On Detection Conditions of Double Faults Related to Terms in Boolean Expressions*

M. F. Lau† and Y. Liu
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria, AUSTRALIA, 3122
{elau,yliu}@ict.swin.edu.au

Y. T. Yu
Department of Computer Science
City University of Hong Kong
Kowloon Tong, HONG KONG
csytyu@cityu.edu.hk

Abstract

Detection conditions of specific classes of faults have recently been studied by many researchers. Under the assumption that at most one of these faults occurs in the software under test, these fault detection conditions were mainly used in two ways. First, they were used to develop test case selection strategies for detecting corresponding classes of faults. Second, they were used to study fault class hierarchies, where a test case that detects a particular class of faults can also detect some other classes of faults. In this paper, we study detection conditions of double faults. Besides developing new test case selection strategies and studying new fault class hierarchies, our analysis provides further insights to the effect of fault coupling. Moreover, these fault detection conditions can be used to compare effectiveness of existing test case selection strategies (which were originally developed for the detection of single occurrence of certain classes of faults) in detecting double faults that may be present in the software.

1. Introduction

The aim of software testing is to reveal failures in programs which are developed based on their specifications. Normally, testing is done by executing the program with selected input values and observing the actual output. If the input values and the actual output do not match the specifications, the program fails. Fault-based testing strategies have been proposed to detect hypothesized faults in a program. If a hypothesized fault has been injected into the program, test cases generated by fault-based testing strategies can reveal program failures due to the hypothesized fault.

Detection conditions of hypothesized faults have been studied for many years [1, 4, 8, 15] and mainly used in two ways. First, they were used to develop test case selection strategies aiming at the detection of pre-defined classes of faults. Chen and Lau [1] proposed three test case selection strategies based on detection conditions of seven classes of faults. Second, fault detection conditions were used to study fault class hierarchies [4, 8, 15]. A fault class hierarchy establishes relationships between different classes of faults. If test cases that detect faults of class A can always detect faults of class B, A is represented in a position in the fault class hierarchy lower than that of B. Kuhn [4] and Lau and Yu [8] used fault class hierarchies to explain previous empirical results for existing fault-based testing strategies.

Most work on fault detection conditions assumes that only one of the hypothesized faults occurs in the software under test, but multiple faults may also occur in programs [10, 16]. Previous studies on multiple faults [3, 12] mainly consider double faults and fault coupling effects, and they are mostly empirical in nature, while this paper analyses fault detection conditions analytically. Lau and Yu [7] explore the use of fault class hierarchy to detect double faults related to terms. They find that a test case that detects some classes of fault in the lower part of a fault class hierarchy can also detect double faults which involve the classes of faults in the upper part of the hierarchy. For example, if a test case detects single occurrence of a fault of class A which is at a lower position than that of class B in the hierarchy, the same test case will also be able to detect the double fault involving both fault classes A and B.

In this paper, we study detection conditions of double faults related to terms. The analysis of detection conditions of double faults not only helps developing test case selection strategies and fault class hierarchies, but also provides insights to the study of fault coupling. More specifically, we identify classes of double faults which are guaranteed to be detected by test cases that detect
certain classes of single faults. Moreover, based on these detection conditions, we can compare the effectiveness of existing test case selection strategies (originally developed to detect single occurrence of certain classes of faults) in detecting double faults.

The rest of this paper is organized as follows. Section 2 introduces the notation and fault classes studied in this paper. Section 3 presents double faults and their detection conditions. Sections 4 and 5 analyse the effects of fault coupling between single and double faults, and the effectiveness of existing test case selection strategies in detecting double faults. Finally, Section 6 concludes the paper and discusses future work.

2. Preliminary

The readers are assumed to be familiar with the basics of Boolean algebra, and may refer to [9] if necessary. This section introduces the notation used and the fault classes considered in this paper.

2.1. Notation

We use '·', '+' and '¬' to represent the Boolean operators, AND, OR and NOT, respectively. Usually, '·' is omitted whenever it is clear from the context. A literal in a Boolean expression is an occurrence of a Boolean variable in the expression. We use 1 and 0 to represent the truth values 'TRUE' and 'FALSE', respectively. The set of all truth values, that is \{0, 1\}, is denoted as \( \mathbb{B} \).

