Detecting Double Faults on Term and Literal 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

Fault-based testing aims at selecting test cases to guarantee the detection of certain prescribed faults in programs. The detection conditions of single faults have been studied and used in areas like developing test case selection strategies, establishing relationships between faults and investigating the fault coupling effect. It is common, however, for programmers to commit more than one fault. Our previous studies on the detection conditions of faults in Boolean expressions show that (1) some test case selection strategies developed for the detection of single faults can also detect all double faults related to terms, but (2) these strategies cannot guarantee to detect all double faults related to literals. This paper supplements our previous studies and completes our series of analysis of the detection condition of all double fault classes in Boolean expressions. Here we consider the fault detection conditions of combinations of two single faults, in which one is related to term and the other is related to literal. We find that all such faulty expressions, except two, can be detected by some test case selection strategies for single fault detection. Moreover, the two exception faulty expressions can be detected by existing strategies when used together with a supplementary strategy which we earlier developed to detect double literal faults.

1. Introduction

Software testing is an important process in software quality assurance. One of the many goals of software testing is to verify that the software behaves correctly according to its specifications. Fault-based testing aims at selecting test cases to guarantee the detection of certain prescribed faults that commonly occur during program development.

There are various methods to model the specifications of software systems using Boolean expressions as part of the formalism in industrial systems including avionics, nuclear power plant control and cruise control [1,11,14]. Such formalism can help to automatically generate test cases based on specifications [2,11,14].

Furthermore, research has been done on using fault-based techniques to generate test cases based on Boolean expressions [2,4,12–14]. Although previous studies focus mainly on specifications, the results can also be applied to program-based predicate testing which usually involves logic expressions [12].

Recently, the detection conditions of various hypothesized faults have been studied [2,4,7–9,13]. The detection condition of a particular fault in a program is a condition that makes the actual output of the program different from its expected output. Previous studies on detection conditions are mainly used in three areas. First, the detection conditions have been used to develop test case selection strategies, such as BASIC, MAX-B and MUMCUT [14,15]. Second, the detection conditions have been used to develop the fault class hierarchy. A fault class hierarchy establishes relationships among different types of faults. Kuhn [4] and Lau and Yu [9] used fault class hierarchies to explain fault-detecting abilities of various test case selection strategies. Third, the detection conditions have been used to investigate fault coupling. When two faults combine in such a way that they cannot be detected by the test cases which can detect them separately, the two faults are said to be coupled together.

Both experimental and theoretical studies on fault coupling investigate the chance of a test set in detecting double faults given that the test set can detect two individual faults in isolation [3,10]. However, in these studies, the relationships between single and double faults have not been studied and, hence, have not been used for double fault detection. In order to achieve detection of particular double fault classes, we need to study their detection conditions. Furthermore, by analysing detection conditions of double fault classes, we can obtain effective test case generation strategies for single and double fault detection. Since there are altogether 81 different possible ways to form a double fault based on 9 different single...
fault classes related to terms and literals reported in [9], we apply a divide-and-conquer approach to further classify all double fault classes into following three categories:

1. double faults related to terms only;
2. double faults related to literals only;
3. double faults related to a term and a literal.

For the first category, there are altogether 31 different double fault expressions\(^1\), that is, expressions that contain double faults [7]. We studied their detection conditions and showed that all these faulty expressions can be detected by any test case selection strategy that subsumes the BASIC meaningful impact strategy.

A similar study on the second category showed that there are altogether 19 different double fault expressions, and that existing test case selection strategies are insufficient in detecting these faulty expressions. Hence, new test case selection strategies were developed [8].

In this paper, we study the detection conditions of double faults in the third category, so as to find suitable test case selection strategies for their detection.

The rest of the paper is organized as follows: Section 2 introduces the notation and fault classes studied in this paper. Section 3 discusses the double fault classes and their detection conditions. Section 4 analyses the fault detection abilities of existing test case selection strategies in detecting double faults. Section 5 presents our analysis that the MUMCUT strategy, when used together with a supplementary strategy which we previously developed, guarantees to detect all the double faults in this study. Section 6 concludes the paper.

### 2. Preliminary

#### 2.1. Notation

In this paper, 1 and 0 are used to represent ‘TRUE’ and ‘FALSE’ respectively. The Boolean operators AND, OR and NOT, are denoted by ‘·’, ‘+’ and ‘−’, respectively. Usually, ‘−’ is omitted whenever it is clear in the context. The set of all truth values \(\{0, 1\}\) is denoted by \(\mathbb{B}\).

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 and \(p_i\) is the \(i\)-th term of \(S\). Let \(p_i = x_1^{k_1} \cdots x_j^{k_j}\), where \(x_j\) is the \(j\)-th literal in \(p_i\) and \(k_j\) is the number of literals in \(p_i\). A term \(p_i\) is said to be a minterm if it consists of every literal or its negation in the formula. A literal \(x\) is a missing literal of a term \(p_i\) if both \(x\) and \(\overline{x}\) do not appear in the term. If \(S\) has \(n\) literals, the input domain is the \(n\)-dimensional Boolean space \(\mathbb{B}^n\). A test case for \(S\) is a point in \(\mathbb{B}^n\).

---

\(^1\)See Section 3 for the definition of a double fault expression.

A true point of \(S\) is a point in \(\mathbb{B}^n\) such that \(S\) evaluates to 1. We use \(TP(S)\) to denote the set of all true points of \(S\). A true point of the \(i\)-th term, \(p_i\), of \(S\) is a point such that \(p_i\) evaluates to 1. We use \(TP_i(S)\) to denote the set of all true points of \(p_i\) in \(S\). A unique true point of \(p_i\) in \(S\) is a true point of \(p_i\) such that terms other than \(p_i\) evaluate to 0. The set of all unique true points of \(p_i\) is denoted by \(UTP_i(S)\). True points that are not unique true points are overlapping true points. The set of all overlapping true points of \(S\) is denoted by \(OTP(S)\). For example, let \(S = ab + cd + de\). A true point of \(S = ab\) in \(S\) is 11000 (that is, \(a = b = 1\), \(c = d = e = 0\)). The set \(TP(S)\) of all true points of \(p_i\) is \{11000, 11001, 11010, 11101, 11110, 11111\}. The point 11000 is a unique true point of \(p_i\) in \(S\). The point 11110 is an overlapping true point of \(S\).

