutation operator could be classified into the same category, namely fault type or fault class. We follow the definitions of ten fault types proposed by Kapoor and Bowen (2007) and Lau and Yu (2005) in this paper. These fault types are classified into the types of operand faults (including `CCF`, `CDF`, `LRF`, `MLF`, `SA0`, and `SA1`) and the types of operator faults (including `ASF`, `ENF`, `LNF`, and `ORF`).

As we mentioned previously, many empirical studies have shown the existence of coupling effects in most cases. However, these studies are only for ordinary mutation testing. There are few studies on whether there is a coupling effect in the weak mutation testing (especially in fault-based Boolean-specification testing). As fault-based Boolean-specification testing is mainly used in safety-critical scenarios, the question of whether high-order mutation testing is necessary for Boolean-specifications needs to be answered more strictly. Kapoor proved that in Boolean-specification testing, the coupling effect hypothesis holds for most of the fault types (including `ASF`, `CCF`, `CDF`, `LRF`, `MLF`, `ORF`, `SA0`, and `SA1`) based on a strict assumption that "there can be at most one fault in leaves of a Boolean expression tree" (Kapoor, 2006). However, such an assumption is too strict to reflect real-world faults, since it is unreasonable to limit the positions where programmers make mistakes when writing Boolean expressions. Therefore, the question of to what extent coupling effects exist in fault-based Boolean-specification testing still needs to be answered.

This paper mainly focuses on the question of whether the coupling effect hypothesis holds in the fault-based Boolean-specification testing. By answering such a question, the question of whether the high-order mutation testing is required for Boolean-specifications can be finally answered. To answer the mentioned questions, we conduct an empirical study. In our experiment, we use 20 general-form Boolean expressions extracted from the well-known TCAS II system and 20 general-form Boolean expressions extracted from a nuclear industrial distributed control system (NDCS) as experimental subjects, and take second-order mutants (2nd-HOMs) as representative of HOMs. Experimental results show us that: (1) Generally, it is slightly easier to kill HOMs than FOMs since fault detection probabilities for HOMs are marginally higher than that for FOMs for almost all the cases. (2) However, for any fault types in Boolean-specifications, the coupling effect hypothesis is not guaranteed to hold with absolute certainty. (3) When HOMs involve fault type ENF or ORF, the percent of killed HOMs is lower. It means that, rather than answering the question of whether the coupling effect hypothesis holds in fault-based Boolean-specification testing, the contribution of this paper also includes providing guidance for the use of HOMs in Boolean-specification testing.

The key contributions of this paper are as follows:

- We conduct a large-scale experiment involving 40 original general-form Boolean expressions from two real industrial systems, 24,535 FOMs, and 24, 863, 935 HOMs. By examining whether FOM-adequate test set can kill all the HOMs, we answer the question of whether coupling effect exists in fault-based Boolean-specification testing.
- For ten different mutation operators commonly utilized in Boolean-specification testing, we count the ratios of HOMs that can be killed by FOM-adequate test sets. We compare these ratios and point out that mutants involving some specific mutation operators require high-order mutation testing.
- Based on the experimental results, we give a suggested implication about do we need to perform high-order mutation in fault-based Boolean-specification testing. Moreover, we also provided a series of hierarchical suggestions on which fault types should be focused on when performing high-order mutation testing for Boolean-specifications.

The remainder of this paper is organized as follows. The Section 2 introduces the preliminaries. The Section 3 describe the design of out experiments including research questions, experimental subjects, evaluation metrics, and the process of data collection. The Section 4 illustrates experimental results, summarizes findings and remarks, and discusses potential threats to validity. The Section 5 surveys related works. And a conclusion is given finally.

2. Preliminaries

2.1. Mutation testing

Mutation testing can be used to measure the effectiveness of test cases detecting faults. Testers introduce some syntactic changes called "mutation operators" into an original program to create mutants. E.g., Table 1 shows a mutant p' obtained by replacing a logic operator (`||`) in an original program p by another logic operator (`&&`). A mutation operator is known as a transformational rule that creates a series of mutants from the original program. A test set T will be executed against each mutant p'. If the result of p' is different from the result of p for at least one test case in T, then the mutant p' is said to be "killed" by T; otherwise, it is said to be "survived". However, some mutants will never be killed because they always have the same result as the original program. They are syntactically different but functionally equivalent to the original program, so they are called "equivalent mutants". Mutation testing concludes with a testing criterion called "mutation score" that indicates the quality of test cases. The mutation score reflects the ratio of the number of mutants killed by test cases to the total number of non-equivalent mutants. Besides evaluating the effectiveness of test cases,

Table 1
An example of mutants in predicate statement.

Original program p	Mutant p'

`...`
`if ( a < 0 || b == 0 )`
`  return true;`
`else`
`  return false;`
`...`
	`...`
	`if ( a < 0 && b == 0 )`
	`  return true;`
	`else`
	`  return false;`
	`...`

mutation score can also be used improve the testing adequacy of test cases. After removing all the equivalent mutants, if the existing test sets cannot kill all the non-equivalent mutants, testers need to append extra test cases to improve the effectiveness of test set.

In mutation testing, mutants are created to simulate real faults in the original program. However, the quantity of all the potential faults for a given program may be huge, so it is hard to create mutants representing all of them. The feasibility of mutation testing depends on the control of the number of mutants. Both the competent programmer hypothesis and the coupling effect hypothesis, firstly introduced by DeMillo et al. (1978) and Acree et al. (1979), suggest focusing only on single and simple faults in mutation testing to reduce the number of mutants.

The competent programmer hypothesis states that all the programmers are competent enough so their programs are very close to the correct version. All the faults they made are a few simple syntactical faults. Based on such a hypothesis, simple syntactical changes of the program can be simulated mistakes that competent programmers could make. In mutation testing, only faults made by competent programmers are applied. In another word, only faults leaded by simple syntactical changes are applied.

The coupling effect hypothesis states that "test cases that distinguish all the programs differing from a correct one by only simple faults are so sensitive that they also implicitly distinguish more complex faults" (Acree et al., 1979). A simple fault is created by making a single syntactical change in an original program and a complex fault is created by making more than one changes in an original program. The coupling effect hypothesis is extended into the mutation coupling effect hypothesis that states "complex mutants that contain more than one faults are coupled to simple mutants that contain single fault in such a way that test cases that detect all the simple mutants for an original program can also detect a large percentage of complex mutants" (Offutt, 1992, 1989; Morell, 1983). Such a hypothesis provides the evidence why the mutants in traditional mutation testing are only simple mutants.

2.2. Fault-based Boolean-specification testing

The computational cost of mutation testing is mainly the total cost of the executions of all the mutants. Therefore, besides reducing the number of mutants, another way to reduce the computational cost of mutation testing is by reducing the cost of execution of each mutant. One attempt to reduce the execution cost of each mutant is weak mutation testing proposed by Howden (1982). It is indicated that test sets generated by weak mutation testing can be expected to be as effective as mutation testing under certain conditions. In weak mutation testing, the program p is composed of some simple component $c_1, c_2, \ldots, c_n$. The p' is a mutated version of p, which is made by changing the component c_i to the mutated c_i' $(i = 1, 2, \ldots, n)$. The mutant p' is killed if any execution of c_i' is different from c_i. Hence, weak mutation testing does not need to execute the whole program entirely. Howden proposed five types of program components that could be utilized in weak mutation testing: variable reference, variable assignment, arithmetic, relational, and Boolean logic expression (Howden, 1982).

As one of weak mutation testing techniques that focuses on a specific type of program component, i.e., Boolean logic expression, fault-based Boolean-specification testing technique has been studied for a long time. The Boolean logic expressions that appeared in predicate statements could be considered as the most critical components of a program. E.g., we can see from the Table 1 that the variables `a` and `b` determines the returned value of the program since the predicate statement "`if(a<0||b==0)`" decides which branches the program will go.

A Boolean expression consists of some Boolean variables $x_1, x_2, \ldots, x_n$, three logical operators '$\vee$' (`OR`), '$\wedge$' (`AND`), '$\neg$' (`NOT`), and the brackets '(' and ')'. Boolean-specification testing aims to detect faults in Boolean expressions extracted from the predicate logics in programs. Typically, faults introduced by programmers in programs may include path domain boundary faults caused by incorrect control predictions, missing or extra conditions and paths, and the incorrect use of operators or operators. When we focus on Boolean specifications, these faults manifest as the omission, insertion, or incorrect reference of Boolean operands or operators. Therefore, the ten fault types usually utilized in fault-based Boolean-specification testing could be classified into operand fault types and operator fault types (Kapoor and Bowen, 2007; Lau and Yu, 2005).

There are six types of operand faults refer to the omission, insertion, and incorrect reference of Boolean operands:

- `CCF`: clause conjunction fault is caused by replace an occurrence of a Boolean variable x_i by $(x_i \wedge x_j)$, where x_j may be any possible Boolean variable or its negation. E.g., $x_1 \wedge x_2$ is implemented as $(x_1 \wedge x_3) \wedge x_2$ or $(x_1 \wedge \neg x_3) \wedge x_2$.
- `CDF`: clause disjunction fault is caused by replace an occurrence of a Boolean variable x_i by $(x_i \vee x_j)$, where x_j may be any possible Boolean variable or its negation. E.g., $x_1 \wedge x_2$ is implemented as $(x_1 \vee x_3) \wedge x_2$ or $(x_1 \vee \neg x_3) \wedge x_2$.
- `LRF`: literal reference fault is caused by replace an occurrence of a Boolean variable by another Boolean variable or its negation. E.g., $x_1 \wedge x_2$ is implemented as $x_1 \wedge x_3$ or $x_1 \wedge \neg x_3$. It may sometimes be named as variable reference fault.
- `MLF`: missing literal fault is caused by omission of an occurrence of a Boolean variable. E.g., $x_1 \wedge x_2 \vee x_3$ is implemented as $x_1 \wedge x_3$ or $x_1 \vee x_3$. It may sometimes be named as missing variable fault.
- `SA0`: stuck-at -0 fault is caused by replace an occurrence of a Boolean variable by the logic constant `FALSE` (0). E.g., $x_1 \wedge x_2$ is implemented as $\texttt{FALSE} \wedge x_2$.
- `SA1`: stuck-at -1 fault is caused by replace an occurrence of a Boolean variable by the logic constant `TRUE` (1). E.g., $x_1 \wedge x_2$ is implemented as $\texttt{TRUE} \wedge x_2$.

There are four types of operator faults refer to the omission, insertion, and incorrect reference of Boolean operators:

- `ASF`: associative shift fault is caused by omission of a pair of brackets. E.g., $x_1 \wedge (x_2 \vee x_3)$ is implemented as $x_1 \wedge x_2 \vee x_3$.
- `ENF`: expression negation fault is caused by replace a subexpression by its negation. E.g., $x_1 \wedge (x_2 \vee x_3)$ implemented as $x_1 \wedge \neg(x_2 \vee x_3)$.
- `LNF`: literal negation fault is caused by replace an occurrence of a Boolean variable by its negation. E.g., $x_1 \wedge (x_2 \vee x_3)$ implemented as $\neg x_1 \wedge (x_2 \vee x_3)$. It may sometimes be named as variable negation fault.
- `ORF`: operator reference fault is caused by replace an occurrence of a logic connective '$\wedge$' (or '$\vee$') with '$\vee$' (or '$\wedge$'). E.g., $x_1 \wedge x_2$ is implemented as $x_1 \vee x_2$.

Kuhn firstly presented that it is possible to use theoretical approach to compare fault types (Kuhn, 1999; Tsuchiya and Kikuno, 2002). Kapoor and Bowen gave a fault class hierarchies for general-form Boolean expressions (Kapoor and Bowen, 2007). Chen et al. indicated the incorrect fault relationships in previous works and introduced a new fault class hierarchy for general-form Boolean specifications (Chen

et al., 2011). They proved that ASF, CCF, CDF, ENF, and ORF are core fault classes. Detection of faults with these five core types could guarantee the detection of faults with the remain five types.

In Boolean-specification testing, a test case is a n-bit vector. Suppose a test case is an input of the Boolean expression, and the output result is different from the expected result, then the test case is said to detect a fault. A test set is said to detect a fault if it includes at least one test case that can detect such a fault. The mutation score can evaluate the test case's ability to detect faults in Boolean expressions, and guide the generation of high-quality test cases. Although there are differences between detecting faults in specific components (e.g. Boolean-specifications) and in entire programs (Papadakis et al., 2014), the quality of test cases for Boolean-specifications could reflect the quality of test cases for the entire program, and the generation of test cases for Boolean-specifications could guide the generation of test cases for the entire program too.