Let \( S \) be a Boolean expression in irredundant disjunctive normal form (IDNF) (that is, disjunctive normal form with no redundant term or literal), given by

\[
S = p_1 + \cdots + p_m
\]

where \( m \) is the number of terms, \( p_i = x_{i1} \cdots x_{ik_i} \) is the \( i \)-th term of \( S \), \( x_{ij} \) is the \( j \)-th literal in \( p_i \), and \( k_i \) is the number of literals in \( p_i \). If \( S \) has \( n \) variables, the input domain is the \( n \)-dimensional Boolean space \( \mathbb{B}^n \). A test case for \( S \) is thus a point in \( \mathbb{B}^n \).

A true point of the term \( p_i \) in \( S \) is a point that makes \( p_i \) evaluate to 1. The set of all true points of \( p_i \) in \( S \) is denoted by \( TP_i(S) \). True points of \( S \) are those that cause \( S \) to evaluate to 1. The set of all true points of \( S \) is denoted by \( TP(S) \) and given by \( TP(S) = \bigcup TP_i(S) \).

A unique true point of \( p_i \) in \( S \) is a point that makes (1) \( p_i \) evaluate to 1, and (2) all other terms evaluate to 0. The set of all unique true points of \( p_i \) in \( S \) is denoted by \( UTP_i(S) \). The set of all unique true points of \( S \) is denoted by \( UTP(S) \) and given by \( UTP(S) = \bigcup UTP_i(S) \).

False points of \( S \) are those that make \( S \) evaluate to 0. The set of all false points of \( S \) is denoted by \( FP(S) \). A near false point of the \( j \)-th literal \( x_{ij} \) of the \( i \)-th term \( p_i \) in \( S \) is a point that makes (1) \( x_{ij} \) evaluate to 0, (2) all literals in \( p_i \) other than \( x_{ij} \) evaluate to 1, and (3) \( S \) evaluate to 0. The set of all near false points of the \( j \)-th literal \( x_{ij} \) of the \( i \)-th term \( p_i \) in \( S \) is denoted by \( NFP_{ij}(S) \). The set of all near false points of \( p_i \) in \( S \) is denoted by \( NFP_i(S) \) and given by \( NFP_i(S) = \bigcup NFP_{ij}(S) \). Similarly, the set of all near false points of \( S \) is denoted by \( NFP(S) \) and given by \( NFP(S) = \bigcup NFP_i(S) \).

2.2. Fault Class

The five single fault classes below are related to terms in a Boolean expression and considered in this paper:

1. Expression Negation Fault (ENF): The entire Boolean expression or its subexpression is wrongly implemented as its negation.
2. Term Negation Fault (TNF): A term in the Boolean expression is wrongly implemented as its negation.
3. Term Omission Fault (TOF): A term in the Boolean expression is wrongly omitted in its implementation.
4. Disjunctive Operator Reference Fault (DORF): The Boolean operator '+' between two consecutive terms is wrongly implemented as '·'.
5. Conjunctive Operator Reference Fault (CORF): The Boolean operator '·' between two consecutive literals in a term is wrongly implemented as '+'.

Given a Boolean expression, an occurrence of any fault of the above classes may result in a faulty expression that differs from the original (correct) one by a syntactic change. For ease of reference, such a faulty expression is called a single-fault expression. The detection condition of a fault class \( A \), denoted by \( DC_A \), is the logical condition such that any test case satisfying it is guaranteed to detect every single occurrence of faults of class \( A \). In other words, a test case satisfying \( DC_A \) guarantees to distinguish the original (correct) Boolean expression from the resulting expression due to an occurrence of a fault of class \( A \). In such situations, we shall simply say that \( DC_A \) guarantees to detect fault class \( A \).

Table 1 lists these five fault classes, their corresponding faulty expressions and detection conditions. For example, the second row of the table represents the situation where a TNF is committed. If the term \( p_i \) in the correct expression \( S = p_1 + \cdots + p_m \) is negated, the single-fault expression \( I' \) is equivalent to \( p_1 + \cdots + p_i-1 + \bar{p_i} + p_{i+1} + \cdots + p_m \). The corresponding detection condition indicates that any unique true point in \( UTP_i(S) \) or any false point in \( FP(S) \) can distinguish \( S \) and \( I' \).