A false point of \(S\) is a point in \(\mathbb{B}^n\) such that \(S\) evaluates to 0. We use \(FP(S)\) to denote the set of all false points of \(S\). A near false point for the \(j\)-th literal, \(x_j\), of the \(i\)-th term, \(p_i\), in \(S\) is a false point of \(S\) such that (1) \(x_j\) evaluates to 0, and (2) all literals in \(p_i\) other than \(x_j\) evaluate to 1. The set of all near false points for \(x_j\) of \(p_i\) in \(S\) is denoted by \(NFP_{i,j}(S)\). False points that are not near false points are remaining false points. The set of all remaining false points of \(S\) is denoted by \(RFP(S)\). For example, let \(S = ab + cd + de\). A near false point for the first literal \(a\) of the first term \(ab\) is 01000. The set \(NFP_{1,1}(S)\) of all near false points for the first literal \(a\) of the first term \(ab\) is \{01000, 01001, 01010, 01100, 01101\}. The point 00000 is a remaining false point of \(S\).

#### 2.2. Fault Class

Lau and Yu [9] have considered various common types of single faults that may be committed in Boolean expressions in the research literature [2, 4, 13, 14] and proposed nine different fault classes. Among these fault classes, five of them are related to terms in Boolean expressions, while the remaining four are literal faults. The detection conditions of double faults related to terms and double faults related to literals have been studied [7, 8].

In this paper, we study the detection conditions of double faults in which one fault is related to term and the other is related to literal. The nine single fault classes are described as follows.

1. **Expression Negation Fault (ENF):** The entire Boolean expression or its subexpression is negated.
2. **Term Negation Fault (TNF):** A term is negated.
3. **Term Omission Fault (TOF):** A term is omitted.
4. **Disjunctive Operator Reference Fault (DORF):** The Boolean operator ‘+’ between two consecutive terms is implemented as ‘·’.
5. **Conjunctive Operator Reference Fault (CORF):** The Boolean operator ‘·’ between two consecutive literals in a term is implemented as ‘+’.
6. **Literal Negation Fault (LNF):** A literal is negated.
7. Literal Omission Fault (LOF): A literal is omitted.
8. Literal Insertion Fault (LIF): A missing literal of a term is inserted into the term.
9. Literal Reference Fault (LRF): A literal in a term is replaced by a missing literal of the term.

Table 1 lists these nine fault classes, their corresponding faulty expressions and detection conditions. As an example, let us consider the row in Table 1 which represents the situation that a LIF is committed during the implementation of the specification $S = ab + cd + de$. Suppose the literal $c$ is wrongly inserted into the first term $ab$ in $S$, resulting in the faulty expression $I = abc + cd + de$. Then, the corresponding detection condition is “any unique true point of the first term $ab$ in $S$ such that $c$ evaluates to 0”.

Table 1

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Single Fault Expression</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENF</td>
<td>$p_1 + \cdots + p_{i-1} + p_i + \cdots + p_{j-1} + p_{j+1} + \cdots + p_m$</td>
<td>any point in $\bigcup_{i=1}^{h_1} TP_i(S)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td>TNF</td>
<td>$p_1 + \cdots + p_m$</td>
<td>any point in $UTP_i(S)$ or any point in $FP(S)$</td>
</tr>
<tr>
<td>TOF</td>
<td>$p_1 + \cdots + p_i + p_{i+1} + \cdots + p_m$</td>
<td>any point in $UTP_i(S)$</td>
</tr>
<tr>
<td>DORF</td>
<td>$p_1 + \cdots + p_i + p_{i+1} + \cdots + p_m$</td>
<td>any point in $UTP_i(S)$</td>
</tr>
<tr>
<td>CORF</td>
<td>$p_1 + \cdots + p_i + p_{i+1} + p_{i+1} + p_{j+1} + p_{j+1} + \cdots + p_m$</td>
<td>any point in $FP(S)$ such that $p_{i+1} + p_{j+1} + p_{j+1} = 1$</td>
</tr>
<tr>
<td>LNF</td>
<td>$p_1 + \cdots + x^1_i + \cdots + x^1_i + \cdots + x^1_i$</td>
<td>any point in $UTP_i(S)$ or any point in $NFP_{i,j,i}(S)$</td>
</tr>
<tr>
<td>LOF</td>
<td>$p_1 + \cdots + x^1_i + \cdots + x^1_i + \cdots + x^1_i$</td>
<td>any point in $NFP_{i,j,i}(S)$</td>
</tr>
<tr>
<td>LIF</td>
<td>$p_1 + \cdots + p_i x_i + \cdots + p_m$</td>
<td>any point in $UTP_i(S)$ such that $x_i = 0$</td>
</tr>
<tr>
<td>LRF</td>
<td>$p_1 + \cdots + x^1_i + \cdots + x^1_i + \cdots + x^1_i$</td>
<td>any point in $UTP_i(S)$ such that $x_i = 1$</td>
</tr>
</tbody>
</table>

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 $S$. Here, the negation of the entire Boolean expression is included as a special case of the negation of the subexpression $p_1 + \cdots + p_m$ when $i_1 = 1$ and $h_1 = m$.

The detection condition of TNF can also be expressed as “any point such that $p_1 + \cdots + p_{i-1} + p_{i+1} + \cdots + p_m = 0$”. However, for ease of reference and discussions in later sections, we choose to present it in the form shown here to be consistent with those of other faults.

7.4.1 Fault Class: Double Faults on Term and Literal

Multiple occurrences of single fault classes in an expression may result in a faulty expression which differs from the original expression by several syntactic changes. An expression which differs from the original expression by more than one syntactic change is considered to have multiple faults. For example, $\overline{ab} + cd + ef$ is a result of making two syntactic changes to $ab + cd + ef$, namely, the negation of the literal $b$ and the negation of the term $cd$.

Lau et al. [7, 8] point out that, when two single faults occur in a Boolean expression, the resulting faulty expression may be equivalent to either the original expression or an expression with a single fault (that is, a faulty expression which differs from the original expression by one syntactic change). Since there are test case generation strategies that can guarantee to detect all the nine types of single faults in the previous section [2], this study will only concentrate on the detection of double-fault expressions. A double-fault expression is defined as an expression that (1) differs from the original expression by two syntactic changes and (2) is equivalent to neither the original expression nor any faulty expression with a single fault.

In [7, 8], double-fault expressions related to both term faults and both literal faults have been studied. In this paper, we study double fault on term and literal, which is defined as the double faults with one fault being a term fault and the other being a literal fault.

