

Faculty of Information and Communication Technologies

# **Relationship of Double Literal Faults in Boolean Expressions Was: The Investigation of Double Faults Related to Literals**

Technical Report: SUTICT-TR2006.03

Man Fai Lau and Ying Liu

First version: 30th June 2006

Last revised: 30th Nov 2007 [Renummer the expression ids]



**SWIN  
BUR  
NE**  
\* \* \* \*

SWINBURNE UNIVERSITY  
OF TECHNOLOGY

This page is intentionally left blank.

# Relationship of Double Literal Faults in Boolean Expressions

## Table of contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Preliminary</b>	<b>3</b>
2.1	Notation . . . . .	3
2.2	Single Literal Fault Classes . . . . .	4
<b>3</b>	<b>Double Faults without Ordering</b>	<b>6</b>
3.1	LNF with Other Faults . . . . .	7
3.2	LOF with Other Faults . . . . .	11
3.3	LIF with Other Faults . . . . .	13
3.4	LRF with LRF . . . . .	15
<b>4</b>	<b>Double Faults with Ordering</b>	<b>16</b>
4.1	LNF first, then Other Faults . . . . .	16
4.2	LOF first, then Other Faults . . . . .	19
4.3	LIF first, then Other Faults . . . . .	22
4.4	LRF first, then Other Faults . . . . .	25
<b>5</b>	<b>Relation between Double Faults with and without Ordering</b>	<b>29</b>
<b>6</b>	<b>Conclusion</b>	<b>32</b>

This page is intentionally left blank.

# 1 Introduction

Fault-based testing techniques which aim at detecting hypothesized faults in a program have been studied for many years [1, 4, 8, 10, 12, 14]. More precisely, if any hypothesized faults occur in the program, test cases generated by fault-based testing techniques can detect those faults. Most studies related to fault-based testing techniques assume only one fault being occurred in the program [10]. However, it is quite common that programmer makes mistakes during code development which in turn injects multiple faults. Moreover, empirical investigations show that multiple faults occur more frequently in practice [9, 13]. Besides, the study on single faults cannot clearly reveal characteristics of multiple faults. For example, when multiple faults occur in a program, they may interact in such a way that they cannot be detected by those test cases that can detect the original individual faults. Hence, the study of multiple faults helps us to better understand how the multiple faults are committed and how they affect each other.

Previous work on multiple faults mainly studies the coupling effect of double faults [2, 3, 11]. In [11], an empirical study of fault coupling via mutation analysis was performed. A *1-order mutant* (respectively, *2-order mutant*) is a program 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 (LOCs) were studied. Test sets that killed all 1-order mutants were generated and used to kill 2-order mutants. As mentioned in [11], the experiment was a study of the *mutation coupling effect* instead of fault coupling effect. 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 coupled together rarely occurred.

In [2, 3], How Tai Wah investigated fault coupling from a theoretical perspective. A program is modelled as a composition of mathematical functions. A program with a single fault (respectively, double fault) is modelled as an incorrect use of one (respectively, two) of the functions. Test sets that detect the 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*. The *coupling ratio* of the proper test sets is defined as the ratio of the number of coupled test sets to that of proper test sets. He proves that, if  $|D|$  is the size of the input domain  $D$  of a program, the coupling ratio is approximately  $1/|D|$  and  $1/|D|^2$  for proper test sets of size 1 and 2, respectively. As  $|D|$  is usually very large, the coupling ratio is very small. He then concluded that fault coupling rarely occurred.

Previous studies on fault coupling of double faults assume that the two individual single faults are considered to be independently committed on the program source [11] or the functions that model a program [2, 3]. However, previous studies have not considered the situation that two single faults involved in a double fault may occur in such a way that the first fault may affect the occurrence of the second one.

In [5], Lau and Liu have shown that there are two ways in which a double fault can occur. The first one, referred to as *double faults without ordering*, is that the two single faults are independent to each other. The second one, referred to as *double faults with ordering*, is that the first fault affects the occurrence of the second fault. They studied the double faults related to terms from these two different ways. They find that, among the 15 types of double fault without ordering, there are 27 distinct non-equivalent faulty implementations. For the 25 types of double faults with ordering, there are 53 faulty implementations. Then, they further investigated the relationship between these two categories, and found that 49 out of 53 faulty implementations of double fault with ordering are equivalent to those 27 faulty implementations of double fault without ordering. Only 4 faulty implementations of double fault with ordering do not have their equivalent counterparts in double fault without ordering. In summary, there are altogether 31 different faulty implementations for these two categories of double faults. More precisely, the 31 different faulty implementations can represent all possible double faults related to terms.