Focusing on Boolean-specifications can significantly reduce the cost of mutation testing, as the computational cost of evaluating Boolean expressions is much lower than the computational cost of executing the entire program. Although extracting Boolean logic from a program may require some effort, this burden is one-time. The threat that comes with cost reduction of weak mutation testing technique is the problem of clean program assumption (Chekam et al., 2017). The program's behavior, such as the selection of the execution path, is usually determined by the Boolean logics appearing in predicate statements, which is the component that Boolean-specification testing focuses on. If the predicate statement we are concerned about appears in a block controlled by another predicate statement containing a fault, we may have no opportunity to observe the result of this Boolean expression.

2.3. High-order mutation for Boolean- specifications

The first-order mutant (FOM) is generated by performing a mutation operator on an original program once. The high-order mutant (HOM) is generated by performing one or more mutation operators more than once on an original program. Especially, the second-order mutant (2nd-HOM), sometimes called the double fault, is generated by performing one mutation operator twice on an original program, or performing two different mutation operators one by one on an original program.

When performing high-order mutation testing in Boolean-specification testing, one of ten mutation operators is performed on the original Boolean expression to generate a FOM after the first mutation operation, then a same mutation operator or a different mutation operator is performed on such a FOM to generate a 2nd-HOM after the second mutation operation. E.g., for an original Boolean expression $x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$, a FOM $\neg x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ could be generated by performing the mutation operator LNF once, and two 2nd-HOMs $\neg x_1 \wedge (x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ and $\neg x_1 \vee (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ could be generated by performing the same mutation operator LNF on such a FOM again or by performing another mutation operator ORF on such a FOM.

In most cases, the mutation operation that generates the FOM based on the original expression and the mutation operation that generates the 2nd-HOM based on the FOM are independent of each other. In such a situation, the order of the two mutation operations is free. However, there are also some cases where the two mutation operations are not independent, because the position of the second mutation operation may be connected with the position of the first mutation operation. In such a situation, two mutation operations must be performed in the given order. For example, considering an original Boolean expression $x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$:

- If there are two different FOMs $(x_1 \wedge x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ and $x_1 \wedge \neg x_2 \vee \neg x_3 \wedge x_4 \vee x_5$, a 2nd-HOM $(x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ can be generated by merge such two FOMs directly. The mutation operations that generate such two FOMs do not influence each other since they modify the original Boolean expression in the different positions. In such a case where the order of two mutation operations could be exchanged, we say that two mutation operations involved in such a 2nd-HOM are independent.
- If a FOM $(x_1 \wedge x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$ is generated by a mutation operation that replace the variable x_1 in original expression by $(x_1 \wedge x_2)$. Based on such a FOM, another mutation operation removes the first pair of brackets to generate a 2nd-HOM $x_1 \wedge x_2 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$. Two mutation operations perform on the same position in the original Boolean expression, so the first mutation operation may influence the second mutation operation. E.g., if we apply the second mutation operator firstly and the first mutation operator secondly, there will be a different 2nd-HOM $(x_1 \wedge x_2) \wedge \neg x_2 \vee \neg x_3 \wedge x_4 \vee x_5$. In such a case where the order of two mutation operations cannot be exchanged, we say that two mutation operations involved in such a 2nd-HOM are not independent.

Above issue may cause another problem in the high-order mutation testing called the redundancy of mutants. If two mutation operations involved in a 2nd-HOM are independent, there may be duplicate HOMs in the process of generating 2nd-HOMs automatically. E.g., for an original expression $x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$, there could be a FOM $x_1 \wedge \neg x_2 \vee \neg x_3 \wedge x_4 \vee x_5$ by performing an ASF mutation operator firstly and a 2nd-HOM $x_1 \wedge \neg x_2 \vee \neg x_3 \wedge x_4 \vee (x_5 \wedge x_1)$ by performing a CCF mutation operator on such a FOM. Such a 2nd-HOM could be also obtained by performing CCF mutation operator firstly (obtain a FOM $x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee (x_5 \wedge x_1)$) and ASF mutation operator secondly (obtain the same 2nd-HOM by removing the first pair of brackets in such a FOM).

Although the computational cost of evaluating a Boolean expression is low, considering the number of mutants increases exponentially after performing 2nd-order mutation operators, the high-order mutation testing for Boolean-specifications should be cautious.

3. Experimental setup

To answer the question of whether high-order mutation testing is required in fault-based Boolean-specification testing, we design experiments to answer the question of whether the coupling effect hypothesis holds for Boolean-specifications. We will investigate the following three research questions to guide the use of high-order mutants in fault-based Boolean-specification testing.

- **RQ1**: Are first-order mutants (FOMs) more difficult to be killed than second-order mutants (2nd-HOMs)?
- **RQ2**: Can a test set generated for killing all the FOMs kill all the related 2nd-HOMs?
- **RQ3**: For which fault types is it necessary to perform high-order mutation testing?

3.1. Experimental subjects

We select two sets, each containing 20 general-form Boolean expressions, as the experimental subjects (see Table 2). One set of Boolean expressions was extracted from the TCAS II system by Weyuker et al. (1994). It was widely utilized as a benchmark in the field of Boolean-specification testing (Tai, 1996; Chen et al., 1999; Yu and Lau, 2002; Kobayashi et al., 2002; Yu et al., 2006; Kaminski and Ammann, 2009; Arcaini et al., 2011; Gargantini and Fraser, 2011; Yu and Lau, 2012; Paul et al., 2014; Yu and Tsai, 2018; Paul et al., 2021). Another set of Boolean expressions was extracted from a nuclear industrial distributed control system (NDCS) located in Jiangsu Nuclear Power Corporation, China (Zong et al., 2021).[1]

[1] Some Boolean expressions are slightly different from those given in Zong et al. (2021) because some mistakes have been corrected.

Table 2
Experimental subjects: general-form Boolean expressions extracted from the TCAS II system and the Nuclear DCS system.

No.	Boolean expression
TCAS1	$x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$
TCAS2	$\neg(x_1 \wedge x_2) \wedge (x_4 \wedge \neg x_5 \wedge x_6 \vee \neg x_4 \wedge x_5 \wedge \neg x_6 \vee \neg x_4 \wedge \neg x_5 \wedge \neg x_6) \wedge (x_1 \wedge x_3 \wedge (x_4 \vee x_5) \wedge x_8 \vee x_1 \wedge (x_4 \vee x_5) \wedge \neg x_8 \vee x_2 \wedge (x_5 \vee x_6))$
TCAS3	$\neg(x_3 \wedge x_4) \wedge (\neg x_5 \wedge x_6 \wedge \neg x_7 \wedge \neg x_1 \wedge (x_2 \wedge x_3 \vee \neg x_2 \wedge x_4))$
TCAS4	$x_1 \wedge x_3 \wedge (x_4 \vee x_5) \wedge x_8 \vee x_1 \wedge (x_4 \vee x_5) \wedge \neg x_8 \vee x_2 \wedge (x_5 \vee x_6)$
TCAS5	$\neg x_5 \wedge x_6 \wedge \neg x_7 \wedge \neg x_1 \wedge (x_2 \wedge x_3 \vee \neg x_2 \wedge x_4)$
TCAS6	$(\neg x_1 \wedge x_2 \vee x_1 \wedge \neg x_2) \wedge \neg(x_3 \wedge x_4) \wedge \neg(x_7 \wedge x_8) \wedge ((x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge x_5 \wedge (x_6 \wedge x_7 \vee \neg x_6 \wedge x_8))$
TCAS7	$(x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge x_5 \wedge (x_6 \wedge x_7 \vee \neg x_6 \wedge x_8)$
TCAS8	$(x_1 \wedge ((x_3 \vee x_4 \vee x_5) \wedge x_7 \vee x_1 \wedge x_6 \vee x_3 \wedge (x_6 \vee x_7 \vee x_8 \vee x_9)) \vee (x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5) \wedge x_9) \wedge \neg(x_1 \wedge x_2) \wedge \neg(x_3 \wedge x_4) \wedge$ $\neg(x_3 \wedge x_5) \wedge \neg(x_4 \wedge x_5) \wedge \neg(x_6 \wedge x_7) \wedge \neg(x_6 \wedge x_8) \wedge \neg(x_6 \wedge x_9) \wedge \neg(x_7 \wedge x_8) \wedge \neg(x_8 \wedge x_9)$
TCAS9	$x_1 \wedge (\neg x_2 \wedge \neg x_3 \vee x_2 \wedge x_3 \wedge \neg(\neg x_6 \wedge x_7 \wedge x_8 \wedge \neg x_9 \vee \neg x_7 \wedge x_8 \wedge x_9) \wedge \neg(\neg x_6 \wedge x_7 \wedge x_{12} \wedge x_{11} \vee \neg x_7 \wedge \neg x_9 \wedge x_{11})) \vee x_6$
TCAS10	$x_1 \wedge ((x_3 \vee x_4 \vee x_5) \wedge x_7 \vee x_1 \wedge x_6 \vee x_3 \wedge (x_6 \vee x_7 \vee x_8 \vee x_9)) \vee (x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5) \wedge x_9$
TCAS11	$(\neg x_1 \wedge x_2 \vee x_1 \wedge \neg x_2) \wedge \neg(x_3 \wedge x_4) \wedge \neg(x_7 \wedge x_8) \wedge \neg(x_9 \wedge x_{10}) \wedge ((x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge x_5 \wedge (\neg x_9 \vee \neg x_7 \wedge \neg x_{11} \wedge \neg x_{10} \wedge (\neg x_8 \vee \neg x_{11})))$
TCAS12	$(x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge x_5 \wedge (\neg x_9 \vee \neg x_7 \wedge \neg x_{11} \vee \neg x_{10} \wedge (\neg x_8 \vee \neg x_{11}))$
TCAS13	$(\neg x_1 \wedge x_2 \vee x_1 \wedge \neg x_2) \wedge \neg(x_3 \wedge x_4) \wedge (x_6 \wedge \neg x_7 \wedge \neg x_8 \vee \neg x_6 \wedge x_7 \wedge \neg x_8 \vee \neg x_6 \wedge \neg x_7 \wedge \neg x_8) \wedge \neg(x_{10} \wedge x_{11}) \wedge ((x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge$ $x_5 \wedge (x_6 \vee (x_9 \wedge (x_7 \wedge x_{10} \vee x_8 \wedge x_{11}))))$
TCAS14	$(x_1 \wedge x_3 \vee x_2 \wedge x_4) \wedge x_5 \wedge (x_6 \vee (x_9 \wedge (x_7 \wedge x_{10} \vee x_8 \wedge x_{11})))$
TCAS15	$(x_1 \wedge (\neg x_4 \vee \neg x_5 \vee x_4 \wedge x_5 \wedge \neg(\neg x_6 \wedge x_7 \wedge x_8 \wedge \neg x_9 \vee \neg x_7 \wedge x_8 \wedge x_9) \wedge (\neg x_6 \wedge x_7 \wedge x_{12} \wedge x_{11} \vee \neg x_7 \wedge \neg x_9 \wedge x_{11})) \vee \neg(\neg x_6 \wedge x_7 \wedge x_8 \wedge$ $\neg x_9) \wedge \neg(\neg x_6 \wedge x_7 \wedge x_{12} \wedge x_{11} \vee \neg x_7 \wedge \ \neg x_9 \wedge x_{11}) \wedge (x_2 \vee x_3 \wedge \neg x_{13} \vee x_6)) \wedge (x_1 \wedge \neg x_2 \wedge \neg x_3 \vee \neg x_1 \wedge x_2 \wedge \neg x_3 \vee \neg x_1 \wedge \neg x_2 \wedge x_3)$
TCAS16	$x_1 \vee x_2 \vee x_3 \vee \neg x_3 \wedge \neg x_4 \wedge x_5 \wedge x_6 \wedge \neg x_7 \wedge \neg x_8 \vee x_9 \wedge (x_{10} \vee x_{11}) \wedge \neg x_{12}$
TCAS17	$(x_1 \wedge (\neg x_4 \vee \neg x_5 \vee x_4 \wedge x_5 \wedge \neg(\neg x_6 \wedge x_7 \wedge x_8 \wedge \neg x_9 \vee \neg x_7 \wedge x_8 \wedge x_9) \wedge (\neg x_6 \wedge x_7 \wedge x_{12} \wedge x_{11} \vee \neg x_7 \wedge \neg x_9 \wedge x_{11})) \vee$ $\neg(\neg x_6 \wedge x_7 \wedge x_8 \wedge \neg x_9) \wedge \neg(\neg x_6 \wedge x_7 \wedge x_{12} \wedge x_{11} \vee \neg x_7 \wedge \neg x_9 \wedge x_{11}) \wedge (x_2 \vee x_3 \wedge \neg x_{13} \vee x_6))$
TCAS18	$x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge x_4 \wedge \neg x_5 \wedge x_6 \wedge (x_7 \vee \neg x_7 \wedge (x_8 \vee x_9)) \wedge \neg(x_{10} \wedge x_{11} \vee \neg x_{10} \wedge x_{12} \vee x_{13})$
TCAS19	$x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge (\neg(x_6 \wedge (x_7 \vee \neg x_7 \wedge (x_8 \vee x_9))) \vee x_6 \wedge (x_7 \vee \neg x_7 \wedge (x_8 \vee x_9)) \wedge \neg x_4 \wedge \neg x_5) \wedge \neg(x_{10} \wedge x_{11} \vee \neg x_{10} \wedge x_{12} \wedge \neg x_{13})$
TCAS20	$x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge (x_6 \wedge (x_7 \vee \neg x_7 \wedge (x_8 \vee x_9)) \wedge (\neg x_5 \wedge \neg x_{14} \vee x_4) \vee \neg x_{14} \wedge (x_{10} \wedge x_{11} \vee \neg x_{10} \wedge x_{12} \wedge \neg x_{13}))$
NDCS1	$\neg x_1 \wedge x_2 \wedge \neg x_3 \vee \neg(x_4 \wedge \neg x_5) \vee \neg x_6$
NDCS2	$\neg x_1 \wedge \neg(x_2 \vee \neg(\neg(x_3 \vee x_4) \vee x_5)) \vee x_6$
NDCS3	$x_1 \vee x_2 \wedge (\neg x_3 \vee \neg x_4) \vee \neg x_5 \vee \neg x_6$
NDCS4	$x_1 \wedge (x_2 \wedge (x_3 \wedge x_4)) \vee (x_5 \vee \neg x_6)$
NDCS5	$(x_1 \wedge \neg x_2) \wedge x_3 \wedge (\neg(\neg x_3 \wedge x_1) \wedge x_4) \wedge \neg(\neg x_5 \wedge \neg x_1) \wedge (x_5 \wedge x_2) \wedge x_6 \vee \neg x_2 \vee \neg x_5 \vee x_7$
NDCS6	$x_1 \vee x_2 \wedge \neg x_3 \wedge (\neg x_4 \vee \neg x_5) \vee \neg x_6 \wedge \neg x_7 \vee x_8$
NDCS7	$\neg(x_1 \vee x_2) \wedge \neg x_3 \wedge (\neg(\neg x_4 \vee \neg x_3) \vee \neg x_3) \vee \neg x_5 \wedge x_6$
NDCS8	$x_1 \wedge x_2 \vee \neg x_3 \vee x_4 \wedge \neg(\neg x_5 \vee \neg x_2) \wedge x_6 \wedge \neg(x_7 \vee \neg x_8) \vee \neg x_7 \vee x_2 \wedge \neg x_4 \vee \neg x_5$
NDCS9	$x_1 \wedge x_2 \wedge \neg(\neg x_2 \vee x_3) \wedge x_4 \vee (\neg x_5 \vee \neg x_6) \vee \neg x_3 \vee x_4 \wedge x_6 \wedge (\neg x_6 \vee \neg x_2) \wedge \neg x_7 \vee \neg x_4$
NDCS10	$(x_1 \vee \neg x_2) \wedge \neg(\neg x_3 \vee \neg x_4) \vee x_4 \vee x_5 \wedge \neg x_4 \vee \neg(\neg x_4 \wedge \neg x_5) \wedge \neg(x_6 \vee x_1) \wedge \neg x_7 \vee (\neg x_8 \wedge x_3) \wedge \neg x_4 \wedge x_7$
NDCS11	$\neg x_1 \wedge x_2 \vee x_3 \vee x_4 \vee x_5 \vee \neg x_6$
NDCS12	$\neg(\neg x_1 \wedge x_2) \vee x_3 \wedge \neg x_4 \vee x_4 \vee x_5 \vee x_6 \vee x_7$
NDCS13	$x_1 \wedge ((\neg x_2 \vee \neg x_3) \wedge \neg x_4) \vee \neg x_3 \wedge \neg x_4 \wedge x_5 \wedge (\neg x_3 \wedge x_6) \vee x_9 \vee x_6 \vee \neg(\neg(\neg x_7 \vee \neg x_3) \vee x_8)$
NDCS14	$\neg x_1 \vee \neg x_2 \vee (x_3 \vee \neg x_4) \vee x_5 \wedge x_6$
NDCS15	$\neg x_1 \wedge \neg(\neg x_2 \wedge \neg(\neg x_3 \wedge x_4)) \wedge \neg x_2 \vee \neg((\neg x_5 \vee x_7) \wedge x_7) \vee \neg x_7 \vee \neg x_8 \vee \neg x_5 \vee \neg x_9 \wedge x_3$
NDCS16	$x_1 \wedge \neg x_2 \vee x_3 \wedge (\neg x_4 \vee \neg x_5) \vee \neg x_6$
NDCS17	$\neg((\neg x_1 \wedge \neg x_2) \wedge (\neg x_2 \vee \neg x_3)) \vee (\neg x_4 \wedge x_2) \wedge x_5 \vee \neg(x_3 \vee \neg x_4) \wedge x_6 \wedge x_2 \vee \neg x_7 \vee \neg x_8 \vee x_5$
NDCS18	$(\neg x_1 \vee x_2) \vee x_3 \wedge x_4 \wedge x_5 \vee \neg x_6$
NDCS19	$\neg x_1 \vee \neg(x_2 \vee \neg x_3) \wedge x_1 \wedge \neg x_4 \wedge (x_3 \vee x_5) \vee x_6 \vee (\neg x_7 \vee x_5) \vee \neg x_6 \wedge \neg x_8 \wedge \neg x_4$
NDCS20	$\neg x_1 \vee \neg x_2 \wedge \neg x_3 \vee (\neg x_4 \wedge x_5) \vee \neg(x_6 \vee x_2) \vee x_7$