When a double fault is committed in an expression, it is possible that by swapping the order of occurrences of the two individual faults may result in non-equivalent double-fault expressions. Such a double fault is referred to as double fault with ordering. For example, let $S = ab + cd + ef$. If the first term $ab$ of $S$ is negated before the literal $e$ is inserted into the resulting first term, the resulting faulty expression is equivalent to $\overline{I} = \overline{ab}e + cd + ef$. However, if the literal $e$ is inserted in the first term and the resulting first term is then negated, the resulting faulty expression is then equivalent to $\overline{ab}e + cd + ef$, which is not equivalent to $I$. On the contrary, it is also possible for the two resulting faulty expressions to be equivalent to each other, but not to the original expression. For example, consider again the expression $S = ab + cd + ef$. If the literal $a$ is negated before the second term $cd$ is negated, the resulting faulty expression is equivalent to $\overline{ab} + \overline{cd} + ef$. If these two faults are committed in the reverse order (that is, the term $cd$ is negated before the literal $a$ is negated), the resulting faulty expression is still equivalent to $\overline{ab} + \overline{cd} + ef$. In other
Double Fault Expression
Detection Condition
Case 1 \((i_1 < i_2): p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_{i_2-2} x_{i_2} + \cdots + p_m\) (**22**)
- (C1) any point in \(UTP_i(S)\) such that \(x_{i_2} = 0\) or
- (C2) any point in \(UTP_i(S)\) such that \(p_{i_2,j_2} = 0\) or
- (C3) any point in \(UTP_i(S)\) such that \(x_{i_2} = 0\) or
- (C4) any point in \(NFP_{i_2,j_2}(S)\) such that \(x_{i_2} = 1\) or
- (C5) any point in \((TP_i(S) \cap TP_j(S)\) \ \bigcup_{j=1}^{m} TP_j(S)\) such that \(x_{i_2}=0\)

Case 2 \((i_1 + 1 < i_2): p_1 + \cdots + p_i j_1 x_{i_1} p_{i_1+1} + \cdots + p_m\) (**29**)
- (C1) any point in \(UTP_i(S)\) such that \(x_{i_1} = 0\) or
- (C2) any point in \(UTP_i(S)\) such that \(p_{i_1,j_1} = 0\) or
- (C3) any point in \(UTP_i(S)\) such that \(x_{i_1} = 0\) or
- (C4) any point in \(NFP_{i_1,j_1}(S)\) such that \(x_{i_1} = 1\) or
- (C5) any point in \((TP_i(S) \cap TP_j(S)\) \ \bigcup_{j=1}^{m} TP_j(S)\) such that \(x_{i_1}=0\)

(b) The remaining double fault expressions due to double faults with ordering

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Double Fault Expression</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENF is committed before LIF</td>
<td>(p_1 + \cdots + p_{n_1} + \cdots + p_m) (x_{n_1} + \cdots + p_m) (<strong>43</strong>)</td>
<td>(C1) any point in (\bigcup_{i=1}^{n} TP_i(S)) \ \bigcup_{m j=1}^{m} TP_j(S)) or</td>
</tr>
<tr>
<td>TNP is committed before LIF</td>
<td>(p_1 + \cdots + p_{n_1} x_{n_1} + \cdots + p_m) (<strong>52</strong>)</td>
<td>(C1) any point in (UTP_i(S)) or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(C2) any point in (FP(S)) such that (x_{n_1}=1)</td>
</tr>
</tbody>
</table>

Due to page limitation, other 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.

For any two single fault classes \(A\) and \(B\), the double fault class formed by \(A\) and \(B\) without ordering is denoted by \(A \bowtie B\). Table 2 lists some double fault classes on term and literal, their corresponding faulty expressions and detection conditions. As shown in the last row of Table 2(a), there are two subcases of \(DORF \bowtie LRF\). Due to space limitation, we only discuss the second subcase of \(DORF \bowtie LRF\). Interested readers may refer to [5] for details of all double faults on term and literal.

For example, consider the expression \(S = abe + cd + def\). Suppose the literal \(e\) in the first term of \(S\) is replaced by the literal \(f\) and the ‘+’ operator between the first two terms of \(S\) is replaced by the ‘-’ operator, the faulty expression is then equivalent to \(I = abcd f + def\). The corresponding detection condition is \((C1)\) any point in
$UTP_1(S)$, or (C2) any point in $UTP_2(S)$ such that $f = 0$, or (C3) any point in $UTP_2(S)$ such that $ab = 0$, or (C4) any point in $(TP_1(S) \cap TP_2(S)) \setminus TP_3(S)$ such that $f = 0$. Any test case satisfying this condition can detect the corresponding double fault.

4. Fault Detecting Capabilities of Existing Testing Strategies

Many fault-based test case selection strategies, such as the BASIC, MAX-A and MAX-B strategies [14] and the MUMCUT strategy [15, 16] have been proposed. These strategies are fault-based because they aim at the detection of the single faults described in Section 2.2. Among these strategies, the MAX-B strategy is the most powerful because it subsumes the MAX-A strategy, which in turn subsumes the MUMCUT strategy, which in turn subsumes the BASIC strategy. A testing strategy $A$ is said to subsume another testing strategy $B$ if every test set that satisfies $A$ also satisfies $B$.

During the study of double term faults [7], Lau et al. found that any test case selection strategy that subsumes the BASIC strategy can detect all double term faults that may occur in Boolean expressions. However, in another similar study on double literal faults [8], the MAX-B strategy, and hence every strategy mentioned above, is insufficient in detecting all double literal faults. As a result, further test case selection strategies are required and have been proposed to supplement these strategies in detecting the double literal faults.

Example 4.1 illustrates that the MAX-B strategy cannot be used to detect all double faults on term and literal. As a result, none of the abovementioned strategies can detect all double faults on term and literal.

Example 4.1. Let $S = ab + ac + ad + bd + cd + \overline{bc} + \overline{b}d + \overline{c}d + \overline{bc}d + \overline{bc}d$. Table 3 lists the sets $UTP_i(S)$ of all unique true points, the sets $NFP_{i,j}(S)$ of all near false points, the set $OTP(S)$ of all overlapping true points and the set $RFP(S)$ of all remaining false points of $S$. Suppose the first term $ab$ of $S$ is omitted and the literal $d$ in the third term $ad$ of $S$ is replaced by the literal $c$. The resulting double-fault expression is equivalent to $I = ac + ae + bd + cd + \overline{bd} + \overline{cd} + \overline{b}d$. Note that, $S$ and $I$ are not equivalent because $S$ and $I$ evaluate to 0 and 1 on 11010, respectively.

Now, let $T$ be the set that includes (1) all unique true points in $UTP_i(S)$ for every $i$, (2) all near false points in $NFP_{i,j}(S)$ for every $i$ and $j$, and (3) 4 overlapping true points, $11100, 11101, 11000$, and $11101$. It should be noted that $T$ satisfies the MAX-B strategy because it contains all unique true points of $S$, all near false points of $S$, 4 out of the 12 overlapping true points of $S$, and there are no remaining false points of $S$. Since $S$ and $I$ agree on all points in $T$, the MAX-B strategy cannot guarantee to distinguish $S$ and $I$.