3. Double Fault Class

Similar to the situations for single fault classes, multiple occurrences of faults may result in a faulty expression that differs from the original (correct) expression by...
several syntactic changes. The resulting expression is said to contain multiple faults. In this paper, we focus on double faults, defined as two occurrences of single faults which may be of the same or different classes.

Note that when two single faults are committed, the resulting expression may sometimes be equivalent to the original expression, or to a single-fault expression. For example, if the first term of $S$ is negated twice, the resulting expression, $\overline{\overline{p_1}} + p_2 + \cdots + p_m$, is equivalent to $S$. On the other hand, if the first term of $S$ is first negated and then removed, the resulting expression, $p_2 + \cdots + p_m$, is equivalent to a single-fault expression where the first term is omitted. In this paper, we will only consider double-fault expressions. A double-fault expression is one which (1) differs from the original expression by two syntactic changes, (2) is neither equivalent to the original expression, nor equivalent to any expression that results from single faults of the classes discussed in Section 2.

Lau and Liu [5] have shown that there are two different ways in which double faults can be manifested. First, the two faults may be independent of each other. We refer to such a case as a double fault without ordering. Second, the two faults are considered to occur one after the other in such a way that the first fault may affect the occurrence of the second. For example, when a term is omitted first, it cannot be negated afterwards. Such a case is referred to as double fault with ordering.

Of the five single fault classes studied in this paper, there are 15 classes of double faults without ordering, resulting in 27 possible distinct faulty expressions [5]. For the case of double faults with ordering, there are 25 classes of double faults, resulting in 53 possible faulty expressions. Lau and Liu [5] find that 49 out of these 53 double-fault expressions are equivalent to some of the expressions that are results of double faults without ordering. Only 4 faulty expressions due to double faults with ordering do not have their equivalent counterparts in double faults without ordering [5]. In this paper, we will study all these 31 double-fault expressions.

For any two single fault classes $A$ and $B$, we use the notation $A \bowtie B$ to denote the double fault class formed from $A$ and $B$, that is, the class of faults due to the occurrences of two faults: one fault of class $A$ and another fault of class $B$. Due to page limitation, Table 3 lists the double-fault expressions and their detection conditions of the double fault classes $A \bowtie B$. Consider the first row of Table 3, which is concerned with $ENF \bowtie TOF$. Let $S$ be a Boolean expression in the class of double faults in $TNF$. Suppose that the subexpression $p_1 + \cdots + p_{h_1}$ ($i_1 < h_1$) in $S$ is wrongly negated and the $i_2$-th term $p_{i_2}$ in $S$ is wrongly omitted. There are two subcases.

First, the term $p_{i_2}$ is not contained in the subexpression $p_1 + \cdots + p_{h_1}$. Without loss of generality, we may assume that $h_1 < i_2$. The double-fault expression is equivalent to $p_1 + \cdots + p_{i_1} + p_{i_1+1} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m$. The detection condition shows that any true point in $(\bigcup_{i=i_1}^{h_1} TP_i(S)) \setminus \bigcup_{i=i_1}^{m} TP_i(S)$ or any false point of $S$ can distinguish $S$ and the double-fault expression.

Table 1: Single fault, single-fault expression and detection condition ($S = p_1 + \cdots + p_m$)

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Single-fault Expression</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENF</td>
<td>$p_1 + \cdots + p_{i_1-1} + \overline{p_{i_1}} + \cdots + p_{i_1} + p_{i_1+1} + \cdots + p_m$</td>
<td>Any point in $(\bigcup_{i=i_1}^{h_1} TP_i(S)) \setminus \bigcup_{i=i_1}^{h_1} TP_i(S)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td>TNF</td>
<td>$p_1 + \cdots + p_{i_1} + \cdots + p_m$</td>
<td>Any point in $UTP_{i_1}(S)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td>TOF</td>
<td>$p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_m$</td>
<td>Any point in $UTP_{i_1}(S)$</td>
</tr>
<tr>
<td>DORF</td>
<td>$p_1 + \cdots + p_{i_1} p_{i_1+1} + \cdots + p_m$</td>
<td>Any point in $UTP_{i_1}(S)$ or any point in $UTP_{i_1+1}(S)$</td>
</tr>
<tr>
<td>CORF</td>
<td>$p_1 + \cdots + p_{i_1-1} + p_{i_1} j_1 + p_{i_1} j_1 + k_{i_1} + p_{i_1+1} + \cdots + p_m$</td>
<td>Any point in $FP(S)$ such that $p_{i_1} j_1 + p_{i_1} j_1 + k_{i_1} = 1$</td>
</tr>
</tbody>
</table>