In our experiment, we take 2nd-HOMs as representative of HOM. For each Boolean expression, we create mutants with ten fault types and get 33,714 FOMs, where there are 24,535 non-equivalent FOMs. On this basis, we mutate all the FOMs (including equivalent FOMs) once again and get 27,722,086 2nd-HOMs, where there are 24,863,933 non-equivalent 2nd-HOMs. Table 3 shows the numbers of FOMs and 2nd-HOMs for each original Boolean expression. Table 4 shows the numbers of FOMs and 2nd-HOMs with each fault type.

3.2. Experimental metrics

We use two metrics, including the *Fault Detection Probability* and the *Percent of 2nd-HOMs Killed by FOM-Adequate Test Set*, to answer above research questions.

3.2.1. Fault detection probability

For each mutant M, we calculate the probability of the event that the fault is detected by a randomly generated test case, to measure how easily such a mutant being killed. The fault detection probability, using $F\text{-}Probability(M)$ to denote, can be described as the ratio of failed test cases:

$$F\text{-}Probability(M) = \frac{\#AllKillTest(M)}{\#AllTests}$$

where the $AllKillTest(M)$ is the set of all the test cases that can kill the mutant M, and the $AllTests$ is the set of all possible test cases. Obviously, the higher $F\text{-}Probability(M)$ means that the fault M is easier to be detected; otherwise, M is harder to be detected.

For example, there are totally $2^5 = 32$ possible test cases for a Boolean expression with 5 variables (e.g. the Boolean expression TCAS1). If there are 3 test cases can kill one of its mutant M_1, the fault detection probability of M_1 is $F\text{-}Probability(M_1) = \frac{3}{32}$. And if there are 4 test cases can kill another mutant M_2, the fault detection probability of M_2 is $F\text{-}Probability(M_2) = \frac{4}{32}$. So the M_2 is easier to be detected than M_1.

3.2.2. Percent of the 2nd-HOMs killed by the FOM-Adequate test set

For a set of 2nd-HOMs, which are obtained by applying mutation operator O_i in the first-order mutation and mutation operator O_j in the second-order mutation, we calculate the percent of these 2nd-HOMs killed by the first-order mutation-adequate test set. Such a FOM-adequate test set can kill all the FOMs with fault type O_i and all the FOMs with fault type O_j. The percent of 2nd-HOMs killed by the FOM-adequate test set is:

$$K\text{-}Percent(O_i, O_j) = \frac{\#Mu(O_i, O_j)\ killed\ by\ MAT(O_i + O_j)}{\#Mu(O_i, O_j)}$$

Table 3
The number of FOM and 2nd-HOMs for each general-form original Boolean expression.

Expression	FOMs		2nd-HOMs	
	# All	# Non-equivalent	# All	# Non-equivalent
TCAS1	172	138	19 135	16 931
TCAS2	1077	791	617 548	556 754
TCAS3	467	354	125 462	112 117
TCAS4	561	467	177 004	166 516
TCAS5	371	289	81 915	73 460
TCAS6	901	728	441 730	414 298
TCAS7	475	423	131 743	126 266
TCAS8	2121	1624	2 345 199	2 185 545
TCAS9	1170	853	739 613	666 942
TCAS10	1057	801	606 334	555 834
TCAS11	1366	1024	1 001 015	916 106
TCAS12	714	638	290 447	280 562
TCAS13	1987	1372	2 082 998	1 840 904
TCAS14	781	704	347 773	337 168
TCAS15	3296	2576	5 639 504	5 311 694
TCAS16	992	805	551 980	516 399
TCAS17	2606	2025	3 559 952	3 335 826
TCAS18	1239	965	848 420	782 752
TCAS19	1658	1263	1 482 438	1 363 021
TCAS20	1510	1291	1 245 813	1 197 542
TCAS total	24 521	19 131	22 336 023	20 756 637
NDCS1	242	188	61 109	47 475
NDCS2	245	199	62 662	50 896
NDCS3	242	192	61 104	48 579
NDCS4	245	186	62 659	53 835
NDCS5	654	145	440 747	342 017
NDCS6	420	356	182 208	154 443
NDCS7	326	200	110 251	89 283
NDCS8	683	330	481 363	373 056
NDCS9	652	307	437 977	336 804
NDCS10	844	370	732 415	481 466
NDCS11	240	184	60 144	50 721
NDCS12	371	245	142 895	106 632
NDCS13	822	485	693 955	548 241
NDCS14	242	185	61 101	50 959
NDCS15	685	448	484 203	394 094
NDCS16	243	209	61 485	55 916
NDCS17	739	262	562 342	368 832
NDCS18	243	188	61 488	52 803
NDCS19	682	415	481 581	380 957
NDCS20	373	310	144 374	120 289
NDCS total	9193	5404	5 386 063	4 107 298

where the $Mu(O_i, O_j)$ is the set of all the non-equivalent 2nd-HOMs obtained by applying O_i in the first-order mutation operation and O_j in the second-order mutation operation. And the $MAT(O_i + O_j)$ is a FOM-adequate test set for all the non-equivalent FOMs with fault type O_i and all the non-equivalent FOMs with fault type O_j. Especially, if O_i and O_j are the same, $MAT(O_i + O_j)$ could be denoted as $MAT(O_i)$ to reflect a FOM-adequate test set for all the non-equivalent FOMs with fault type O_i.

Take the Boolean expression TCAS1 ($x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$) as example, there are four non-equivalent ORF mutants and $MAT(\texttt{ORF}) = \{00010, 10110\}$ is one of the FOM-adequate test set for all these non-equivalent mutants with fault type ORF,[2] there are 34 non-equivalent CDF mutants and $MAT(\texttt{CDF}) = \{00110, 01010, 10100, 11000, 11110\}$ is one of the FOM-adequate test set for all these non-equivalent mutants with fault type CDF. $MAT(\texttt{ORF} + \texttt{CDF}) = \{00110, 01010, 10100, 10110, 11000, 11110\}$ is one of the FOM-adequate test set for all the 38 non-equivalent mutants with fault type ORF or CDF.[3] If there are a total of 200 non-equivalent 2nd-HOMs in the set $Mu(\texttt{ORF}, \texttt{CDF})$, and 190 of them could be killed by at least one of test cases in the set $MAT(\texttt{ORF} + \texttt{CDF})$, the percent of 2nd-HOMs killed by such a FOM-adequate test set is $\frac{190}{200} = 95\%$.

3.3. Data collection

This experiment includes five steps to obtain the fault detection probabilities for FOMs and 2nd-HOMs and the percents of 2nd-HOMs killed by the FOM-adequate test sets. For each original Boolean expression, Fig. 1 illustrates the whole process of collecting experimental data.