5. Strategies for Double Fault Detection

Example 4.1 shows that the MAX-B strategy cannot guarantee to detect all double faults on term and literal. We shall show in this section that the SMOTP strategy, developed in [8], can be used to supplement the MUMCUT strategy to detect all double faults studied in this paper.

First, we present these strategies. A test set is said to satisfy the MUMCUT strategy if it satisfies all of the following three strategies.

- The MUTC strategy: Select points from $UTP_i(S)$ for every $i$ such that all possible truth values of every missing literal of $p_i$ are covered.
- The NFTP strategy: Select points from $NFP_{i,j}(S)$ for every $i$ such that all possible truth values of every missing literal of $p_i$ are covered.
- The CUTFNP strategy: Select one point from $UTP_i(S)$ and one point from $NFP_{i,j}(S)$ for every $i$ and $j$ pair such that the two points only differ in their corresponding truth values of the $j$-th literal in the $i$-th term of $S$.

The SMOTP strategy is defined as follows.

- The SMOTP strategy: Select points from $(TP_1(S) \cap TP_2(S)) \setminus \bigcup_{i=1}^{m} TP_i(S)$ for every two different terms $p_{i_1}$ and $p_{i_2}$ such that, for every missing literal $x_{i_1}$ of $p_{i_1}$ and every missing literal $x_{i_2}$ of $p_{i_2}$, all possible truth value combinations of $x_{i_1}$ and $x_{i_2}$ (that is, $00, 01, 10, \text{and} 11$) are covered.

The SMOTP strategy was originally developed in [8] to supplement the MUMCUT strategy to guarantee the detection of the double literal fault $\text{LIF} = \text{LIF}$. It aims to select test cases that satisfy the detection condition

$$\{(TP_1(S) \cap TP_2(S)) \setminus \bigcup_{i=1}^{m} TP_i(S)\}$$

such that $x_{i_1} + x_{i_2} = 0$, where $x_{i_1}$ and $x_{i_2}$ are missing literals for terms $p_{i_1}$ and $p_{i_2}$ of $S$, respectively. Here we give an example to illustrate how to apply the SMOTP strategy.

Example 5.1 Using the same notation as in Example 4.1, we apply the SMOTP strategy on $S$ to select the test set

$$\{11101, 11100, 11011, 11000, 10110, 10111, 10100, 10101, 01100, 01101\}$$. It contains the point 11010 which can distinguish $S$ and $I$.

We now show how the SMOTP strategy selects the point 11010. For the first term $p_1 = ab$ and the third term $p_3 = ab + ac + ad + bd + cd + cd + \overline{bc} + \overline{b}d$. Now, let $M = (TP_1(S) \cap TP_2(S)) \setminus \bigcup_{i=1}^{m} TP_i(S)$ be any point in $M$. Then, $p_{i_1} = p_1$, $p_{i_2} = p_3$, $x_{i_1} = x_{i_2} = 0$, and $x_{i_1} + x_{i_2} = 0$. Thus, $x_{i_1} + x_{i_2} = 0$, where $x_{i_1}$ and $x_{i_2}$ are missing literals for terms $p_{i_1}$ and $p_{i_2}$ of $S$, respectively. Here we give an example to illustrate how to apply the SMOTP strategy.

Example 5.1 Using the same notation as in Example 4.1, we apply the SMOTP strategy on $S$ to select the test set

$$\{11101, 11100, 11011, 11000, 10110, 10111, 10100, 10101, 01100, 01101\}$$.
Table 3: All test cases of $S$ where $S = ab + ac + ad + b\overline{d} + cd + b\overline{d} + \overline{b}\overline{c}e$

(a) $UTP_i(S)$ and $NFP_{i,j}(S)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$11001$</th>
<th>$10110$</th>
<th>$10011$</th>
<th>$01000$</th>
<th>$00100$</th>
<th>$00010$</th>
<th>$10001$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTP(S)</td>
<td>11001</td>
<td>10110</td>
<td>10011</td>
<td>01000</td>
<td>00100</td>
<td>00010</td>
<td>10001</td>
<td>00001</td>
</tr>
<tr>
<td>NFP_{i,1}(S)</td>
<td>01111, 01110, 01101, 01011, 01001</td>
<td>01111, 01110</td>
<td>01111, 01110</td>
<td>01101, 01010, 01001, 00110</td>
<td>01000, 00000</td>
<td>01000, 00000</td>
<td>01010</td>
<td>01010</td>
</tr>
<tr>
<td>NFP_{i,2}(S)</td>
<td>10000</td>
<td>10000</td>
<td>10000</td>
<td>01110, 01101</td>
<td>01110, 00110</td>
<td>01110</td>
<td>00110</td>
<td>00110</td>
</tr>
<tr>
<td>NFP_{i,j}(S)</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>01101, 01010</td>
<td>01101, 01010</td>
<td>01000, 00000</td>
<td>00110</td>
<td>00001</td>
</tr>
</tbody>
</table>

(b) $OTP(S)$ and $RFP(S)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$11100, 11101, 11000, 11110, 10010, 11010, 10100, 11011, 11111, 01100</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTP(S)</td>
<td>11100, 11101</td>
<td>11011</td>
</tr>
<tr>
<td>RFP(S)</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 4: All test cases of $S$ where $S = abd + abc + \overline{b}d$

(a) $UTP_i(S)$ and $NFP_{i,j}(S)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$11101, 11010, 10110, 01001, 00101, 00010, 01010, 00110</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTP(S)</td>
<td>11101</td>
<td>11010</td>
</tr>
<tr>
<td>NFP_{i,1}(S)</td>
<td>01111, 01110</td>
<td>01011</td>
</tr>
<tr>
<td>NFP_{i,2}(S)</td>
<td>10000</td>
<td>10000</td>
</tr>
<tr>
<td>NFP_{i,j}(S)</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

(b) $OTP(S)$ and $RFP(S)$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$j$</th>
<th>$11100, 11101, 11000, 11110, 10010, 11010, 10100, 11011, 11111, 01100</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTP(S)</td>
<td>11100</td>
<td>11101</td>
</tr>
<tr>
<td>RFP(S)</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

ad of $S$, the set $(TP_1(S) \cap TP_3(S)) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right)$ of all points such that $p_1 = p_3 = 1$ and all other terms $p_i = 0$ ($i \neq 1, 3$) is $\{11101, 11011\}$. Hence, by selecting these two points, every possible truth value combination of every pair of missing literals of the first and third terms (one missing literal of the first term and the other missing literal of the third term) are covered.

Next, we shall show that the BASIC strategy cannot detect all double faults on term and literal even when supplemented with the SMOTP strategy. The BASIC strategy requires the selection of one test case from $UTP_i(S)$ and one from $NFP_{i,j}(S)$ for every $i$ and $j$ [14].