As reported in [5], the ordering of the two single faults may actually result in two different double faults. Hence, we study the double faults related to literals in Boolean expressions from double faults with and without ordering. Moreover, since the two single faults committed in different ordering

may result in different faulty implementations as shown in [5], we investigate the relationship of these faulty implementations. Through the study, we obtain better understanding on how double faults occur within Boolean expressions and how they interact with each other based on the faulty implementations. Furthermore, we obtain the faulty implementations of all possible double faults studied in this report, which will help us to study the corresponding test cases selection strategies to detect all these double faults.

The rest of the report is organized as follows. Section 2 introduces the notation and fault classes studied in this report. Sections 3 and 4 present different categories of double fault classes and their corresponding faulty implementations. Section 5 studies the relationships of double faults defined in Sections 3 and 4. Section 6 concludes the report.

## 2 Preliminary

### 2.1 Notation

In this report, we use ‘.’, ‘+’ and ‘-’ to represent Boolean operators, AND, OR and NOT, respectively. Usually, ‘.’ is omitted whenever it is clear from the context. 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 by  $\mathbb{B}$ .

Let  $S$  be a Boolean expression in irredundant disjunctive normal form

$$S = p_1 + \cdots + p_m$$

where  $m$  is the number of terms in  $S$ ,  $p_i = x_1^i \cdots x_{k_i}^i$  is the  $i$ -th term of  $S$ ,  $x_j^i$  is the  $j$ -th literal in  $p_i$ , and  $k_i$  is the number of literals in  $p_i$ . A Boolean expression is in *irredundant disjunctive normal form* (IDNF) if (1) none of its terms can be omitted from the expression; and (2) none of its literals can be omitted from any term in the expression. A literal  $x$  is a *missing literal* of a term  $p_i$  if both  $x$  and

$\bar{x}$  do not appear in the term. Let  $S$  be a Boolean expression having  $n$  variables, the input domain is the  $n$ -dimensional Boolean space  $\mathbb{B}^n$ . A test case for  $S$  is a point in  $\mathbb{B}^n$ .

*True points* of  $S$  are those that make  $S$  evaluate to 1. The set of all true points of  $S$  is denoted by  $TP(S)$ . 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  $TP_i(S)$ . Hence,  $TP(S) = \bigcup_i TP_i(S)$ . A *unique true point* of  $p_i$  in  $S$  is a true point of  $S$  such that (1)  $p_i$  evaluates 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  $UTP(S) = \bigcup_i UTP_i(S)$ .

*False points* of  $S$  are those that make  $S$  evaluate to 0 and the set of all false points is denoted by  $FP(S)$ . A *near false point* of the  $j$ -th literal  $x_j^i$  of the  $i$ -th term  $p_i$  in  $S$  is a false point of  $S$  such that (1)  $x_j^i$  evaluates to 0, and (2) all other literals in  $p_i$  evaluate to 1. The set of all near false points for the  $j$ -th literal  $x_j^i$  of the  $i$ -th term  $p_i$  in  $S$  is denoted by  $NFP_{i,\bar{j}}(S)$ . The set of all near false points for the  $i$ -th term  $p_i$  in  $S$  is denoted by  $NFP_i(S)$ . Therefore,  $NFP_i(S) = \bigcup_j NFP_{i,\bar{j}}(S)$ . The set of all near false points of  $S$  is denoted by  $NFP(S)$  and  $NFP(S) = \bigcup_i NFP_i(S)$ .

## 2.2 Single Literal Fault Classes

Let  $S$  be a Boolean expression in IDNF. Suppose a fault  $F$  changes a subexpression  $E$  of  $S$  into a subexpression  $E'$ . The resulting faulty implementation is denoted as  $I_{F(E \rightarrow E')}$ . A faulty implementation is referred to as *single-fault expression* if (1) it differs from the original expression by one syntactic change; and (2) it is not equivalent to the original expression.

Lau and Yu [8] have clarified various common types of single faults that may be committed in Boolean expressions in various research literature [1, 4, 7, 12, 14] and proposed nine different fault classes. Among these single fault classes, five of them are related to terms in Boolean expressions whereas the remaining four are literal faults. Lau *et al.* [6] analyzed the detection conditions of

double faults involving five single fault classes related to terms. They found that all those double faults that can be detected by any test case selection strategies that subsume the BASIC meaningful impact strategy (or, simply the BASIC strategy) proposed in [14].