$^a$When the entire Boolean expression $S$ is negated, the resulting expression is $\overline{S}$ and it can be distinguished from $S$ by any point in $B^a$. Here, the negation of the entire Boolean expression is included as a special case of the negation of the subexpression $p_1 + p_2 + \cdots + p_m$ when $i_1 = 1$ and $p_1 = m$.

$^b$The detection condition of TNF can also be expressed as “any point such that $p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_m = 0$”. However, for ease of reference and discussions in later sections, we choose to present it in the form shown here which is consistent with those of other faults.
Table 2: Double fault, double-fault expression and detection condition$^a$ ($S = p_1 + \ldots + p_m$)

(a) Some double-fault expressions due to double faults without ordering

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Double-fault Expression</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENF $\bowtie$ TOF</td>
<td>Case 1 ($i &lt; h_1 &lt; i_2$): $p_1 + \ldots + p_{i_1} + \ldots + p_{i_2} + \ldots + p_m$</td>
<td>Any point in $\left( \bigcup_{i=1}^{h_1} TP_i(S) \right) \setminus \left( \bigcup_{i=i_1}^{2} TP_i(S) \right)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td></td>
<td>Case 2 ($i_1 \leq i_2 \leq h_1$ and $i &lt; h_1$): $p_1 + \ldots + p_{i_1} + p_{i_1+1} + \ldots + p_{i_2} + \ldots + p_m$</td>
<td>Any point in $\left( \bigcup_{i=i_1}^{h_1} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td>TNF $\bowtie$ CORF</td>
<td>Case 1 ($i &lt; i_2$): $p_1 + \ldots + p_{i_1} + \ldots + p_{i_2} + \ldots + p_m$</td>
<td>Any point in $UTP_i(S)$ such that $p_{i_2-1} + p_{i_2} + 1 = 0$ or any point in $FP(S)$</td>
</tr>
<tr>
<td></td>
<td>Case 2 (both faults occurred at the same term, $p_{i_1}$): $p_1 + \ldots + p_{i_1} + 1 + p_{i_1+1} + \ldots + p_m$</td>
<td>Any point in $FP(S)$ such that $p_{i_1} + 1 + 1 = 0$</td>
</tr>
</tbody>
</table>

(b) A double-fault expression due to double faults with ordering

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Faulty Expression</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORF $\bowtie$ TNF</td>
<td>$p_1 + \ldots + p_{i_1} + 1 + p_{i_1+1} + 1 + \ldots + p_m$</td>
<td>Any point in $FP(S)$ such that $p_{i_1} + 1 + 1 = 0$</td>
</tr>
</tbody>
</table>

$^a$Due to page limitation, other double fault classes and their corresponding faulty expressions and detection conditions are omitted here. Interested readers may refer to [5, 6] for the complete lists and detailed discussions.

The detection condition shows that any true point in $\left( \bigcup_{i=1}^{h_1} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$ or any false point of $S$ can distinguish $S$ and the double-fault expression.

4. Fault Coupling

Most fault-based testing strategies derive their test cases based on the assumption that at most one of the hypothesized faults is committed by programmers [11]. A fundamental question in fault-based testing is whether test cases that detect programs with single fault in isolation are also able to detect programs with multiple faults in combination.

Fault coupling has been studied for years [3, 12], but so far it has no universally agreed definition. In this paper, faults are said to be coupled together if they can be detected in isolation but not when combined together. Most previous studies on fault coupling focused on the combination of two single faults.

An empirical study on fault coupling via mutation analysis was done in [12]. A mutant is a program which differs from the original program by small syntactic changes. A 1-order (respectively, 2-order) mutant is a mutant that differs from the original program by 1 syntactic change (respectively, 2 syntactic changes). Three programs whose size ranges from 16 to 28 lines of code (LOC) were studied. Test sets that can kill all 1-order mutants were generated and used to kill 2-order mutants. As mentioned in [12], the experiment studies the mutation coupling effect, to be precise. It was found that test sets so generated can kill approximately 99.9% of 2-order mutants. It was then concluded that the effect of two faults being coupled together rarely occurs.