By means of a thorough analysis of the detection conditions of all 38 double-fault expressions studied in this paper, we found that the BASIC strategy can detect 36 of them [6]. Due to space limitation, we illustrate, using an example, how all double-fault expressions of type (30) (henceforth simply called Expression (30)) in Table 2 can be detected by the BASIC strategy.

Let us consider $S = ab + cd$. By selecting a unique true point 1100 for the first term of $S$ and a unique true point 0011 for the second term of $S$, condition (C1) for Expression (30) is satisfied. Hence, any strategy that subsumes BASIC can detect Expression (30).

For the 2 remaining double fault expressions, Expressions (22) and (29) in Table 2, we find that the BASIC strategy cannot guarantee to select test cases that satisfy their corresponding detection conditions, especially for (C5) of Expression (22), and (C7) and (C8) of Expression (29). However, since these conditions are similar to one of the detection conditions of UTP $\otimes$ LIF,$$
(TP_1(S) \cap TP_2(S)) \setminus \left( \bigcup_{i=1}^{m} TP_i(S) \right) \text{ such that } x_{i_1} + x_{i_2} = 0 ^*,$$
it seems possible that the SMOTP strategy may select test cases that satisfy these detection conditions. Therefore, we seek to verify whether the BASIC strategy and the SMOTP strategy together can guarantee to detect Expression (22) and (29). The answer is negative, however, as illustrated in Example 5.2.

Example 5.2 Let $S = abd + abc + \overline{b}d$. Table 4 lists all the test points of $S$. Suppose the first term $abd$ is omitted and the literal $\overline{b}$ in the second term $abc$ is replaced by $d$. The resulting double-fault expression is equivalent to $I = acd + \overline{b}d$. This faulty expression is in the form of Expression (22) in Table 2. Note that, $S$ and $I$ are not equivalent because $S$ and $I$ evaluate to 1 and 0 on 1101, respectively. The test cases underlined in Table 4 are selected by the BASIC and SMOTP strategies. $S$ and $I$ evaluate to the same value on each of those test cases.

On the other hand, by analysing the detection conditions (C1)–(C5) of Expression (22) and (C1)–(C8) of Expression (29), we find that the MUMCUT and SMOTP strategies together guarantee to detect these expressions. Furthermore, since the MUMCUT strategy subsumes the BASIC strategy, the MUMCUT and SMOTP strategies together also guarantee to detect all 38 double-fault expressions considered in this paper.

**Theorem 5.1** Let $S = p_1 + \cdots + p_m$ be a Boolean specification in IDNF. Suppose that the $i_1$-th term, $p_{i_1}$, in $S$ is omitted and the $j_2$-th literal, $x_{j_2}$, of the $i_2$-th term, $p_{i_2}$, in $S$ is replaced by $x_{i_2}$, where $1 \leq i_1 < i_2 \leq m$, $1 \leq j_2 \leq k_{i_2}$, and $x_{i_2}$ is a missing literal of $p_{i_2}$. Then the resulting implementation denoted by $I$ will be equivalent to that given by Expression (22) in Table 2. The SMOTP strategy can supplement the MUMCUT strategy to guarantee the detection of Expression (22) provided that $S \neq I$. 

$^*$Refer to [6] for a thorough discussion of all the expressions.
Theorem 5.2 Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF. Suppose that two terms \( p_i \) and \( p_{i+1} \) in \( S \) are implemented as \( p_i, p_{i+1} \) and the \( j_2 \)-th literal, \( x_{j_2}^2 \), of the \( i_2 \)-th term, \( p_{i_2} \), in \( S \) is replaced by \( x_{i_2}^1 \), where \( 1 < i_1 + 1 < i_2 \leq m, 1 \leq j_2 \leq k_{j_2} \), and \( x_{i_2}^1 \) is a missing literal of \( p_{i_2}^1 \). Then the resulting implementation denoted by \( I \) will be equivalent to that given by Expression (29) in Table 2. The SMOTP strategy can supplement the MUMCUT strategy to guarantee the detection of Expression (29) provided that \( S \neq I \).

The detailed analysis and proofs of Theorems 5.1 and 5.2 are given in Appendix A.

6. Conclusion

Double fault classes are modelled as the combination of two single fault classes. In previous papers, we have considered double fault classes related to terms only [7] and those related to literals only [8]. This paper supplements our previous studies and completes our series of analysis [5–8] of the detection condition of all double fault classes in Boolean expressions.

In this paper, we have studied the detection conditions of double faults on term and literal within the context of Boolean expressions in IDNF. There are 20 classes of double faults without ordering, which result in 36 different double-fault expressions, and 40 classes of double faults with ordering, which result in 74 double-fault expressions. Since 72 out of 74 double-fault expressions are equivalent to the 36 distinct double-fault expressions due to double faults without ordering, and the 2 remaining double-fault expressions are not equivalent to any of the previous group of 36 faulty expressions, there are altogether 38 different double-fault expressions among all double-fault classes considered in this paper.

We found that all the test case selection strategies that were proposed for single fault detection (such as the MAX-B strategy) are insufficient to detect all double faults studied in this paper. However, after a thorough analysis, we found that 36 out of 38 double-fault expressions can be detected by any test case selection strategy which subsumes the BASIC strategy. For the remaining 2 faulty expressions, the SMOTP strategy proposed in [8], which aims at detecting double literal fault LIF\( \Leftrightarrow \)LIF, can supplement the MUMCUT strategy to guarantee the detection of these expressions. Since the MUMCUT strategy subsumes the BASIC strategy, the MUMCUT strategy together with SMOTP strategy can detect all double faults studied this paper.

As for further work, empirical studies are currently underway to evaluate the cost-effectiveness of the MUMCUT strategy and the SMOTP strategy for detecting all double faults on term and literal. We shall also investigate the chance of actually requiring the use of the SMOTP strategy to detect these double faults.

References

A. Proofs of Theorems in Section 5

We claimed in Section 5 that the SMOTOP strategy can supplement the MUMCUT strategy for the detection of Expressions (22) and (29). For ease of reading, we did not include the detailed proofs in the main text. In this appendix, we present the detailed proofs. We need the following lemmas before we discuss the main proofs.

Lemma A.1 ([2, Theorem 1]) Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF. Then,

1. \( UTP_{p_i}(S) \neq \emptyset \) for all \( i = 1, \ldots, m \)
2. \( NFP_{i,j}(S) \neq \emptyset \) for all \( i = 1, \ldots, m \) and \( j = 1, \ldots, k_i \), where \( k_i \) is the number of literals in the term \( p_i \).