In this report, we consider double faults involving the remaining four literal faults in Boolean expressions. They are

1. *Literal Negation Fault (LNF)*: A literal in a particular term in the Boolean expression is replaced by its negation. For example, the Boolean expression  $ab + cd + ef$  may be wrongly implemented as  $\bar{a}b + cd + ef$ . As reported in [8], when a term contains just one literal, the negation fault is considered as a term negation fault rather than a literal negation fault. Hence, without loss of generality, we may assume that  $k_i > 1$ . If the  $j$ -th literal,  $x_j^i$ , of the  $i$ -th term,  $p_i$ , of  $S$  is wrongly implemented as its negation, the implementation, denoted as  $I_{LNF(p_i \rightarrow p_{i,\bar{j}})}$ , is then equivalent to  $p_1 + \dots + p_{i-1} + p_{i,\bar{j}} + p_{i+1} + \dots + p_m$  where  $p_{i,\bar{j}} = x_1^i \cdots \bar{x}_j^i \cdots x_{k_i}^i$  can be obtained from  $p_i$  by negating its  $j$ -th literal  $x_j^i$ .
2. *Literal Omission Fault (LOF)*: A literal in a particular term in the Boolean expression is omitted from the term. For example, the Boolean expression  $ab + cd + ef$  may be implemented as  $ab + cd + e$ . Similar to the situation of LNF, as reported in [8], when a term contains just one literal, the omission fault is considered as a term omission fault rather than a literal omission fault. Hence, without loss of generality, we may assume that  $k_i > 1$ . If the  $j$ -th literal  $x_j^i$  of the  $i$ -th term,  $p_i$ , of  $S$  ( $1 \leq j \leq k_i$  and  $k_i > 1$ ) is omitted from  $p_i$ , the implementation, denoted as  $I_{LOF(p_i \rightarrow p_{i,\hat{j}})}$ , is then equivalent to  $p_1 + \dots + p_{i-1} + p_{i,\hat{j}} + p_{i+1} + \dots + p_m$  where  $p_{i,\hat{j}} = x_1^i \cdots x_{j-1}^i \cdot x_{j+1}^i \cdots x_{k_i}^i$  can be obtained from  $p_i$  by omitting its  $j$ -th literal  $x_j^i$ .
3. *Literal Insertion Fault (LIF)*: A missing literal of a particular term of a Boolean expression is inserted into that term. For example, the Boolean expression  $ab + cd + ef$  may be implemented as  $abc + cd + ef$ . If a literal  $x_l$  not appearing in the  $i$ -th term,  $p_i$ , of  $S$  is inserted into  $p_i$  (that is, both  $x_l, \bar{x}_l$  cannot be found in  $\{x_1^i, \dots, x_{k_i}^i\}$ ), the implementation, denoted as

$I_{LIF(p_i \rightarrow p_i x_l)}$ , is then equivalent to  $p_1 + \dots + p_{i-1} + p_i x_l + p_{i+1} + \dots + p_m$ .

4. *Literal Reference Fault* (LRF): A literal in a particular term of a Boolean specification is replaced by a missing literal of the term during the implementation. For example, the Boolean expression  $ab + cd + ef$  may be implemented as  $ac + cd + ef$ . If the  $j$ -th literal,  $x_j^i$ , in the  $i$ -th term,  $p_i$ , of  $S$  is replaced by one of its missing literal,  $x_l$ , (that is, both  $x_l$  and  $\bar{x}_l$  do not occur in  $\{x_1^i, \dots, x_{k_i}^i\}$ ), the implementation, denoted as  $I_{LRF(p_i \rightarrow p_i \hat{x}_l)}$ , is then equivalent to  $p_1 + \dots + p_{i-1} + p_i \hat{x}_l + p_{i+1} + \dots + p_m$ , where  $p_i \hat{x}_l = x_1^i \cdots x_{j-1}^i \cdot x_{j+1}^i \cdots x_{k_i}^i x_l$  can be obtained from  $p_i$  by replacing its  $j$ -th literal  $x_j^i$  with  $x_l$ . Here,  $x_j^i$  and  $x_l$  are called *replaced literal* and *replacing literal*, respectively.