1. **Generating and testing FOMs**: For each original Boolean expression extracted from the TCAS and NDCS system, create all possible FOMs by using ten mutation operators including ASF, CCF, CDF, ENF, LNF, LRF, MLF, ORF, SA0, and SA1. Assuming an original Boolean expression contains n Boolean variables, there will be 2^n test cases in the exhaustive test set. By traversing all the binary numbers with length n from 00...00 to 11...11, we can obtain all these 2^n test cases. By running all these 2^n test cases on the original version and all its mutants, all the equivalent mutants that cannot be killed by any test cases could be filtered out. For each non-equivalent mutant M, all the failed test cases that kill such a mutant could be collected into a set $AllKillTests(M)$.
2. **Generating and testing 2nd-HOMs**: For each FOM obtained in the first step (including both equivalent and non-equivalent mutants), create all possible 2nd-HOMs by using ten mutation operators including ASF, CCF, CDF, ENF, LNF, LRF, MLF, ORF, SA0, and SA1 again (note that the first-order and the second-order mutation operators could be the same). Run all possible exhaustive test cases on the original Boolean expression and all its 2nd-HOMs to filter out equivalent 2nd-HOMs. For each second-order non-equivalent mutant M, find all the failed test cases that kill such a mutant and collect them into a set $AllKillTests(M)$.
3. **Comparing fault detection probabilities**: For each non-equivalent FOM M, count the number of failed test cases in the set $AllKillTests(M)$ that can kill such a mutant, and calculate the fault detection probability $F\text{-}Probability(M)$ for such a mutant. And for each non-equivalent HOM, calculate the fault detection probability in the same way. Then we can compare the fault detection probabilities for FOMs and HOMs from the same original Boolean expression.
4. **Generating reduced FOM-adequate test set**: For each original Boolean expression and two mutation operators O_i and O_j, there are a set of non-equivalent FOMs $M_1, M_2, \ldots, M_k$ and their corresponding failed test sets $AllKillTests(M_1)$, $AllKillTests(M_2)$, $\ldots$, $AllKillTests(M_k)$. In order to obtain a minimal test set that can kill all these FOMs and contain as few test cases as possible, we use four popular test case reduction algorithms (including pure greedy algorithm Chen and Lau, 2003, GE algorithm Chen and Lau, 2003, GRE algorithm Chen and Lau, 2003, and the algorithm proposed by Harrold et al., 1993) to generate four reduced test sets. These reduced test sets are FOM-adequate test sets for all the mutants with fault types O_i and O_j (denoted as $MAT(O_i + O_j)$). If two mutation operators O_i and O_j are the same, generate FOM-adequate test sets for all the FOMs with fault type O_i (or O_j) and let $MAT(O_i)$ (or $MAT(O_j)$) be the $MAT(O_i + O_j)$. The reason why test reduction algorithms must be used in the experiment will be explained in the next subsection.

[2] Test case for Boolean specification could be denoted as a binary integer. E.g., two test case {false, false, false, true, false} and {true, false, true, true, false} for a Boolean expression with five inputs could be described as two binary integers 00010 and 10110.

[3] In most cases, $MAT(O_i + O_j) \neq MAT(O_i) \cup MAT(O_j)$ because some test reduction algorithms could be used when find mutation-adequate test sets. See Section 3.4.

Table 4
The number of 2nd-HOMs for each 1st/2nd-order fault type.

2nd-order mutation operator	1st-order mutation operator										
	ASF	CCF	CDF	ENF	LNF	LRF	MLF	ORF	SA0	SA1	HOM total
ASF	713 748	60 922 68 360	62 055 68 360	869 898	2845 2960	52 856 55 044	3487 3669	2721 2810	2828 2960	2786 2960	192 079 208 769
CCF	53 778 58 004	4 205 001 4 736 832	4 052 468 4 736 832	56 563 58 004	209 888 217 316	4 059 163 4 234 788	240 473 255 446	201 393 206 960	192 962 202 642	189 314 202 642	13 461 002 14 909 466
CDF	53 107 58 004	4 058 205 4 736 832	4 058 205 4 736 832	56 555 58 004	209 740 217 316	4 043 445 4 234 788	240 134 255 446	201 431 206 960	191 752 202 642	188 905 202 642	13 301 479 14 909 466
ENF	727 748	66 028 68 360	66 115 68 360	744 898	2862 2960	53 689 55 044	3560 3669	2719 2810	2865 2960	2864 2960	202 174 208 769
LNF	2845 2960	217 130 227 672	217 130 227 672	2862 2960	10 068 10 898	199 939 206 418	12 529 12 949	9965 10 336	9963 10 336	9963 10 336	692 395 722 537
LRF	52 856 55 044	4 271 376 4 509 160	4 247 584 4 509 160	53 689 55 044	199 939 206 418	3 919 428 4 028 370	235 300 242 497	191 703 196 624	186 450 192 306	185 794 192 306	13 544 120 14 186 929
MLF	3571 3761	255 069 281 358	254 869 281 358	3560 3669	13 133 13 649	250 844 259 981	15 510 16 175	12 459 12 913	12 390 12 921	12 361 12 921	833 766 898 706
ORF	2721 2810	208 275 217 316	208 275 217 316	2719 2810	9965 10 336	191 703 196 624	11 879 12 249	9078 9814	9964 10 336	9992 10 336	664 571 689 947
SA0	2828 2960	215 516 227 672	205 562 227 672	2865 2960	10 462 10 898	199 286 206 418	12 414 12 949	9964 10 336	9936 10 336	9848 10 336	678 680 722 537
SA1	2786 2960	202 930 227 672	210 428 227 672	2864 2960	10 441 10 898	198 160 206 418	12 387 12 949	9992 10 336	9848 10 336	9835 10 336	669 670 722 537
HOM total	175 930 187 999	13 760 451 15 301 234	13 582 691 15 301 234	183 290 188 207	679 342 703 649	13 168 515 13 683 893	787 673 827 998	651 426 669 899	628 958 657 775	621 661 657 775	24 863 935 27 722 086

In each cell, the first line is the number of non-equivalent mutants while the second line is the number of all possible mutants.
The total number of (non-equivalent) 2nd-HOMs is smaller than the count of the numbers of (non-equivalent) 2nd-HOMs with specified first-order/second-order fault types, since there are some redundant 2nd-HOMs.

5. **Coupling effect analysis**: For each 2nd-HOM where O_i is the first-order mutation operator and O_j is the second-order mutation operator, check whether such a mutant could be killed by the reduced FOM-adequate test set $MAT(O_i + O_j)$ generated by one of four test case reduction algorithms. Then compute $K\text{-}Percent(O_i, O_j)$ for all possible combination of two mutation operators O_i and O_j. Note that O_i and O_j could be the same mutation operator, so there are totally 100 combinations of the first-order and the second-order mutation operators.

A tool named "BoolMuTest" is used in experiment to generate and test FOMs and HOMs. By running all possible test cases on each mutant, the tool could filter out all the equivalent mutants that any test cases cannot kill. And for each non-equivalent mutant, the tool could find all the failed test cases that kill such a mutant (Wang and Yu, 2018).

3.4. Test reduction in generating FOM-adequate test set

In the 4th step of the experimental process, we need the FOM-adequate test sets for a group of FOMs with fault types O_i and O_j. It could be obtained by taking the union of failed test sets of these FOMs. However, the reduced FOM-adequate test set generated by the test case reduction techniques could help us improve the accuracy of coupling effect analysis.

For a given set of faults, the test case reduction is to find a subset of an existing test set to ensure that such a subset with few test cases can detect all the given faults too. E.g., for an original Boolean expression TCAS1 ($x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$), one of its mutants $(a \wedge \neg a) \wedge (\neg b \vee \neg c) \wedge d \vee e$ could be killed by any test case in the test set {10010, 10110, 11010}, and another mutant $a \wedge (\neg b \vee \neg c) \wedge d \vee (e \wedge \neg d)$ could be killed by any test case in the test set {00011, 00111, 01011, 01111, 11111} . Obviously, the test set {00011, 00111, 01011, 01111, 10010, 10110, 11010, 11111} could kill all these two mutants. However, a reduced test set {00011, 10010} could kill these two mutants too. Several test case reduction algorithms were proposed, such as pure greedy algorithm (Chen and Lau, 2003), GE algorithm (Chen and Lau, 2003), GRE algorithm (Chen and Lau, 2003), and the algorithm proposed by Harrold et al. (1993). In our experiment, all these four algorithms are utilized to make experimental results more diverse.

The reason why the reduced FOM-adequate test set needs to be utilized in the coupling effect analysis is that, the redundant test cases in the FOM-adequate test case set may kill more HOMs, resulting in the value of $K\text{-}Percent(O_i, O_j)$ becomes incorrectly higher. In extreme cases, a test set containing enough redundant test cases can kill all possible HOMs. E.g., there are 33 non-equivalent CCF mutants for original Boolean expression TCAS1 ($x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$). For each non-equivalent mutant M_i ($i = 1, 2, \ldots, 33$), a test set $AllKillTests(M_i)$ contains all test cases that can kill the mutant M_i. Taking the union of these 33 test sets, a test set $\cup_{i=1}^{33} AllKillTests(M_i)$ containing 16 test cases (00001, 00011, 00101, 00111, 01001, 01011, 01101, 01111, 10001, 10010, 10101, 10110, 11001, 11010, 11101, 11111) is a FOM-adequate test set for FOMs with fault type CCF. After performing a test reduction algorithm on these test cases, a reduced test set containing only 4 test cases (00001, 10110, 11010, 11111) is also a FOM-adequate test set for FOMs with fault type CCF. If we use the test set with 16 test cases as the $MAT(\mathrm{CCF})$, all the 2nd-HOMs whose both the first-order and the second-order mutation operator is CCF could be killed by such a $MAT(\mathrm{CCF})$, so $K\text{-}Percent(\mathrm{CCF}, \mathrm{CCF}) = 100\%$; and if we use the reduced test set as the $MAT(\mathrm{CCF})$, only 99.92% 2nd-HOMs whose both the first-order and the second-order mutation operator is CCF could be killed by such a $MAT(\mathrm{CCF})$, so $K\text{-}Percent(\mathrm{CCF}, \mathrm{CCF}) = 99.92\%$.

4. Experiment results

Due to the massive number of mutants, especially HOMs, although the cost of executing one test case on one mutant is negligible, the whole experiment was very time-consuming (see Table 5). It takes about one minute CPU time to test all the FOMs and find all failed test cases for each FOM, 14,190.78 min (≈ 9.85 days) to test all the HOMs and find all failed test cases for each HOM, and 186,018.87 min (≈ 129.18 days) to analyze coupling effect.

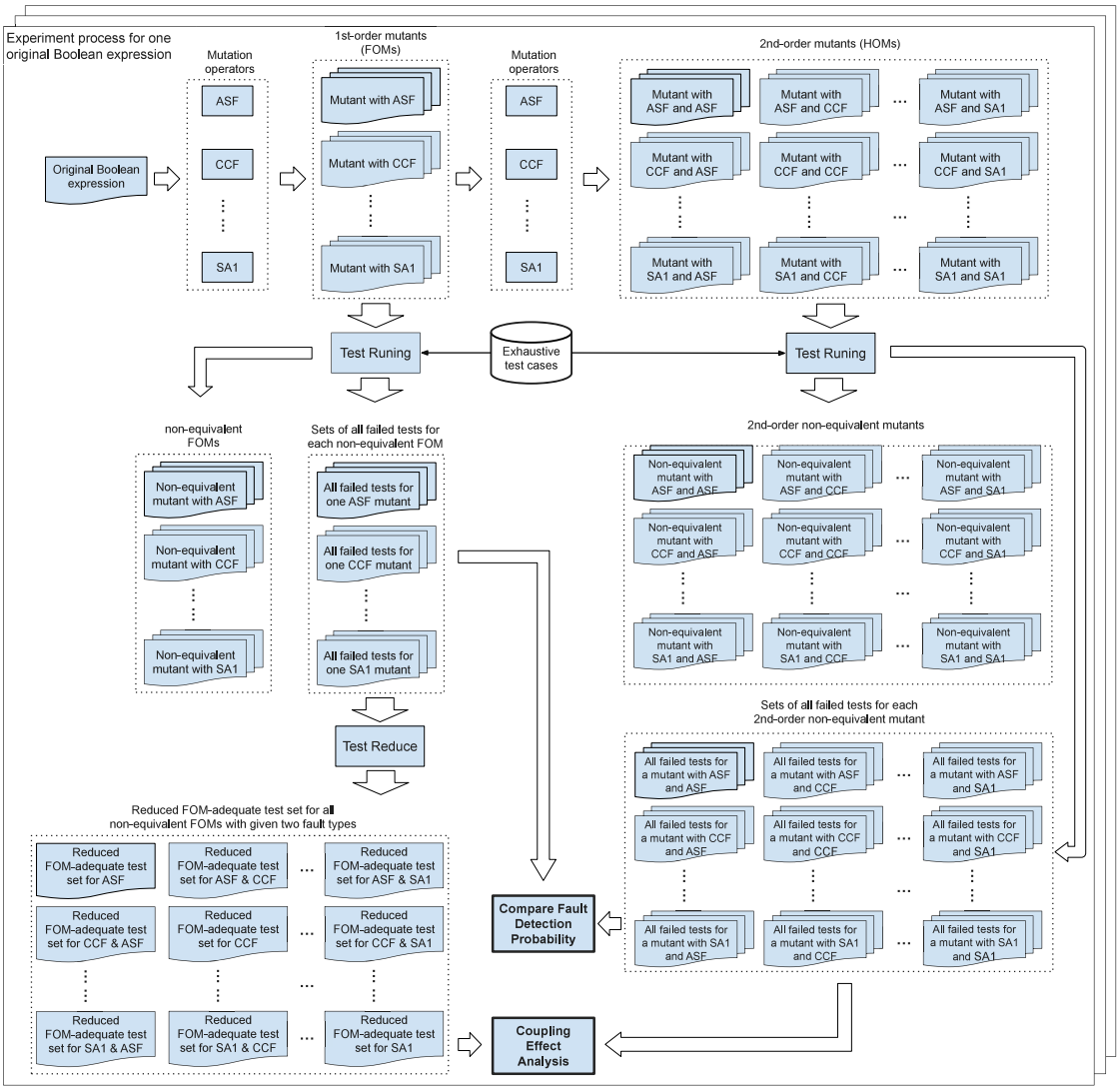


Fig. 1. Process of experiment for each original Boolean expression.