Lemma A.2 Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF and \( p_i \) be a minterm of \( S \). Then, for any term \( p_i \), if \( i_2 = 1, \ldots, m \) and \( i_2 \neq i_1 \), there are at least 2 literals in \( p_i \), whose negations can be found in \( p_i \).

Proof: Since \( S \) is an IDNF, by Lemma A.1, \( UTP_{p_i}(S) \neq \emptyset \) and \( NFP_{i,j}(S) \neq \emptyset \) for \( j = 1, \ldots, k_i \) where \( k_i \) denotes the number of literals in \( p_i \). Since \( p_i \) is a minterm, it contains all variables in \( S \). Suppose, on the contrary, that there are at most 1 literal in \( p_i \) whose negation can be found in \( p_i \).

We have the following two cases:

1. No literal in \( p_i \) whose negation can be found in \( p_i \). Thus, all literals in \( p_i \) can be found in \( p_i \). Then, for any point \( \vec{r} \in \mathbb{B}^n \) such that \( p_i(\vec{r}) = 1 \), \( p_i(\vec{r}) = 1 \). As a result, \( UTP_{p_i}(S) = \emptyset \). This contradicts to the assumption that \( S \) is IDNF.

2. There is exactly one literal in \( p_i \) whose negation can be found in \( p_i \). Without loss of generality, we may assume that this literal is the negation of the first literal \( x^1_j \) in \( p_i \). Otherwise, we can always rearrange the literals so that it can become the first literal of \( p_i \). Since all other literals except \( x^1_j \) in \( p_i \) can be found in \( p_i \), for any \( \vec{r} \in \mathbb{B}^n \) such that \( p_i(\vec{r}) = 1 \), \( p_i(\vec{r}) = 1 \). As a result, \( NFP_{i,j}(S) = \emptyset \). This contradicts to the assumption that \( S \) is IDNF.

Hence, the result follows.

Lemma A.3 Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF and \( p_i \) be a minterm of \( S \). Then,

1. \( UTP_{p_i}(S) = \{ \vec{u} \} \)
2. \( NFP_{i,j}(S) = \{ \vec{n}_j \} \) for all \( j = 1, \ldots, k_i \), where \( k_i \) denotes the number of literals in \( p_i \)
3. \( \vec{u} \) and \( \vec{n}_j \) only differ on the literal \( x^1_j \) for all \( j = 1, \ldots, k_i \), where \( k_i \) denotes the number of literals in \( p_i \).

Proof: Since \( p_i \) is a minterm, it contains all variables in \( S \).

1. Hence, \( TP_{p_i}(S) \) is a singleton. Since \( S \) is an IDNF, by Lemma A.1, \( UTP_{p_i}(S) \neq \emptyset \). Thus, \( UTP_{p_i}(S) = TP_{p_i}(S) \) because \( UTP_{p_i}(S) \subseteq TP_{p_i}(S) \).

2. For each \( j = 1, \ldots, k_i \), let \( X_j = \{ \vec{r} \in \mathbb{B}^n : p_{i,j}(\vec{r}) = 1 \} \) where \( p_{i,j} \) denotes the term obtained by negating the \( j \)-th literal of \( p_i \). By definition, elements in \( NFP_{i,j}(S) \) are those in \( X_j \) such that \( S = \emptyset \). Hence, \( NFP_{i,j}(S) \subseteq X_j \).

3. By (1) and (2), \( UTP_{p_i}(S) = \{ \vec{u} \} \) and \( NFP_{i,j}(S) = \{ \vec{n}_j \} \) for \( j = 1, \ldots, k_i \). By definition, \( p_i(\vec{u}) = 1 \) and \( p_i(\vec{n}_j) = 1 \). Therefore, \( \vec{u} \) and \( \vec{n}_j \) differ at \( x^1_j \) for all \( j = 1, \ldots, k_i \).

Hence, the result follows.

The implication of Lemma A.3 is that, if \( p_i \) is a minterm, for every possible \( UTP_{p_i}(S) \) and \( NFP_{i,j}(S) \) pair, the only point from \( UTP_{p_i}(S) \) and the only point from \( NFP_{i,j}(S) \) differ only at the corresponding truth value of the literal \( x^1_j \). Hence, the CUTPNFP strategy can always select these points.

Lemma A.4 Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF and \( p_i \) be a minterm of \( S \). Then, \( UTP_{p_i}(S) = \{ \vec{u} \} \) and \( p_{i,j}(\vec{u}) = 0 \) for all \( 1 \leq i \leq i_1 \leq m \) and for all \( 1 \leq j \leq k_i \). where \( k_i \) denotes the number of literals in \( p_i \) and \( p_{i,j} \) denotes the term obtained from \( p_i \) by omitting its \( j \)-th literal \( x^1_j \).

Proof: Since \( p_i \) is a minterm, by Lemma A.3, \( UTP_{p_i}(S) = \{ \vec{u} \} \).

Let \( p_i \) be any term in \( S \) different from \( p_i \). By Lemma A.2, there are at least two literals in \( p_i \) whose negations can be found in \( p_i \). Without loss of generality, we may assume that these two literals are \( x^1_j \) and \( x^1_{j+1} \).

Otherwise, we can always rearrange the literals so that they become the first two literals of \( p_i \). Note that, both \( x^1_j \) and \( x^1_{j+1} \) evaluate to 0 on \( \vec{u} \) because their negations are in \( p_i \). Since \( x^{i+j} \) must exist in every \( p_{i,j} \), \( p_{i,j}(\vec{u}) = 0 \) where \( 1 \leq j \leq k_i \). Hence, the result follows.

Lemma A.5 Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF, and \( x \) be a literal of \( S \). If the \( i \)-th term of \( S \), \( p_i \), is not a minterm and there is at least one point in \( UTP_{p_i}(S) \) such that \( x = 0 \), then the MUTP strategy will select a point from \( UTP_{p_i}(S) \) such that \( x = 0 \).

Proof: Let \( p_i \) be the \( i \)-th term in \( S \). Since \( S \) is irredundant, \( UTP_{p_i}(S) \neq \emptyset \) by Lemma A.1. Since \( p_i \) is not a minterm, it has at least one missing literal. Therefore, the MUTP strategy can always be applied on \( p_i \).

Since \( x \) is a literal of \( S \), we have three cases:

1. The literal \( x \) appears in \( p_i \). Then, \( x = 1 \) on all points in \( UTP_{p_i}(S) \). This contradicts to the assumption that there is a point in \( UTP_{p_i}(S) \) such that \( x = 0 \). Hence, it is impossible for \( x \) to be a literal in \( p_i \).