### 3 Double Faults without Ordering

When multiple occurrences of any fault classes happen in a program, the resulted program may differ from the original (correct) program by several syntactic changes. Such program is believed to include multiple faults. In this report, we concentrate on double literal faults which is two occurrences of those single literal faults discussed in Section 2.

When a double literal fault occurs in a Boolean expression, the resulting expression may be equivalent to the original expression or a faulty expression with one syntactic change. For example, if a literal of a particular term in  $S$  is negated twice, the resulting expression is equivalent to  $S$ . On the other hand, if a literal of a particular term in  $S$  is first negated and then omitted from the term, the net result of these two faults is equivalent to a single LOF being committed on the literal. In this report, we do not study the detection conditions of these expressions because they have been studied in previous research work [1, 8]. Therefore, a *double-fault expression* is an expression such that (1) it differs from the original expression by two syntactic changes and (2) it is equivalent to neither the original expression nor any single-fault expression.

In [5], Lau and Liu showed that there are two different ways in which double faults can be manifested. First, the two faults may be independent to each other, resulting the same faulty expression no matter which fault is committed first. It is referred to as *double faults without ordering*. Second, the two faults may occur in such way that the first one affect the occurrence of the second. Such a case is referred to as *double faults with ordering*. Lau and Liu studied double faults related to terms from such two ways and found that there are 15 classes of double faults without ordering, resulting in 27 possible different faulty expressions. For double faults with ordering, there are 25 classes which resulting in 53 possible faulty expressions. They found that 49 out of the 53 double-fault expressions are equivalent some of the expressions that are results of the double fault without ordering. Only 4 faulty expressions due to double faults with ordering do not have their equivalent counterparts in double faults without ordering.

By studying the occurrence of double faults from double faults with and without ordering, a better understanding on their similarities and differences is achieved. Moreover, it helps to better understand how two single faults occur in a double fault and how they interact with each other. Hence, in this report, we study double literal fault with and without ordering.

In the rest of this section, we will introduce different types of double fault without ordering. Given a Boolean expression  $S$ , suppose that two single fault classes  $F_1$  and  $F_2$  are committed in  $S$  changing its subexpressions  $E_1$  and  $E_2$  to  $E'_1$  and  $E'_2$ , respectively, the resulting double-fault implementation is denoted by  $I_{F_1(E_1 \rightarrow E'_1), F_2(E_2 \rightarrow E'_2)}$  because their order of occurrences will result in the same faulty implementation. Table 1 lists all 10 types of double faults without ordering for those four single fault classes discussed in Section 2.2.

### 3.1 LNF with Other Faults

**LNF and LNF** Let  $S = p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively. Sup-

Table 1: Types of Double Faults without Ordering

	LNF	LOF	LIF	LRF
LNF	✓	✓	✓	✓
LOF		✓	✓	✓
LIF			✓	✓
LRF				✓

pose that both literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are negated. We use  $I_{LNF(p_{i_1} \rightarrow p_{i_1, \bar{j}_1}), LNF(p_{i_2} \rightarrow p_{i_2, \bar{j}_2})}$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ .<sup>1</sup> Furthermore, without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \bar{j}_2} + \dots + p_m \quad (1)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when a literal is negated twice, the implementation is then equivalent to the original expression  $S$ . Without loss of generality, we can assume  $j_1 < j_2$ .<sup>2</sup> The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1, \bar{j}_2} + \dots + p_m \quad (2)$$

where  $p_{i_1, \bar{j}_1, \bar{j}_2} = x_1^{i_1} \dots \bar{x}_{j_1}^{i_1} \dots \bar{x}_{j_2}^{i_1} \dots x_{k_{i_1}}^{i_1}$  denotes the term obtained from  $p_{i_1}$  by negating its literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  ( $j_1 < j_2$ ).

<sup>1</sup>Otherwise, if a term contains just one literal, the negation fault is considered to be a term negation fault, which is studied in other reports and is out of the scope of this report. In the sequel, we will make similar notes related to negation and omission faults, and the reason is similar.

<sup>2</sup>This implies  $k_{i_1} > 1$  because  $1 \leq j_1 < j_2 \leq k_{i_1}$ .