4.1. Results for RQ1

For each original Boolean expression, the contrast of detection probabilities of FOMs and 2nd-HOMs is made in Fig. 2. There are 40 groups of box-graphs in Fig. 2, where each group stands for an original Boolean expression extracted from the TCAS or NDCS system. The fault detection probabilities for FOMs are in the first box-graph of each group, and the second box-graph shows the fault detection probabilities for 2nd-HOMs.

For each fault type, Fig. 3(a) displays the comparison of fault detection probabilities for FOMs with the specified fault type and for 2nd-HOMs with this fault type as the first-order mutation operator. There are ten groups of box-graphs, where each group stands for a fault type, respectively representing detection probabilities of FOMs and 2nd-HOMs.

Considering ten fault types, we also compare the detection probabilities of all FOMs (without regard to types) and 2nd-HOMs with this type as the second mutant in Fig. 3(b). There are 11 groups of box-graphs in it. Besides the detection probabilities of all FOMs in the first box-graph, the last ten box-graphs in each group illustrate detection probabilities of 2nd-HOMs, which are generated by using the current type as the second mutant.

Fig. 2 indicates that, for all the original Boolean expressions, the 2nd-HOMs are slightly easier than FOMs to be detected. Fig. 3(a) indicates that, if the first-order fault type is CCF, CDF, LNF, LRF, MLF, SA0, and SA1, the 2nd-HOMs are easier than FOMs to be detected. And Fig. 3(b) indicates that, no matter what the second-order fault type is, the 2nd-HOMs are easier than FOMs to be detected.

As a part of the investigation, it is necessary to determine the statistical significance of any differences between values of the fault detection probabilities, for which there are many statistical tests, such as the Mann–Whiney Test. The fault detection probabilities for FOMs and 2nd-HOMs are two groups of parallel designs so they were two independent samples. The fault detection probabilities were divided into the intervention group (FOMs) and the comparison group (2nd-HOMs). Furthermore, because no assumptions were made about which the samples were normally distributed, the nonparametric test was used. Follow the previous guidelines on inferential statistical approaches for dealing with randomized algorithms, we used the 2-independent samples nonparametric Mann–Whiney Test to check the statistical significance (at the significance level of 5%).

Suppose X_1 is the fault detection probabilities for all the FOMs from an original expression, Y_1 is the fault detection probabilities for all the 2nd-HOMs form the same original expression:

- Null Hypothesis (H_0): $X_1 \geq Y_1$ means that the fault detection probabilities for FOMs tend to be higher than that for 2nd-HOMs.

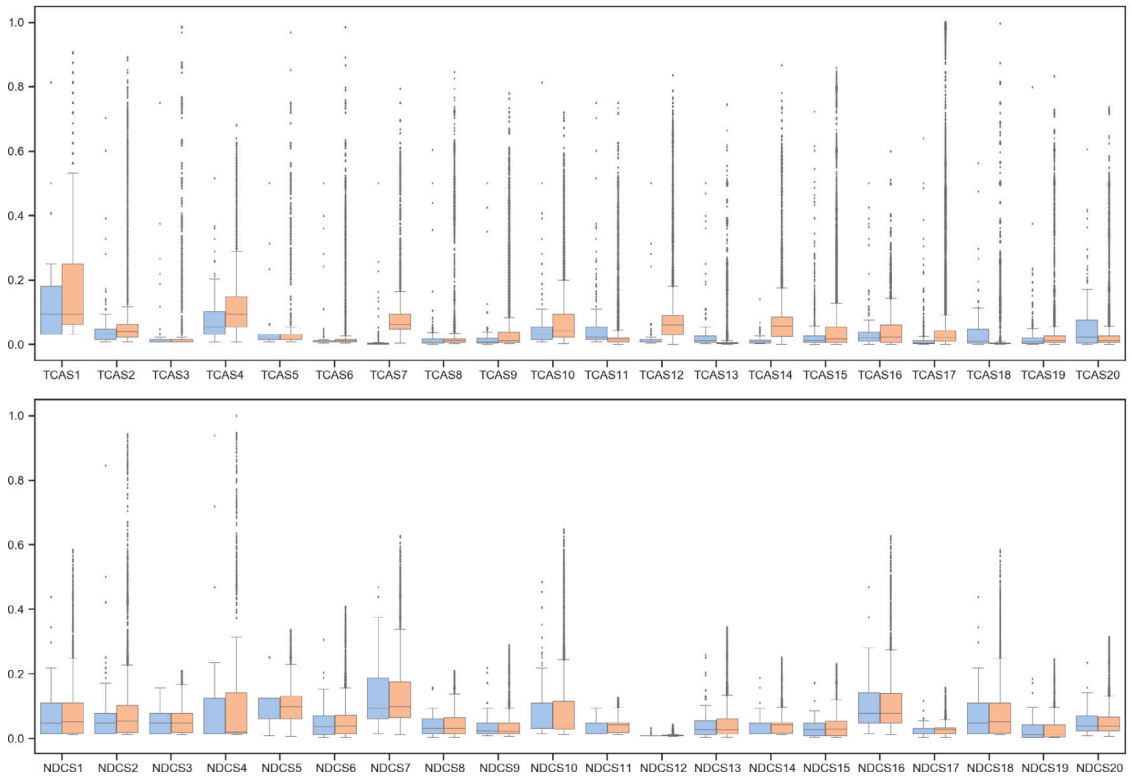
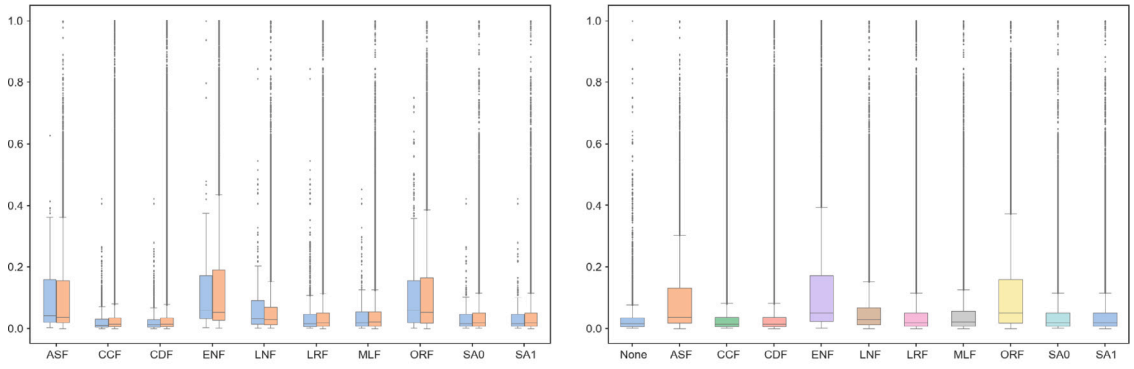


Fig. 2. Compare fault detection probabilities for FOMs and 2nd-HOMs (grouped for original expressions).



(a) Ten first-order fault types (b) Ten second-order fault types

Fig. 3. Compare fault detection probabilities for FOMs and 2nd-HOMs (grouped for ten fault types).

- Alternative Hypothesis (H_1): $X_1 < Y_1$ means that the fault detection probabilities for FOMs tend to be less than that for 2nd-HOMs.

The P-values for 40 original Boolean expressions are shown in Table 6. We refuse the null hypothesis, since all the P-values are less than 0.05. It means that, for each expression, the fault detection probabilities for FOMs are tend to be less than that for 2nd-HOMs significantly.

Suppose X_2 is the fault detection probabilities for all the FOMs with one of all the ten fault types, Y_2 is the fault detection probabilities for all the 2nd-HOMs in which the first-order fault type is the same as the fault type of those FOMs:

- Null Hypothesis (H_0): $X_2 \geq Y_2$ means that the fault detection probabilities for FOMs tend to be higher than that for 2nd-HOMs.
- Alternative Hypothesis (H_1): $X_2 < Y_2$ means that the fault detection probabilities for FOMs tend to be less than that for 2nd-HOMs.

The P-values for ten first-order mutation operators are shown in Table 7. We refuse the null hypothesis for 7 first-order fault types including `CCF`, `CDF`, `LNF`, `LRF`, `MLF`, `SAO` and `SA1`, since P-values for these 7 first-order fault types are less than 0.05. It means that, when first-order fault type is one of `CCF`, `CDF`, `LNF`, `LRF`, `MLF`, `SAO`, and `SA1`, the fault detection probabilities for FOMs are tend to be less than that for 2nd-HOMs significantly. However, for other fault types including `ASF`, `ENF` and `ORF`, the fault detection probabilities for FOMs are not significantly less than that for 2nd-HOMs.

Suppose X_3 is the fault detection probabilities for all the FOMs, Y_3 is the fault detection probabilities for all the 2nd-HOMs with one of all the ten second-order fault types:

- Null Hypothesis (H_0): $X_3 \geq Y_3$ means that the fault detection probabilities for FOMs tend to be higher than that for 2nd-HOMs.
- Alternative Hypothesis (H_1): $X_3 < Y_3$ means that the fault detection probabilities for FOMs tend to be less than that for 2nd-HOMs.

Table 5
Consumed time in experiment.

Expression	Testing FOMs	Testing HOMs	Analysis
TCAS1	<1 s	1.35 m	16.20 m
TCAS2	2 s	77.47 m	976.40 m
TCAS3	<1 s	13.32 m	173.20 m
TCAS4	<1 s	15.01 m	189.00 m
TCAS5	<1 s	6.28 m	97.80 m
TCAS6	2 s	50.15 m	662.40 m
TCAS7	<1 s	12.42 m	161.50 m
TCAS8	2 s	2925.15 m	38 423.20 m
TCAS9	2 s	66.26 m	861.40 m
TCAS10	<1 s	64.17 m	852.50 m
TCAS11	2 s	108.23 m	1261.30 m
TCAS12	<1 s	34.35 m	467.10 m
TCAS13	4 s	1231.25 m	16 505.60 m
TCAS14	<1 s	43.01 m	555.30 m
TCAS15	6 s	3524.97 m	46 264.30 m
TCAS16	4 s	77.02 m	1003.80 m
TCAS17	4 s	1878.65 m	24 835.80 m
TCAS18	<1 s	121.58 m	1556.20 m
TCAS19	2 s	302.03 m	398.32 m
TCAS20	4 s	437.18 m	5682.50 m
NDCS1	<1 s	5.26 m	65.31m
NDCS2	<1 s	5.15 m	67.40 m
NDCS3	<1 s	6.33 m	73.56 m
NDCS4	<1 s	5.09 m	66.89 m
NDCS5	2 s	68.02 m	897.07 m
NDCS6	1 s	10.51 m	264.20 m
NDCS7	2 s	104.12 m	1695.04 m
NDCS8	2 s	195.51 m	3834.22 m
NDCS9	2 s	266.44 m	2863.01 m
NDCS10	3s	774.76 m	6825.60 m
NDCS11	<1 s	8.41 m	263.11 m
NDCS12	1 s	53.40 m	467.10 m
NDCS13	3 s	239.52 m	3605.40 m
NDCS14	<1 s	13.04 m	93.15 m
NDCS15	2 s	397.74 m	4336.64 m
NDCS16	<1 s	7.20 m	83.00 m
NDCS17	3 s	658.68 m	8932.35 m
NDCS18	<1 s	10.34 m	68.21 m
NDCS19	2 s	206.00 m	4833.58 m
NDCS20	2 s	166.78 m	2155.33 m

Table 6
P-values for each original Boolean expressions.