2. The literal \( x \) appears in \( p_i \). Then, \( x = 0 \) on all points in \( UTP_{p_i}(S) \). Hence, the MUTP strategy can select a point from \( UTP_{p_i}(S) \) such that \( x = 0 \).

3. Both literals \( x \) and \( x \) do not appear in \( p_i \). Then, \( x \) is a missing literal of \( p_i \). Based on the given condition, there exist some points in \( UTP_{p_i}(S) \) such
that \( x \), as a missing literal, evaluates to 0. Since the MUTP strategy selects points in \( UTP_i(S) \) such that all possible truth values of every missing literal of \( p_i \) are covered. Hence, it can select at least a point from \( UTP_i(S) \) such that \( x \) evaluates to 0.

Hence, the result follows. \( \square \)

Let \( X_i \) be the set \( \{ \vec{t} \in UTP_i(S) : x \text{ evaluates to } 0 \text{ on } \vec{t} \} \) of all points \( \vec{t} \) in \( UTP_i(S) \) such that the literal \( x \) evaluates to 0 on \( \vec{t} \). Lemma A.5 shows that if \( p_i \) is not a minterm and \( X_1 \) is non-empty, the MUTP strategy will be able to select a test case from \( UTP_i(S) \) such that it is in \( X_1 \). As a result, test cases selected by the MUTP strategy can satisfy (C1) of Expression (29) provided that the corresponding condition is not a minterm and there exists some test cases that can satisfy the corresponding condition. Similarly, the MUTP strategy, if applicable, can select test cases that satisfy (C1) and (C3) of Expression (29).

**Lemma A.6** Let \( S = p_1 + \cdots + p_m \) be a Boolean expression in IDNF and \( p_i \) and \( p_j \) (\( i \neq j \)) be two different terms of \( S \). Suppose that \( p_i \) is not a minterm and that there is at least one point from \( UTP_{p_i}(S) \) such that \( p_{i,j} = 0 \) where \( p_{i,j} = x_i^0 \cdots x_{i,j-1} \bar{x}_{i,j} x_{i,j+1} \cdots x_{i,j} \) is the term obtained from \( p_j \) by omitting its \( j \)-th literal, \( x_{i,j} \). Then, the MUTP strategy will be able to select a point from \( UTP_{p_i}(S) \) such that \( p_{i,j} = 0 \).

**Proof**: Since \( S \) is irredundant, \( UTP_{p_i}(S) \neq \emptyset \) by Lemma A.1. Since \( p_i \) is not a minterm, it has at least one missing literal. Therefore, the MUTP strategy can always be applied on \( p_i \). Since \( p_i \) and \( p_j \) are two different terms of \( S \), we have the following two cases:

1. All literals of \( p_{i,j} \) are in \( p_i \). Then, \( p_{i,j} = 1 \) on all points in \( UTP_{p_i}(S) \) because all literals in \( p_i \) evaluate to 1. This contradicts the assumption. Hence, such a case is impossible.
2. There exists some literals of \( p_{i,j} \) such that they do not appear in \( p_i \). Two subcases arise:
   (a) Among these literals, there is at least one literal \( x \) such that \( \bar{x} \) appears in \( p_i \).
      Then, \( x \) will evaluate to 0 on all points from \( UTP_{p_i}(S) \). As a result, \( p_{i,j} \) evaluates to 0 on all points from \( UTP_{p_i}(S) \). Hence, the MUTP strategy can always select a point from \( UTP_{p_i}(S) \) such that \( p_{i,j} = 0 \).
   (b) Among these literals, all their negations do not appear in \( p_i \). Hence, all these literals are missing literals of \( p_i \).

Now, literals in \( p_{i,j} \) can be divided into two groups. The first group contains those literals that are in \( p_i \), whereas the second group contains those literals that are not in \( p_i \). Based on this subcase, the negations of all literals in the second group are also not in \( p_i \).

If all literals in the second group evaluate to 1 on all points of \( UTP_{p_i}(S) \), \( p_{i,j} \) evaluates to 1 on all points of \( UTP_{p_i} \). This contradicts the given condition that \( p_{i,j} \) evaluates to 0 on some point of \( UTP_{p_i} \).

Hence, there is at least one literal \( y \) in the second group such that it evaluates to 0 on some points of \( UTP_{p_i} \).

In summary, we have (1) both \( y \) and \( \bar{y} \) do not appear in \( p_i \); and (2) there exist some points in \( UTP_{p_i}(S) \) such that \( y \), as a missing literal, evaluates to 0. Since the MUTP strategy selects points from \( UTP_{p_i}(S) \) such that all possible truth values of every missing literal of \( p_i \) are covered, for every \( p_i \), \( S \) is non-empty. Hence, it can select at least a point from \( UTP_{p_i}(S) \) such that \( y = 0 \), and hence, \( p_{i,j} = 0 \).

The result follows. \( \square \)

Similar to Lemma A.5, let \( X_2 \) be the set \( \{ \vec{t} \in UTP_{p_1}(S) : p_{i,j} \text{ evaluates to } 0 \text{ on } \vec{t} \text{ where } i \neq j \} \) of all points \( \vec{t} \) in \( UTP_{p_1}(S) \) such that \( p_{i,j} \) evaluates to 0 on \( \vec{t} \). Lemma A.6 shows that if \( p_i \) is not a minterm and \( X_2 \) is non-empty, the MUTP strategy will be able to select a test case from \( UTP_{p_i}(S) \) such that it is in \( X_2 \). Hence, test cases selected by the MUTP strategy can satisfy (C2) of Expression (29) provided that the corresponding term is not a minterm and there exist test cases that satisfy the corresponding condition. Similarly, the MUTP strategy, if applicable, can select test cases that satisfy (C2) and (C4) of Expression (29).

We now prove Theorems 5.1 and 5.2 in the main text.

**Theorem 5.1** Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF. Suppose that the \( i \)-th term, \( p_i \), in \( S \) is omitted and the \( j \)-th term, \( x_{i,j} \), of the \( i \)-th term, \( p_i \), in \( S \) is replaced by \( x_{i,k} \), where \( 1 \leq i < j \leq m \), \( 1 \leq j \leq k_{i,j} \), and \( x_{i,j} \) is a missing literal of \( p_i \). Then the resulting implementation denoted by \( I \) will be equivalent to that given by Expression (22) in Table 2. The SMOTP strategy can supplement the MUMCUT strategy to guarantee the detection of Expression (22) provided that \( S \neq I \).