**LNF and LOF** Let  $S = p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively. Suppose that the literal  $x_{j_1}^{i_1}$  is negated and the literal  $x_{j_2}^{i_2}$  is omitted. We use  $I_{LNF(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})}$ ,  $LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we can further assume  $i_1 < i_2$ . Otherwise, we can always interchange the two terms so that the term with LOF comes after that of LNF.<sup>3</sup> The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \hat{j}_2} + \dots + p_m \quad (3)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when  $x_{j_1}^{i_1}$  is first negated and then omitted, the implementation is then equivalent to a single LOF with respect to the original expression  $S$ . On the other hand, if  $x_{j_2}^{i_2}$  is first omitted, it is impossible to negate it afterwards. As a result,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are two different literals. Without loss of generality, we can assume  $j_1 < j_2$ . Otherwise, we can always interchange the two literals, so that the literal with LOF comes after that of LNF.<sup>4</sup> The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1, \hat{j}_2} + \dots + p_m \quad (4)$$

where  $p_{i_1, \bar{j}_1, \hat{j}_2} = x_1^{i_1} \dots \bar{x}_{j_1}^{i_1} \dots x_{j_2-1}^{i_1} x_{j_2+1}^{i_1} \dots x_{k_{i_1}}^{i_1}$  denotes the term obtained from  $p_{i_1}$  by negating its literal  $x_{j_1}^{i_1}$  and omitting its literal  $x_{j_2}^{i_2}$  ( $j_1 < j_2$ ).

<sup>3</sup>In the rest of this report, we will make similar assumption whenever we encounter situations with two faults committed at two different terms.

<sup>4</sup>In the rest of this report, we will make similar assumption whenever we encounter situations with two faults committed at two different literals in the same term.

**LNF and LIF** Let  $S=p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  be a literal in the  $i_1$ -th term,  $p_{i_1}$ , of  $S$  and  $x_{l_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $S$ . Suppose that the literal  $x_{j_1}^{i_1}$  is negated and the literal  $x_{l_2}$  is inserted into  $p_{i_2}$ . We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ ,  $LIF(p_{i_2} \rightarrow p_{i_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2} x_{l_2} + \dots + p_m \quad (5)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1} x_{l_2} + \dots + p_m \quad (6)$$

**LNF and LRF** Let  $S=p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively, and  $x_{l_2}$  be a missing literal of  $p_{i_2}$  in  $S$ . Suppose the literal  $x_{j_1}^{i_1}$  is negated and the literal  $x_{j_2}^{i_2}$  is replaced by the literal  $x_{l_2}$ . We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ ,  $LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \hat{j}_2} x_{l_2} + \dots + p_m \quad (7)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when  $x_{j_1}^{i_1}$  is first negated and then replaced by  $x_{i_2}$ , the implementation is then equivalent to a single LRF with respect to the original expression  $S$ . On the other hand, if  $x_{j_2}^{i_1}$  is first replaced by  $x_{i_2}$ , it is impossible to negate it afterwards. Without loss of generality, we can assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{i_2} + \cdots + p_m \quad (8)$$

### 3.2 LOF with Other Faults

**LOF and LOF** Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively. Suppose that both literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are omitted. We use  $I_{LOF(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})}$ ,  $LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2, \hat{j}_2} + \cdots + p_m \quad (9)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when a literal is first omitted, it is impossible to omit it afterwards. Furthermore, we do not consider the situation when  $p_{i_1}$  contains just only two literals because the net effect will be equivalent to a term omission fault, which is a single term fault. Without loss of generality, we can assume  $j_1 < j_2$  and  $k_{i_1} > 2$ . The implementation is then equivalent

to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} + \cdots + p_m \quad (10)$$

where  $p_{i_1, \hat{j}_1, \hat{j}_2} = x_1^{i_1} \cdots x_{j_1-1}^{i_1} x_{j_1+1}^{i_1} \cdots x_{j_2-1}^{i_1} x_{j_2+1}^{i_1} \cdots x_{k_{i_1}}^{i_1}$  denotes the term obtained from  $p_{i_1}$  by omitting its literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  ( $j_1 < j_2$ ).

**LOF and LIF** Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  be a literal in the  $i_1$ -th term,  $p_{i_1}$ , of  $S$ , and  $x_{l_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $S$ . Suppose the literal  $x_{j_1}^{i_1}$  is omitted and the literal  $x_{l_2}$  is inserted into  $p_{i_2}$  of  $S$ . We use  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})$ ,  $LIF(p_{i_2} \rightarrow p_{i_2} x_{l_2})$  to denote the corresponding faulty implementation.