How Tai Wah [3] investigated fault coupling from a theoretical perspective. He also studied the behaviour of double faults. A program is modelled as a composition of several mathematical functions. For example, suppose that a program $P$ is considered as a composition of three mathematical functions $f$, $g$ and $h$ in the order of $f$ being computed first, followed by $g$ and finally $h$, then $P$ is equivalent to the composite function $h \circ g \circ f$. A single fault in a program is modelled as an incorrect use of one of the functions during the composition. For example, if a fault occurs in the program $P$, it is possible that any one of the three functions $f$, $g$ or $h$ is implemented wrongly as $f'$, $g'$ or $h'$, respectively, to result in a faulty program which may be equivalent to $h \circ g \circ f'$, $h \circ g' \circ f$ or $h' \circ g \circ f$.

Moreover, a double fault in a program is modelled as the incorrect use of any of two functions during the composition. Test sets that detect individual faults of a double fault are called proper test sets. Among the proper test sets, those that cannot detect the double fault are called coupled test sets. How Tai Wah [3] then calculates the coupling ratio, defined as the ratio of the number of coupled test sets to that of proper test sets. He shows analytically that the coupling ratio is approximately $1/|D|^2$ for test sets of size 1 and $1/|D|^2$ for test sets of size 2, where $|D|$ is the size of the input domain $D$. It should be noted that when a test set of size 1 detects a double fault, the only test case in the set must be able to detect each of the
two individual faults in isolation. As \(|D|\) is usually very large, the coupling ratio is very small, and he concludes that fault coupling rarely occurs.

In this study, we are more interested to know which double fault class can be detected by test cases that can detect the individual single fault classes. Hence, instead of performing empirical study via mutation analysis or calculating the coupling ratio via mathematical analysis, we analyse the relationship between single and double faults based on their detection conditions. More precisely, if \(DC_A\), \(DC_B\) and \(DC_{A\&B}\) are the detection conditions of the single fault classes \(A\) and \(B\) and the double fault class \(A \bowtie B\), respectively, we would like to identify the relationship among \(DC_A\), \(DC_B\) and \(DC_{A\&B}\). Our aim is to find out which double fault class \(A \bowtie B\) can be detected by test cases that can detect either of the two single fault classes \(A\) and \(B\).

For test cases that detect fault class \(A\), there are three mutually exclusive possibilities:

\((R1)\) All points that satisfy \(DC_A\) will also satisfy \(DC_{A\&B}\); that is, any point that can detect \(A\) will also detect \(A \bowtie B\).

\((R2)\) Some but not all points that satisfy \(DC_A\) also satisfy \(DC_{A\&B}\); that is, some but not all points that can detect \(A\) can also detect \(A \bowtie B\).

\((R3)\) None of the points that satisfy \(DC_A\) also satisfies \(DC_{A\&B}\); that is, none of the points that can detect \(A\) will also detect \(A \bowtie B\).

Similarly, for test cases that can detect \(B\), there are three other mutually exclusive possibilities:

\((R4)\) All points that satisfy \(DC_B\) will also satisfy \(DC_{A\&B}\); that is, any point that can detect \(B\) will also detect \(A \bowtie B\).

\((R5)\) Some but not all points that satisfy \(DC_B\) also satisfy \(DC_{A\&B}\); that is, some but not all points that can detect \(B\) can also detect \(A \bowtie B\).

\((R6)\) None of the points that satisfy \(DC_B\) also satisfies \(DC_{A\&B}\); that is, none of the points that can detect \(B\) will also detect \(A \bowtie B\).