Expression	P-value	Expression	P-value
TCAS1	1.09E−6	TCAS11	2.92E−49
TCAS2	6.08E−51	TCAS12	1.45E−48
TCAS3	8.13E−17	TCAS13	8.12E−59
TCAS4	4.86E−34	TCAS14	7.51E−45
TCAS5	6.77E−14	TCAS15	7.57E−110
TCAS6	2.49E−58	TCAS16	1.50E−32
TCAS7	5.31E−43	TCAS17	7.73E−99
TCAS8	4.67E−108	TCAS18	4.43E−47
TCAS9	1.47E−35	TCAS19	4.59E−58
TCAS10	6.49E−39	TCAS20	4.53E−85
NDCS1	6.77E−13	NDCS11	5.42E−40
NDCS2	7.79E−21	NDCS12	9.52E−23
NDCS3	7.70E−5	NDCS13	7.80E−39
NDCS4	1.04E−77	NDCS14	8.03E−49
NDCS5	3.61E−32	NDCS15	1.54E−16
NDCS6	2.99E−27	NDCS16	4.96E−5
NDCS7	4.36E−20	NDCS17	1.95E−68
NDCS8	7.05E−9	NDCS18	4.97E−38
NDCS9	3.79E−27	NDCS19	2.81E−21
NDCS10	3.85E−26	NDCS20	6.97E−3

The P-values for ten second-order fault types are shown in Table 8. We refuse the null hypothesis, since P-values for all the second-order fault types are less than 0.05. It means that, when the second-order fault type is ASF, CCF, CDF, ENF, LNF, LRF, ORF, MLF, SA0, and SA1, the fault detection probabilities for FOMs are tend to be less than that for 2nd-HOMs significantly.

Table 7
P-values for ten first-order mutation operators.

Type	*P*-value	Type	*P*-value
ASF	4.84E−1	LRF	1.74E−171
CCF	1.30E−157	ORF	7.17E−2
CDF	4.84E−133	MLF	2.46E−7
ENF	4.14E−1	SA0	6.00E−10
LNF	7.05E−3	SA1	2.81E−9

Table 8
P-values for ten second-order mutation operators.

Type	*P*-value	Type	*P*-value
ASF	0	LRF	0
CCF	2.89E−175	ORF	0
CDF	3.78E−58	MLF	0
ENF	0	SA0	0
LNF	0	SA1	0

4.2. Results for RQ2

For each original Boolean expression, the four test case reduction algorithms generate four different reduced FOM-adequate test sets for each combination of the first-order and the second-order fault types. Fig. 4 illustrates the percent of 2nd-HOMs killed by reduced FOM-adequate test sets. There are a total of ten sub-figures that show the cases when the first mutation operator is ASF, CCF, CDF, ENF, LNF, LRF, MLF, ORF, SA0, and SA1 respectively. In Fig. 4, there are ten groups of box-graphs in each sub-figure, where each group stands for a fault type that is the second mutant of the 2nd-HOMs. Each box-graph illustrates the percents of 2nd-HOMs that are killed by reduced test suites selected for FOMs.

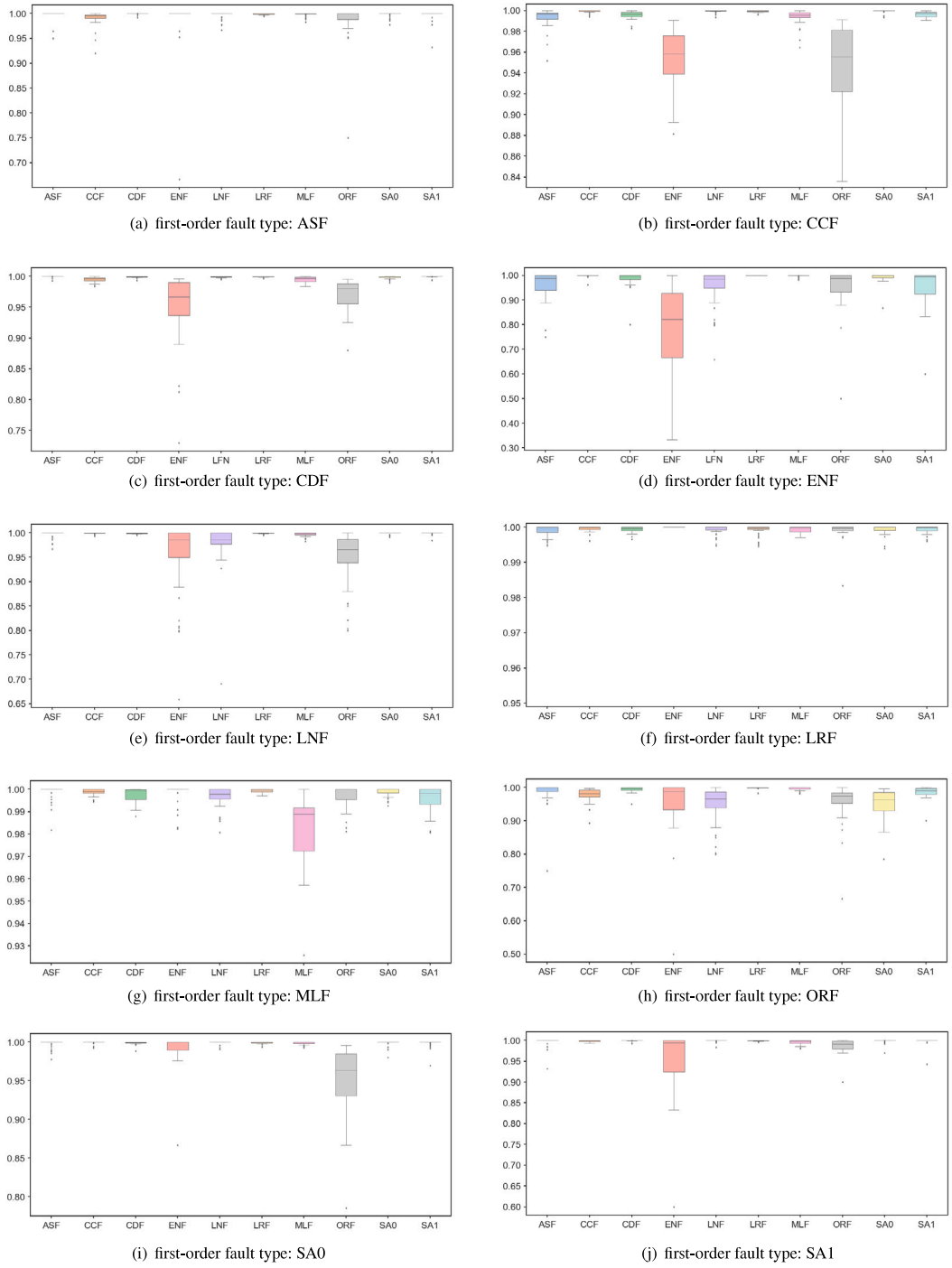
These results indicate that, for any fault type, the coupling effect cannot be guaranteed to exist with absolute certainty. Especially, when mutants involve fault types ENF or ORF, the ratio of the coupling effect existing is obviously lower.

Kapoor proved that the coupling effect holds on fault types ASF, CCF, CDF, LRF, MLF, ORF, SA0, and SA1 under the condition that "there can be at most one fault in leaves of a Boolean expression tree" (Kapoor, 2006). In other words, if a test set guarantees detection of FOMs with the above fault types then it guarantees the detection of corresponding HOMs. However, Kapoor's conclusion does not seem to work in our experiment, because the prerequisite condition is too strict to reflect all the mistakes made by programmers in the real scenarios. In fact, there is a possibility that programmers may make two or more mistakes at the same position in a Boolean expression.

For example, a FOM-adequate test set of TCAS1 ($x_1 \wedge (\neg x_2 \vee \neg x_3) \wedge x_4 \vee x_5$) for all the FOMs with fault types ASF and CCF is {00001, 00010, 10110, 11010, 11111}, but the set of all the failed test cases for one of 2nd-HOMs ($x_1 \wedge \neg x_2 \vee (\neg x_3 \wedge x_1) \wedge x_4 \vee x_5$), which is obtained by performing ASF firstly and CCF secondly on TCAS1, is {10000, 10100}. This 2nd-HOM cannot be killed by any test case in the reduced FOM-adequate test set. The reason is that two mistakes, including the omission of the pair of brackets and the replacement of $\neg x_3$ by $(\neg x_3 \wedge x_1)$, in this 2nd-HOM involve the same Boolean variable x_3.

4.3. Results for RQ3

According to the theory of coupling effect hypothesis, a complex mutants (such as HOM) are formed by coupling two or more simple mutants (such as FOM). The coupling effect hypothesis means that, if a test case can kill a simple mutant, it is also highly likely to kill a complex mutant. In the experiment of this paper, when we use a reduced test set that can kill all the FOMs to test those second-order HOMs, the probability of these HOMs being killed should be high. But



(a) first-order fault type: ASF
(b) first-order fault type: CCF
(c) first-order fault type: CDF
(d) first-order fault type: ENF
(e) first-order fault type: LNF
(f) first-order fault type: LRF
(g) first-order fault type: MLF
(h) first-order fault type: ORF
(i) first-order fault type: SA0
(j) first-order fault type: SA1

Fig. 4. Percent of 2nd-HOMs killed by reduced FOM-adequate test set.

if there are some second-order HOMs that cannot be killed by such a reduced FOM-adequate test set, it means that there are risks if omitting second-order mutation testing. Therefore, for 10×10 combinations of the first-order and the second-order fault types, we count the average percent of mutants that cannot be killed by reduced FOM-adequate test set for the corresponding first-order mutation operators. Obviously, the larger the value shown in Table 9, the greater the risk of omitting second-order mutation testing for Boolean-specifications.

It can be concluded from Table 9 that, nearly a quarter of 2nd-HOMs that contain two expression negation faults (ENF) cannot be killed by the reduced FOM-adequate test set for fault type ENF. In addition, when the fault type involves ENF or ORF (whether they are the first-order mutation operators or the second-order mutation operators), a relatively big number of HOMs cannot be killed by the reduced FOM-adequate test set. On the contrary, only when the first-order mutation operator is ENF and the second-order one is LRF, and the first-order mutation operator is LRF and the second-order one is ENF, the reduced FOM-adequate test set for ENF and LRF can kill all the HOMs. In addition, when the fault type involves LRF (whether it is the first-order mutation operator or the second-order mutation operator), most of the HOMs could be killed by the reduced FOM-adequate test set. These results are consistent with that shown in Fig. 4.

Table 9
Average percent of 2nd-HOMs that not killed by reduced FOM-adequate test set for each 1st/2nd-order fault type.

2nd-order mutation operator	1st-order mutation operator										
	ASF	CCF	CDF	ENF	LNF	LRF	MLF	ORF	SA0	SA1	All HOMs
ASF	0.00724	0.00966	0.00044	0.02450	0.00386	0.00103	0.00242	0.00161	0.00161	0.00238	0.00548
CCF	0.00681	0.00084	0.00454	0.05573	0.00075	0.00074	0.00605	0.05595	0.00064	0.00372	0.00358
CDF	0.00044	0.00534	0.00099	0.05968	0.00080	0.00062	0.00555	0.03156	0.00153	0.00042	0.00169
ENF	0.04849	0.00267	0.01878	0.24406	0.04598	0	0.00242	0.04977	0.01108	0.05472	0.04779
LNF	0.00386	0.00078	0.00082	0.04598	0.01818	0.00065	0.00342	0.05497	0.00052	0.00105	0.00132
LRF	0.00103	0.00057	0.00065	0	0.00065	0.00092	0.00071	0.00130	0.00063	0.00074	0.00072
MLF	0.00092	0.00146	0.00247	0.00242	0.00373	0.00065	0.00845	0.00320	0.00101	0.00471	0.00390
ORF	0.01629	0.02440	0.00665	0.04977	0.05497	0.00130	0.00309	0.04618	0.05239	0.01449	0.02695
SA0	0.00161	0.00071	0.00084	0.01108	0.00054	0.00067	0.00101	0.05239	0.00102	0.00122	0.00711
SA1	0.00238	0.00174	0.00047	0.05472	0.00108	0.00066	0.00476	0.01449	0.00122	0.00225	0.00838
All HOMs	0.00891	0.00482	0.00367	0.05479	0.00130	0.00072	0.00479	0.03114	0.00716	0.00857	0.00367

4.4. Findings and remarks

The answers to all the research questions could be obtained from experimental results in the previous contents:

- **Answer to RQ1**: It is slightly easier to kill 2nd-HOMs than FOMs generally, since fault detection probabilities for complex faults are slightly greater than that for simple faults for most cases.
- **Answer to RQ2**: For any fault type, the coupling effect cannot be guaranteed to exist with absolute certainty, since the percents of the 2nd-HOMs killed by the reduced FOM-adequate test suites are not 100%.
- **Answer to RQ3**: When HOMs involve fault types ENF or ORF, the percents of the coupling effect existing are obviously lower. Especially, when 2nd-HOMs contain two expression negation faults (ENF), there is a high risk that these mutants cannot be killed by the FOM-adequate test set.