**Proof**: We will prove that the SMOTP and MUMCUT strategies can select test cases to collectively satisfy conditions (C1)--(C5) of Expression (22) in Table 2. Since the SMOTP strategy can only be applied when \( p_i \) is not a minterm, we have the following two cases:

1. The term \( p_i \) is a minterm.

   By Lemma A.3, \( UTP_{p_i}(S) = \{ \bar{i} \} \), \( NFP_{p_i,j}(S) = \{ \bar{i}_j \} \), and \( \bar{i} \) and \( \bar{i}_j \) only differ on the literal \( x_{i,j} \) for all \( j = 1, \ldots, k_{i,j} \) where \( k_{i,j} \) denotes the number of literals in the term \( p_i \). Therefore, the CUTFPNFP strategy will select all these points. By Lemma A.4, \( p_{i,j} \) evaluates to 0 on \( \bar{i} \). Therefore, (C2) can always be satisfied and the CUTFPNFP strategy can guarantee to select the required test cases.
2. The term \( p_1 \) is not a minterm.

Since \( S \neq I \), there is at least one test case \( \vec{t} \) such that \( \vec{t} \) satisfies any one of \((C1)–(C5)\). We have the following three subcases:

(a) The test case \( \vec{t} \) satisfies any one of the conditions \((C1)–(C3)\). We will prove that the MUTP strategy can select points that satisfy conditions \((C1)–(C3)\). However, the points actually selected may not be \( \vec{t} \).

If \( \vec{t} \) satisfies \((C1)\), the MUTP strategy can select test cases that satisfy this condition by Lemma A.5.

If \( \vec{t} \) satisfies \((C2)\), the MUTP strategy can select test cases that satisfy this condition by Lemma A.6.

If \( \vec{t} \) satisfies \((C3)\), it is a test case from \( UTP_{i_2}(S) \) such that the missing literal \( x_{i_2} \) of \( p_{i_2} \) evaluates to 0. Since the MUTP strategy requires to select points from \( UTP_{i_2}(S) \) such that all possible truth values of every missing literal of \( p_i \) are covered, it will select test cases that satisfy \((C3)\).

(b) The test case \( \vec{t} \) satisfies condition \((C4)\). We will prove that the MNFP strategy can select points that satisfy \((C4)\). However, the points actually selected may not be \( \vec{t} \).

Now, \( \vec{t} \) is a test case from \( NFP_{i_2,j_2}(S) \) such that the missing literal \( x_{i_2} \) of \( p_{i_2} \) evaluates to 1. Since the MNFP strategy requires to select points from \( NFP_{i_2,j_2}(S) \) such that all possible truth values of every missing literal of \( p_i \) are covered, it can select test cases that satisfy \((C4)\).

(c) The test case \( \vec{t} \) satisfies condition \((C5)\). We will prove that the SMOTP strategy can select points that satisfy \((C5)\) although the points actually selected may not be \( \vec{t} \).

Note that, \( \vec{t} \) is a test case from \( (TP_{i_1}(S) \cap TP_{i_2}(S)) \setminus \bigcup_{i=1 \atop i \neq i_1,i_2} TP_i(S) \) such that the missing literal \( x_{i_2} \) of \( p_{i_2} \) evaluates to 0. Now, since \( p_{i_1} \) is not a minterm, it has at least one missing literal. Suppose that \( x_{i_1} \) is a missing literal of \( p_{i_1} \).

The SMOTP strategy requires to select test cases that can collectively cover every possible truth value combinations of the missing literals \( x_{i_1} \) and \( x_{i_2} \) of \( p_{i_1} \) and \( p_{i_2} \), respectively, from \( (TP_{i_1}(S) \cap TP_{i_2}(S)) \setminus \bigcup_{i=1 \atop i \neq i_1,i_2} TP_i(S) \) for every two different terms \( p_{i_1} \) and \( p_{i_2} \).

As a result, it is straightforward to see that the SMOTP strategy can select test cases that satisfy \((C5)\).

Hence, the SMOTP, MUTP and MNFP strategies together can always select test cases that satisfy \((C1)–(C5)\) provided that \( p_1 \) is not a minterm. The result follows.

\[\square\]

**Theorem 5.2** Let \( S = p_1 + \cdots + p_m \) be a Boolean specification in IDNF. Suppose that two terms \( p_{i_1} \) and \( p_{i_1+1} \) in \( S \) are implemented as \( p_{i_1}p_{i_1+1} \) and the \( j_2\)-th literal, \( x_{j_2}^2 \), of the \( i_2\)-th term, \( p_{i_2} \), in \( S \) is replaced by \( x_{i_2} \), where \( 1 < i_1 + 1 < i_2 \leq m \), \( 1 \leq j_2 \leq k_{i_2} \), and \( x_{i_2} \) is a missing literal of \( p_{i_2} \). Then the resulting implementation denoted by \( I \) will be equivalent to that given by Expression (29) in Table 2. The SMOTP strategy can supplement the MUMCUT strategy to guarantee the detection of Expression (29) provided that \( S \neq I \).

**Proof**: We will prove that the SMOTP and MUMCUT strategies can select test cases to collectively satisfy the conditions \((C1)–(C8)\) of Expression (29) in Table 2. Since the SMOTP strategy can only be applied when \( p_{i_1} \) or \( p_{i_1+1} \) is not a minterm, we have the following three cases:

1. \( p_{i_1} \) is a minterm.

   Condition \((C2)\) can always be satisfied and the CUTPNFP strategy guarantees to select the required test cases. The proof is similar to that of Case 1 in Theorem 5.1.

2. \( p_{i_1+1} \) is a minterm.

   Condition \((C4)\) can always be satisfied and the CUTPNFP strategy guarantees to select the required test cases. The proof is similar to Case 1 above.

3. Both \( p_{i_1} \) and \( p_{i_1+1} \) are not minterms.

   Since \( S \neq I \), there is at least one test case \( \vec{t} \) such that \( \vec{t} \) satisfies any one of \((C1)–(C8)\). We have the following three subcases:

   (a) The test case \( \vec{t} \) satisfies any one of \((C1)–(C5)\).

   We will prove that the MUTP strategy can select points that satisfy \((C1)–(C5)\). However, the points selected may not be \( \vec{t} \). The proof is similar to that of Case 2(a) in Theorem 5.1.

   (b) The test case \( \vec{t} \) satisfies \((C6)\).

   We will prove that the MNFP strategy can select points that satisfy \((C6)\). The proof is similar to that of Case 2(b) in Theorem 5.1.

   (c) The test case \( \vec{t} \) satisfies any one of \((C7)–(C8)\).

   We will prove that the SMOTP strategy can select points that satisfy any one of \((C7)\) and \((C8)\). The proof is similar to that of Case 2(c) in Theorem 5.1.

Hence, the SMOTP, MUTP and MNFP strategies together can always select test cases that satisfy \((C1)–(C8)\) provided that both \( p_{i_1} \) and \( p_{i_1+1} \) are not minterms.

The result follows.

\[\square\]