Table 4 extracted from [6] shows the relationship between detection conditions of double fault classes and their individual single fault classes. As an example, let us consider \(ENF \bowtie TOF\). In Case 1 of \(ENF \bowtie TOF\), the subexpression \(p_{i_1} + \ldots + p_{h_1}\) of \(S\) is wrongly negated and the \(i_2\)-th term \(p_{i_2}\) is omitted from \(S\) where \(h_1 < i_2\), and the corresponding double-fault expression is given by (5) in Table 3. From Table 3, the detection condition of Case 1 of \(ENF \bowtie TOF\), denoted as \(DC_{ENF\bowtie TOF-1}\), is \(\{\) any point in \(\bigcup_{i=i_1}^{h_1} TP_i(S) \setminus \bigcup_{i \neq i_1 \ldots h_1, i_2} TP_i(S)\}\) or any point in \(FP(S)\). Now, the detection conditions of the individual faults \(ENF\) and \(TOF\), denoted as \(DC_{ENF}\) and \(DC_{TOF}\), are given by “any point in \(\bigcup_{i=i_1}^{h_1} TP_i(S) \setminus \bigcup_{i \neq i_1 \ldots h_1} TP_i(S)\) or any point in \(FP(S)\)” and “any point in \(UTP_{i_2}(S)\), respectively. Since \(\bigcup_{i=i_1}^{h_1} \bigcup_{i \neq i_1 \ldots h_1, i_2} TP_i(S) \subset \bigcup_{i=i_1}^{h_1} TP_i(S)\), we have \(\bigcup_{i=i_1}^{h_1} \bigcup_{i \neq i_1 \ldots h_1, i_2} TP_i(S) \subset \bigcup_{i=i_1}^{h_1} TP_i(S)\). Hence, \(DC_{ENF}\) implies \(DC_{ENF\bowtie TOF-1}\). Thus, \(R1\) holds. For the relationship between \(DC_{TOF}\) and \(DC_{ENF\bowtie TOF-1}\), since every term in \(p_{i_1}, \ldots, p_{h_1}\) evaluates to 0 on every element in \(UTP_{i_2}(S)\)

\((i_2 \notin \{i_1, \ldots, h_1\})\), \(UTP_{i_2}(S) \cap \bigcup_{i=i_1}^{h_1} TP_i(S) = \emptyset\). Hence,

\(UTP_{i_2}(S) \cap \bigcup_{i=i_1}^{h_1} TP_i(S) = \emptyset\).

Therefore, satisfying \(DC_{TOF}\) implies that \(DC_{ENF\bowtie TOF-1}\) cannot be satisfied. Thus, \(R6\) holds. Similarly, in Case 2 of \(ENF \bowtie TOF\), it is straightforward to show that only \(R2\) and \(R6\) hold.

Since our objective is to determine which double fault classes can always be detected by test cases that can detect individual single fault classes, we are particularly interested in \(R1\) or \(R4\). From Table 3, 15 out of 31 double-fault expressions have \(R1\) or \(R4\). These 15 expressions can always be detected by test cases that collectively detect the two individual single fault classes. For the remaining expressions, there is no guarantee that those test cases that detect each individual single faults will detect the double-fault expressions. It is interesting to note that \(R3\) and \(R6\) do not hold simultaneously. Hence, regarding to double fault classes studied in this paper, there is always a chance that test cases that collectively detect individual single fault classes may also detect those double fault classes.

Up to now, when analysing the relationships among detection conditions, we have confined to the question of whether a test case that can detect a single fault class can also detect a double fault class. One may actually also ask the following question:

Can a test set that detect all individual single fault classes in Section 2 also detect all double...
Unfortunately, the answer is “NO” as illustrated in the following example.

**Example 4.1** Let \( S_1 = a + b cde \). Choose the unique true point \( \vec{t}_1 = (1, 0, 0, 0, 0) \) (that is, \( a = 1, b = c = d = e = 0 \)) of term 1 of \( S_1 \), the unique true point \( \vec{t}_2 = (0, 1, 1, 1, 1) \) of term 2 of \( S_1 \), and the false point \( \vec{t}_3 = (0, 0, 1, 1, 1) \) of \( S_1 \). Then, according to Table 1, \( \vec{t}_1 \) and \( \vec{t}_2 \) together detect all single-fault expressions related to ENF, TNF, TOF and DORF, and \( \vec{t}_3 \) detects all single-fault expressions related to CORF, that is, \( a + b + cde, a + b + d e \) and \( a + b + cde, a + b + d e \) and \( a + b + cde, a + b + d e \) evaluate to 1 on \( \vec{t}_3 \). Hence, \( T = \{ \vec{t}_1, \vec{t}_2, \vec{t}_3 \} \) detects all single faults classes studied in this paper. However, this test set cannot detect the double-fault expression \( a + b + cde, a + b + d e \) where a CORF \( \Rightarrow \) TNF is committed, because \( S_1 \) and \( a + b + cde, a + b + d e \) agree on all points in \( T \).