Above answers could help us to determine do we need high-order mutation in fault-based Boolean-specification testing. In addition, based on these answers, we can also provide a series of hierarchical suggestions on which fault types should be focused on when performing high-order mutation testing for Boolean-specifications. Overall, the implications include:

- Due to the fact that the coupling effect hypothesis is not guaranteed to hold with absolute certainty for any fault type in Boolean-specifications, the necessity of adopting high-order mutants in Boolean-specification testing cannot be thoughtlessly ruled out. However, the use of high-order mutation testing should be cautious. Whether we need high-order mutation testing in practice, and how we perform high-order mutation testing in practice, should be determined based on the available testing resources.
- If the testing resources are not extremely sufficient, high-order mutation testing is not recommended in fault-based Boolean-specification testing. The main reason is the massive number of high-order mutants and the resulting huge consumed time. Another reason is that, although some 2nd-HOMs cannot be killed by the FOM-adequate test sets, their proportion in all the HOMs is tiny, even less than 1%.
- If the testing resources are sufficient enough, and there is a high requirement for fault detection effectiveness of test cases under the evaluation of mutation testing, high-order mutation testing could be performed for some selected specific fault types. Firstly, 2nd-HOMs that contain two expression negation faults (ENF) could be considered in mutation testing, as experimental results show that nearly a quarter of these mutants cannot be killed by the FOM-adequate test set. Secondly, 2nd-HOMs containing at least one expression negation fault (ENF) could also be considered (except for those with another fault being literal reference fault (LRF)). Thirdly, 2nd-HOMs containing at least one operator reference fault (ORF) could also be considered as supplements. Note that the number of mutants that involve ENF and ORF are significantly less than other fault types, so extra consumed testing resources for HOMs with ENF or ORF may be acceptable.
- Some HOMs with certain specific fault types are completely unnecessary in fault-based Boolean-specification testing, even if the testing resources are sufficient enough. Firstly, dropping those 2nd-HOMs with one expression negation fault (ENF) and one literal reference fault (LRF) is safe, as experimental results show that all of these mutants are killed by the FOM-adequate test set. Secondly, 2nd-HOMs containing at least one literal reference fault (LRF), it is likely to be safe not to use them in mutation testing. Thirdly, when HOMs involve ASF, LNF, MLF, SA0, and SA1, as long as these HOMs do not involve the fault types mentioned previously (ENF and ORF), they can also be omitted. Finally, we do not recommend using HOMs that contain clause conjunction fault (CCF) or clause disjunction fault (CDF), as these two fault types are introduced to investigate the relationship between fault types. When writing a general-form Boolean expression, the probability of programmers making such mistakes is very low.

4.5. Threats to validity

Threats to internal validity are concerned with the uncontrolled factors that may be responsible for the results. Test reduction algorithms, which were utilized when generating the mutation-adequate test set for FOMs, may output different results for the same input, and may affect the results of coupling effect analysis. To reduce this threat, we utilized four popular test reduction algorithms, including pure greedy algorithm (Chen and Lau, 2003), GE algorithm (Chen and Lau, 2003), GRE algorithm (Chen and Lau, 2003), and the algorithm proposed by Harrold et al. (1993), in our experiments. Another threat is the potential bugs in the tool for generating and testing mutants and the tool for calculating metric values of the fault detection probability and the percent of 2nd-HOMs Killed by FOM-adequate test set. All these tools are tested well before experiment.

Threats to external validity are concerned with whether the experimental results obtained in our experiment are generalizable for other situations. We select 20 general-form Boolean expressions extracted from the TCAS II system (Weyuker et al., 1994) as experimental subjects in our experiment. Such a set of Boolean expressions is one of the most popular benchmarks in the field of Boolean-specification testing (Tai, 1996; Chen et al., 1999; Yu and Lau, 2002; Kobayashi et al., 2002; Yu et al., 2006; Kaminski and Ammann, 2009; Arcaini et al., 2011; Gargantini and Fraser, 2011; Yu and Lau, 2012; Paul et al., 2014; Yu and Tsai, 2018; Paul et al., 2021). As a supplement, we also select 20 general-form Boolean expressions extracted from a nuclear industrial distributed control system (Zong et al., 2021) as experimental subjects. Some other experimental subjects have also been used in the field of Boolean-specification testing, but these subjects are either not

published or are not general-form (usually conjunctive normal form or disjunctive normal form) Boolean expressions, so they cannot be used in our experiment. Although the experimental subjects we have selected are limited, we believe that the experimental results still have a certain degree of representativeness.

5. Related works

The problem of the existence of the coupling effect has been studied for a long time. Lipton and Sayward used a program, called FIND, to illustrate that an adequate test set from FOMs was adequate for the samples of kth order mutants ($k = 2, \ldots, 4$) (Lipton and Sayward, 1978). Offutt also conducted an experiment to suggest that the test set that generated to kill first order mutants killed 99% second-order and third-order mutants (Offutt, 1992, 1989). Wah applied a theoretical model to indicate that the average survival ratio of FOMs is $1/n$ and the average survival ratio of 2nd-HOMs is $1/n^2$, where n is the order of the domain (Wah, 2000, 2003). Kapoor also proved that the coupling effect hypothesis held for some Boolean logic fault types (Kapoor, 2006).

Jia and Harman proposed that the high-order mutation testing could be utilized to reduce the cost of mutation testing (Jia and Harman, 2008; Harman et al., 2010; Langdon et al., 2010; Jia and Harman, 2009; Harman et al., 2011). It is possible to replace FOMs with a single HOM to reduce the number of mutants. Papadakis and Malevris evaluated the relative application cost and effectiveness of the first-order and the second-order mutation testing strategies (Papadakis and Malevris, 2010), and isolated first-order equivalent mutants via second-order mutation (Kintis et al., 2012). Polo et al.'s experiment demonstrated that 2nd-HOMs reduced test effort by 50 percent without losing the test effectiveness (Polo et al., 2009). Parsai and Murgia provided a model to estimate FOM coverage from HOM coverage (Parsai et al., 2016). Belli analyzed model-based high-order mutation (Belli et al., 2010). Langdon, Harman, and Jia researched multi-objective HOM with genetic programming (Langdon et al., 2009). Mateo, Usaola, and Alemn designed an empirical study to evaluate different combination strategies that compose two FOMs to a 2nd-HOM (Reales Mateo et al., 2013). Nguyen and Madeyski provided the problems of mutation testing and HOM testing (Nguyen and Madeyski, 2014). Lima and Vergilio reviewed mutant generation strategies and evaluation approaches of high-order mutation testing, and analyzed application trends and research opportunities (do Prado Lima and Vergilio, 2019). Liu et al. conducted empirical study to investigate the impact of FOMs and HOMs on the performance of mutation-based fault localization (Wang et al., 2022).

The double faults on the term and literal in Boolean expression have been studied by Lau et al. (2007b,a) and Lau et al. (2006). Kaminski and Ammann used logic criterion feasibility to reduce test set size in double fault detection (Kaminski and Ammann, 2009). Gopinath, Jensen, and Groce also researched the theory of composite faults (Gopinath et al., 2017).

6. Conclusions

The mutation testing technique is powerful for evaluating the effectiveness of a given test set. It could help to generate test cases with fault detection capabilities. Since the logical expressions in the predicate statements can directly affect the execution path of the program, Boolean-specification testing becomes one of the essential weak mutation testing techniques. As one of the two primary hypotheses in the field of mutation testing, the coupling effect hypothesis suggests that only the FOMs need to be considered in mutation testing. It means that whether the coupling effect hypothesis holds will determine whether we need to perform high-order mutation testing. We design an empirical study to answer the question of whether high-order mutation testing is required in fault-based Boolean-specification testing, by answering the question of whether the coupling effect hypothesis holds for mutants of general-form Boolean specifications. Experimental results tell us that the coupling effect hypothesis is not guaranteed to hold for all the ten fault types. And it is suggested that the high-order mutation testing could be carried out for ENF faults and ORF faults if the testing resource is sufficient.

As well known, high-order mutation testing can also be used to reduce the cost of mutation testing. After combining multiple FOMs containing different faults into a single HOM containing these faults, the number of mutants that need to be executed will be significantly reduced if test cases that kill the HOM also kill the corresponding FOMs. Therefore, in the fault-based Boolean-specification testing, it is necessary to study what strategy should be adopted to merge multiple FOMs into one HOM in future works.

CRediT authorship contribution statement

Ziyuan Wang: Methodology, Software, Data curation, Investigation, Validation, Writing – review & editing, Supervision. **Min Yu:** Data curation, Investigation, Validation, Writing – original draft. **Yang Feng:** Methodology, Writing – review & editing. **Weifeng Zhang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work was supported by theNational Natural Science Foundation of China under Grant 62272214.

References

Acree, A.T., Budd, T.A., DeMillo, R.A., Lipton, R.J., Sayward, F.G., 1979. Mutation Analysis. Technical Report, GIT-ICS-79/08, Georgia Institute of Technology.

Arcaini, P., Gargantini, A., Riccobene, E., 2011. Optimizing the automatic test generation by SAT and SMT solving for boolean expressions. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). pp. 388–391. http://dx.doi.org/10.1109/ASE.2011.6100079.

Belli, F., Güler, N., Hollmann, A., Suna, G., Yıldız, E., 2010. Model-based higher-order mutation analysis. In: Kim, T.-h., Kim, H.-K., Khan, M.K., Kiumi, A., Fang, W.-c., Ślęzak, D. (Eds.), Advances in Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 164–173. http://dx.doi.org/10.1007/978-3-642-17578-7_17.

Chekam, T.T., Papadakis, M., Le Traon, Y., Harman, M., 2017. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). pp. 597–608. http://dx.doi.org/10.1109/ICSE.2017.61.

Chen, Z., Chen, T.Y., Xu, B., 2011. A revisit of fault class hierarchies in general boolean specifications. ACM Trans. Softw. Eng. Methodol. 20 (3), http://dx.doi.org/10.1145/2000791.2000797.

Chen, T., Lau, M., 2003. On the divide-and-conquer approach towards test suite reduction. Inform. Sci. 152, 89–119. http://dx.doi.org/10.1016/S0020-0255(03)00060-4.

Chen, T., Lau, M., Yu, Y., 1999. MUMCUT: a fault-based strategy for testing boolean specifications. In: Proceedings Sixth Asia Pacific Software Engineering Conference (ASPEC'99) (Cat. No.PR00509). pp. 606–613. http://dx.doi.org/10.1109/APSEC.1999.809656.

DeMillo, R., Lipton, R., Sayward, F., 1978. Hints on test data selection: Help for the practicing programmer. Computer 11 (4), 34–41. http://dx.doi.org/10.1109/C-M.1978.218136.

Gargantini, A., Fraser, G., 2011. Generating minimal fault detecting test suites for general boolean specifications. Inf. Softw. Technol. 53 (11), 1263–1273. http://dx.doi.org/10.1016/j.infsof.2011.06.008, AMOST 2010.

Gopinath, R., Jensen, C., Groce, A., 2017. The theory of composite faults. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST). pp. 47–57. http://dx.doi.org/10.1109/ICST.2017.12.

Harman, M., Jia, Y., Langdon, W.B., 2010. A manifesto for higher order mutation testing. In: 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. pp. 80–89. http://dx.doi.org/10.1109/ICSTW.2010.13.

Harman, M., Jia, Y., Langdon, W.B., 2011. Strong higher order mutation-based test data generation. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. In: ESEC/FSE '11, Association for Computing Machinery, New York, NY, USA, pp. 212–222. http://dx.doi.org/10.1145/2025113.2025144.