## 5. Comparing Existing Testing Strategies

In this section, we consider the following question:

> Many existing test case selection strategies have been developed that can detect all individual single fault classes in Section 2. Can these strategies also detect all double fault classes considered in this paper?

By using detection conditions of double faults, we find that the answer to this question is "YES".

Existing test case selection strategies for detecting faults in Boolean expressions include the BOR strategy [13, 14], the BASIC meaningful impact strategy (or simply the BASIC strategy) [17], the MUMCUT strategy [19] and the modified condition/decision coverage (MC/DC) criterion [2]. Since the BOR strategy requires every variable in the expression to occur only once, it is not widely applicable to all Boolean expressions in IDNF. It has been shown in [18] that MC/DC cannot guarantee to detect single-fault expressions due to TOF, DORF and CORF. As shown in [1, 19], both the BASIC and MUMCUT strategies can detect all single fault classes described in Section 2. Hence, we consider these two strategies further.

Given a Boolean expression \( S \) in IDNF, the BASIC strategy selects a unique true point from every \( UTP_i(S) \), and a near false point from every \( NFP_{i,j}(S) \). The following three theorems extracted from [6] show that the test set selected by the BASIC strategy, and hence, any strategy that subsumes it, satisfies all detection conditions of all double-fault expressions considered in this paper. (A testing criterion \( C_1 \) is said to subsume another criterion \( C_2 \) if any test set that satisfies \( C_1 \) must also satisfy \( C_2 \).)

**Theorem 5.1** Let \( S = p_1 + \cdots + p_m \) be a Boolean expression in irredundant disjunctive normal form. Suppose that \( T \) is the set of near false points formed by selecting a near false point from \( NFP_{i,j}(S) \) for every \( i \) and \( j \). Then, \( T \) satisfies the following conditions:

1. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \).
2. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \) and \( p_{i_1,j_1} + p_{i_2,j_2} \) evaluates to 0 on \( \vec{t} \).
3. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \) and \( p_{i_1,j_1} + p_{i_2,j_2} + p_{i_3,j_3} \) evaluates to 0 on \( \vec{t} \).
4. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \) and \( p_{i_1,j_1} + p_{i_2,j_2} + p_{i_3,j_3} \) evaluates to 0 on \( \vec{t} \).
5. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \) and \( p_{i_1,j_1} + p_{i_2,j_2} + p_{i_3,j_3} \) evaluates to 0 on \( \vec{t} \).
6. There exists \( \vec{t} \in T \) such that \( \vec{t} \in FP(S) \) and \( p_{i_1,j_1} + p_{i_2,j_2} + p_{i_3,j_3} \) evaluates to 0 on \( \vec{t} \).

### Table 3: Relationship of detection conditions for single and double faults (extracted from [6])

<table>
<thead>
<tr>
<th>Fault class</th>
<th>ENF</th>
<th>TNF</th>
<th>TOF</th>
<th>DORF</th>
<th>CORF</th>
</tr>
</thead>
<tbody>
<tr>
<td>First fault class A</td>
<td>TOF</td>
<td>ENF</td>
<td>TNF</td>
<td>TOF</td>
<td>DORF</td>
</tr>
<tr>
<td>Second fault class B</td>
<td>ENF</td>
<td>TNF</td>
<td>TOF</td>
<td>DORF</td>
<td>CORF</td>
</tr>
</tbody>
</table>
Proof: Due to page limitation, we only present the proofs of conditions (1) and (3). The proofs of other conditions can be found in [6].

(1) By the definition of $T$, any $\vec{t} \in T \subset NFP(S) \subset FP(S)$.

(2) By the definition of $T$, there exists $\vec{t} \in T$ such that $\vec{t} \in NFP_{i,j_1+1}(S) \subset FP(S)$. Therefore, the $(j_1+1)$-th literal $x_{j_1+1}$ evaluates to 0 and other literals of $p_{i_1}$ evaluate to 1 on $\vec{t}$. Thus, $p_{i_1,j_1+1} = x_{j_1+1} \cdots x_{j_1}$ evaluates to 0 on $\vec{t}$. Hence, the result follows.

Theorem 5.2 Let $S = p_1 + \cdots + p_m$ be a Boolean expression in irredundant disjunctive normal form. Suppose that $T$ is the set of unique true points formed by selecting a unique true point from $UTP_i(S)$ for every $i$. Then, $T$ satisfies the following conditions:

1. There exists $\vec{t} \in T$ such that $\vec{t} \in UTP_{i_1}(S)$.
2. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
3. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
4. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
5. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
6. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
7. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.
8. There exists $\vec{t} \in T$ such that $\vec{t} \in \left( \bigcup_{i=1}^{m} TP_i(S) \right) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$.

Proof: Due to page limitation, we only present the proof of condition (1). Interested reader may refer to [6] for proofs of other conditions.

(1) By the definition of $T$, there is a $\vec{t} \in T$ such that $\vec{t} \in UTP_{i_1}(S)$. Hence, the result follows.

Theorem 5.3 Let $S = p_1 + \cdots + p_m$ be a Boolean expression in irredundant disjunctive normal form. The BASIC meaningful impact strategy, and hence any strategy that subsumes it, can detect all double-fault expressions considered in this paper.

Proof: Any test set generated by the BASIC strategy satisfies all the conditions stated in Theorems 5.1 and 5.2, and hence must satisfy all detection conditions of all double-fault expressions. For example, condition (3) of Theorem 5.1 satisfies the detection condition of CORF $\propto$ TNF in Table 3. Interested readers may refer to [6] for the detailed proof. Thus, the BASIC meaningful impact strategy can detect all corresponding double-fault expressions. Hence, the result follows.

Among the existing test case selection strategies proposed in research literature, those that subsume the BASIC strategy include the MANY-A strategy, the MAX-A strategy and the MUMCUT strategy [17, 19]. By Theorem 5.3, all these strategies can also detect all double fault classes considered in this paper. The empirical study in [17] shows that, for the group of Boolean expressions under study, the BASIC, MANY-A and MAX-A strategies require, on average, test sets of sizes 9.8%, 19.3% and 40.6%, respectively, of the entire input domain. As reported in [19], the sizes of test sets generated by the MUMCUT strategy for the same group of Boolean expressions are, on average, approximately 12.0% of the entire input domain. Hence, both the BASIC and MUMCUT strategies generate much smaller test sets than the MANY-A and MAX-A strategies do, and that they can detect all double faults related to terms in Boolean expressions. Although the MUMCUT strategy requires slightly larger test sets, it can detect
more classes of single faults (such as literal insertion fault and literal reference fault) than the BASIC strategy does [1, 19].

6. Conclusion and Future Work

This paper presents the detection conditions of double faults related to terms within a Boolean expression. Among the five single fault classes discussed, there are 15 double fault classes without ordering, resulting in 27 non-equivalent faulty expressions. Moreover, there are 25 double fault classes with ordering, resulting in 53 faulty expressions, of which 49 of them are equivalent to the 27 distinct double-fault expressions due to double faults without ordering, and the 4 remaining double-fault expressions are not equivalent to any of the previous group of 27 faulty expressions. Altogether, there are 31 different double-fault expressions among all double-fault classes considered in this paper.

Based on the detection conditions of double-fault expressions, we have also analysed the fault coupling effect. We find that 15 out of the 31 double-fault expressions can always be detected by test cases that detect single fault classes. Moreover, for the remaining 16 double-fault expressions, some but not all test cases that detect each individual single fault class will also detect the double-fault expressions.

Furthermore, based on the detection conditions, we prove that any test case selection strategy that subsumes the BASIC meaningful impact strategy can detect all double fault classes considered in this paper. This is very interesting because none of these strategies were originally developed for detecting double faults. Among existing test case selection strategies based on Boolean expressions, both the BASIC strategy and the MUMCUT strategy generate much smaller test sets than other strategies that subsume the BASIC strategy.

Some other classes of single faults related to literals in Boolean expressions have not been studied in this paper. We are extending our work to explore detection conditions of double faults related to literals and to further analyse whether existing test case selection strategies are able to detect all these double fault classes. We intend to complete the analysis of detection conditions so as to understand more precisely the behaviour of multiple faults for Boolean expressions.

References