Harrold, M.J., Gupta, R., Soffa, M.L., 1993. A methodology for controlling the size of a test suite. ACM Trans. Softw. Eng. Methodol. 2 (3), 270–285. http://dx.doi.org/10.1145/152388.152391.

Horgan, J.R., Mathur, A.P., 1990. Weak Mutation is Probably Strong Mutation. Technical Report, SERC-TR-83-P. Purdue University.

Howden, W., 1982. Weak mutation testing and completeness of test sets. IEEE Trans. Softw. Eng. SE-8 (4), 371–379. http://dx.doi.org/10.1109/TSE.1982.235571.

Jia, Y., Harman, M., 2008. Constructing subtle faults using higher order mutation testing. In: 2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation. pp. 249–258. http://dx.doi.org/10.1109/SCAM.2008.36.

Jia, Y., Harman, M., 2009. Higher order mutation testing. Inf. Softw. Technol. 51 (10), 1379–1393. http://dx.doi.org/10.1016/j.infsof.2009.04.016.

Jia, Y., Harman, M., 2011. An analysis and survey of the development of mutation testing. IEEE Trans. Softw. Eng. 37 (5), 649–678. http://dx.doi.org/10.1109/TSE.2010.62.

Kaminski, G., Ammann, P., 2009. Using logic criterion feasibility to reduce test set size while guaranteeing double fault detection. In: 2009 International Conference on Software Testing, Verification, and Validation Workshops. pp. 167–176. http://dx.doi.org/10.1109/ICSTW.2009.13.

Kaminski, G.K., Ammann, P., 2009. Using logic criterion feasibility to reduce test set size while guaranteeing fault detection. In: 2009 International Conference on Software Testing Verification and Validation. pp. 356–365. http://dx.doi.org/10.1109/ICST.2009.14.

Kapoor, K., 2006. Formal analysis of coupling hypothesis for logical faults. Innov. Syst. Softw. Eng. 2 (2), 80–87. http://dx.doi.org/10.1007/s11334-006-0002-z.

Kapoor, K., Bowen, J.P., 2007. Test conditions for fault classes in boolean specifications. ACM Trans. Softw. Eng. Methodol. 16 (3), 10–es. http://dx.doi.org/10.1145/1243987.1243988.

Kintis, M., Papadakis, M., Malevris, N., 2012. Isolating first order equivalent mutants via second order mutation. In: 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation. pp. 701–710. http://dx.doi.org/10.1109/ICST.2012.160.

Kobayashi, N., Tsuchiya, T., Kikuno, T., 2002. Non-specification-based approaches to logic testing for software. Inf. Softw. Technol. 44 (2), 113–121. http://dx.doi.org/10.1016/S0950-5849(01)00222-1.

Kuhn, D.R., 1999. Fault classes and error detection capability of specification-based testing. ACM Trans. Softw. Eng. Methodol. 8 (4), 411–424. http://dx.doi.org/10.1145/322993.322996.

Langdon, W.B., Harman, M., Jia, Y., 2009. Multi objective higher order mutation testing with genetic programming. In: 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques. pp. 21–29. http://dx.doi.org/10.1109/TAICPART.2009.18.

Langdon, W.B., Harman, M., Jia, Y., 2010. Efficient multi-objective higher order mutation testing with genetic programming. J. Syst. Softw. 83 (12), 2416–2430. http://dx.doi.org/10.1016/j.jss.2010.07.027.

Lau, M.F., Liu, Y., Chen, T.Y., Yu, Y.T., 2007a. On detecting double literal faults in boolean expressions. In: Abdennadher, N., Kordon, F. (Eds.), Reliable Software Technologies – Ada Europe 2007. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 55–68. http://dx.doi.org/10.1007/978-3-540-73230-3_5.

Lau, M.F., Liu, Y., Yu, Y.T., 2006. On detection conditions of double FaultsRelated to terms in boolean expressions. In: 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Vol. 1. pp. 403–410. http://dx.doi.org/10.1109/COMPSAC.2006.63.

Lau, M.F., Liu, Y., Yu, Y.T., 2007b. Detecting double faults on term and literal in boolean expressions. In: Seventh International Conference on Quality Software (QSIC 2007). pp. 117–126. http://dx.doi.org/10.1109/QSIC.2007.4385487.

Lau, M.F., Yu, Y.T., 2005. An extended fault class hierarchy for specification-based testing. ACM Trans. Softw. Eng. Methodol. 14 (3), 247–276. http://dx.doi.org/10.1145/1072997.1072998.

Lipton, R.J., Sayward, F.G., 1978. The status of research on program mutation. In: Proceedings of the Workshop Software Testing and Test Documentation. pp. 355–373.

Morell, L.J., 1983. A Theory of Error-Based Testing (Ph.D. thesis). University of Maryland at College Park, USA, AAI8419537.

Nguyen, Q.V., Madeyski, L., 2014. Problems of mutation testing and higher order mutation testing. In: van Do, T., Thi, H.A.L., Nguyen, N.T. (Eds.), Advanced Computational Methods for Knowledge Engineering. Springer International Publishing, Cham, pp. 157–172. http://dx.doi.org/10.1007/978-3-319-06569-4_12.

Offutt, A., 1989. The coupling effect: Fact or fiction. SIGSOFT Softw. Eng. Notes 14 (8), 131–140. http://dx.doi.org/10.1145/75309.75324.

Offutt, A.J., 1992. Investigations of the software testing coupling effect. ACM Trans. Softw. Eng. Methodol. 1 (1), 5–20. http://dx.doi.org/10.1145/125489.125473.

Offutt, A.J., Lee, S.D., 1991. How strong is weak mutation? In: Proceedings of the Symposium on Testing, Analysis, and Verification. In: TAV4, Association for Computing Machinery, New York, NY, USA, pp. 200–213. http://dx.doi.org/10.1145/120807.120826.

Offutt, A., Lee, S., 1994. An empirical evaluation of weak mutation. IEEE Trans. Softw. Eng. 20 (5), 337–344. http://dx.doi.org/10.1109/32.286422.

Papadakis, M., Henard, C., Traon, Y.L., 2014. Sampling program inputs with mutation analysis: Going beyond combinatorial interaction testing. In: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. pp. 1–10. http://dx.doi.org/10.1109/ICST.2014.11.

Papadakis, M., Kintis, M., Zhang, J., Jia, Y., Traon, Y.L., Harman, M., 2019. In: Memon, A.M. (Ed.), Mutation Testing Advances: An Analysis and Survey. In: Advances in Computers, vol. 112, Elsevier, pp. 275–378. http://dx.doi.org/10.1016/bs.adcom.2018.03.015.

Papadakis, M., Malevris, N., 2010. An empirical evaluation of the first and second order mutation testing strategies. In: 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. pp. 90–99. http://dx.doi.org/10.1109/ICSTW.2010.50.

Parsai, A., Murgia, A., Demeyer, S., 2016. A model to estimate first-order mutation coverage from higher-order mutation coverage. In: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 365–373. http://dx.doi.org/10.1109/QRS.2016.48.

Paul, T.K., Chowdhury, M.J.M., Lau, M.F., 2021. A new disjunctive literal insertion fault detection strategy in boolean specifications. J. Softw.: Evol. Process 33 (5), e2336. http://dx.doi.org/10.1002/smr.2336.

Paul, T.K., Lau, M.F., Ng, S., 2014. On a new detecting technique for conjunctive literal insertion fault in boolean expressions. In: 2014 14th International Conference on Quality Software. pp. 266–275. http://dx.doi.org/10.1109/QSIC.2014.41.

Polo, M., Piattini, M., García-Rodríguez, I., 2009. Decreasing the cost of mutation testing with second-order mutants. Softw. Test. Verif. Reliab. 19 (2), 111–131. http://dx.doi.org/10.1002/stvr.392.

do Prado Lima, J.A., Vergilio, S.R., 2019. A systematic mapping study on higher order mutation testing. J. Syst. Softw. 154, 92–109. http://dx.doi.org/10.1016/j.jss.2019.04.031.

Reales Mateo, P., Polo Usaola, M., Fernández Alemán, J.L., 2013. Validating second-order mutation at system level. IEEE Trans. Softw. Eng. 39 (4), 570–587. http://dx.doi.org/10.1109/TSE.2012.39.

Sánchez, A.B., Delgado-Pérez, P., Medina-Bulo, I., Segura, S., 2022. Mutation testing in the wild: findings from GitHub. Empir. Softw. Eng. 27, http://dx.doi.org/10.1007/s10664-022-10177-8.

Tai, K.-C., 1996. Theory of fault-based predicate testing for computer programs. IEEE Trans. Softw. Eng. 22 (8), 552–562. http://dx.doi.org/10.1109/32.536956.

Tsuchiya, T., Kikuno, T., 2002. On fault classes and error detection capability of specification-based testing. ACM Trans. Softw. Eng. Methodol. 11 (1), 58–62. http://dx.doi.org/10.1145/504087.504089.

Wah, K.S.H.T., 2000. A theoretical study of fault coupling. Softw. Test. Verif. Reliab. 10 (1), 3–45. http://dx.doi.org/10.1002/(SICI)1099-1689(200003)10:1<3::AID-STVR196>3.0.CO;2-P.

Wah, K.S.H.T., 2003. An analysis of the coupling effect I: single test data. Sci. Comput. Program. 48 (2), 119–161. http://dx.doi.org/10.1016/S0167-6423(03)00022-4.

Wang, H., Li, Z., Liu, Y., Chen, X., Paul, D., Cai, Y., Fan, L., 2022. Can higher-order mutants improve the performance of mutation-based fault localization? IEEE Trans. Reliab. 71 (2), 1157–1173. http://dx.doi.org/10.1109/TR.2022.3162039.

Wang, Z., Yu, M., 2018. BoolMuTest: A prototype tool for fault-based boolean-specification testing. In: The 30th International Conference on Software Engineering and Knowledge Engineering (SEKE 2008). pp. 720–721. http://dx.doi.org/10.18293/SEKE2018-224.

Weyuker, E., Goradia, T., Singh, A., 1994. Automatically generating test data from a boolean specification. IEEE Trans. Softw. Eng. 20 (5), 353–363. http://dx.doi.org/10.1109/32.286420.

Wong, W.E., Mathur, A.P., 1995. Fault detection effectiveness of mutation and data flow testing. Softw. Qual. J. 4, 69–83. http://dx.doi.org/10.1007/BF00404650.

Yu, Y.T., Lau, M.F., 2002. Prioritization of test cases in MUMCUT test sets: An empirical study. In: Blieberger, J., Strohmeier, A. (Eds.), Reliable Software Technologies — Ada-Europe 2002. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 245–256. http://dx.doi.org/10.1007/3-540-48046-3_19.

Yu, Y.T., Lau, M.F., 2012. Fault-based test suite prioritization for specification-based testing. Inf. Softw. Technol. 54 (2), 179–202. http://dx.doi.org/10.1016/j.infsof.2011.09.005.

Yu, Y.T., Lau, M.F., Chen, T.Y., 2006. Automatic generation of test cases from boolean specifications using the MUMCUT strategy. J. Syst. Softw. 79 (6), 820–840. http://dx.doi.org/10.1016/j.jss.2005.08.016.

Yu, L., Tsai, W.-T., 2018. Test case generation for boolean expressions by cell covering. IEEE Trans. Softw. Eng. 44 (1), 70–99. http://dx.doi.org/10.1109/TSE.2017.2669184.

Zong, C., Zhang, Y., Yao, Y., Shuang, S., Wang, Z., 2021. A comparison of fault detection efficiency between adaptive random testing and greedy combinatorial testing for control logics in nuclear industrial distributed control systems. IEEE Access 9, 84021–84033. http://dx.doi.org/10.1109/ACCESS.2021.3087165.

Ziyuan Wang received the B.S. degree in mathematics and the Ph.D. degree in computer science from Southeast University, Nanjing, China, in 2004 and 2009, respectively. From 2009 to 2012, he worked as a Postdoctoral Researcher with the Department of Computer Science and Technology, Nanjing University, Nanjing, China. He is currently an Associate Professor with the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, China. His research interests include software testing.

Min Yu received the M.S. degree in software engineering from Nanjing University of Posts and Telecommunications, Nanjing, China, in 2019. She is currently engaged in software research and development. Her research interests include software testing and software system management.

Yang Feng received the Ph.D. degree in software engineering from University of California, Irvine, USA, in 2019. He is currently an Assistant Professor with the Department of Computer Science and Technology, Nanjing University, Nanjing, China. His research interests include software testing and automatic program debugging.

Weifeng Zhang received the Ph.D. degree in computer science from Southeast University, Nanjing, China, in 2000. He is currently a Professor with the School of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, China. His research interests include software engineering.