We do not consider the situation when the LOF and LIF are committed at the same term (that is,  $i_1 = i_2$ ). It is because when  $x_{j_1}^{i_1}$  is omitted from  $p_{i_1}$  and  $x_{l_2}$  is inserted into  $p_{i_1}$ , the implementation is then equivalent to a single LRF with respect to the original expression  $S$ . Therefore, two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2} x_{l_2} + \cdots + p_m \quad (11)$$

**LOF and LRF** Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively, and  $x_{l_2}$  be a missing literal of the  $p_{i_2}$  in  $S$ . Suppose the literal  $x_{j_1}^{i_1}$  is omitted and the literal  $x_{j_2}^{i_2}$  is replaced by the literal  $x_{l_2}$ . We use  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})$ ,  $LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ .

Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent

to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (12)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when a literal is first omitted, it is impossible to replace it afterwards. On the other hand, if a literal is first replaced, it is impossible to omit the literal afterwards. Without loss of generality, we assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (13)$$

### 3.3 LIF with Other Faults

**LIF and LIF** Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{l_1}$  and  $x_{l_2}$  be two missing literals of the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively. Suppose two literals  $x_{l_1}$  and  $x_{l_2}$  are inserted into  $p_{i_1}$  and  $p_{i_2}$ , respectively. We use  $I_{LIF(p_{i_1} \rightarrow p_{i_1} x_{l_1})}$ ,  $LIF(p_{i_2} \rightarrow p_{i_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1} x_{l_1} + \cdots + p_{i_2} x_{l_2} + \cdots + p_m \quad (14)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the following two situations. First, two literals  $x_{l_1}$  and  $x_{l_2}$  are exactly the same (that is  $x_{l_1} = x_{l_2}$ ). It is because when a literal is inserted into a term twice, the implementation is then equivalent to a single LIF with respect to the original expression  $S$ . Second, the two

literals  $x_{l_1}$  and  $x_{l_2}$  are negations of each other (that is  $x_{l_1} = \bar{x}_{l_2}$ ). It is because when a literal and its negation are inserted into a term, the implementation is then equivalent to a single TOF with respect to the original expression  $S$ . Hence, without considering the above two situations,  $x_{l_1}$  and  $x_{l_2}$  are two different missing literals of  $p_{i_1}$  and are from different Boolean variables. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}x_{l_1}x_{l_2} + \cdots + p_m \quad (15)$$

**LIF and LRF** Let  $S=p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_2}^{i_2}$  be a literal in the  $i_2$ -th term,  $p_{i_2}$ , of  $S$ , and  $x_{l_1}$  and  $x_{l_2}$  be two missing literals of the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively. Suppose the literal  $x_{l_1}$  is inserted into  $p_{i_1}$  and the literal  $x_{j_2}^{i_2}$  is replaced by the literal  $x_{l_2}$ . We use  $I_{LIF}(p_{i_1} \rightarrow p_{i_1}x_{l_1})$ ,  $LRF(p_{i_2} \rightarrow p_{i_2,\hat{j}_2}x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}x_{l_1} + \cdots + p_{i_2,\hat{j}_2}x_{l_2} + \cdots + p_m \quad (16)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the two situations where the literals  $x_{l_1}$  and  $x_{l_2}$  are exactly the same or are negations of each other. First, when  $x_{l_1}$  and  $x_{l_2}$  are exactly the same, the net effect of inserting  $x_{l_1}$  into  $p_{i_1}$  and replacing  $x_{j_2}^{i_1}$  by  $x_{l_1}$  is equivalent to a single LRF with  $x_{j_2}^{i_1}$  in  $S$  being replaced by  $x_{l_1}$ . Second, when  $x_{l_1}$  and  $x_{l_2}$  are negations of each other (that is  $x_{l_1} = \bar{x}_{l_2}$ ), the net effect of inserting  $x_{l_1}$  into  $p_{i_1}$  and replacing  $x_{j_2}^{i_1}$  by  $\bar{x}_{l_1}$  is equivalent to a single TOF with  $p_{i_1}$  in  $S$  being omitted. Hence, we only consider situations where  $x_{l_1}$  and  $x_{l_2}$  are two different missing literals of  $p_{i_1}$ . As a result, the implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,\hat{j}_2}x_{l_1}x_{l_2} + \cdots + p_m \quad (17)$$

### 3.4 LRF with LRF

Let  $S = p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  be two literals in the  $i_1$ -th and  $i_2$ -th terms,  $p_{i_1}$  and  $p_{i_2}$ , of  $S$ , respectively, and  $x_{l_1}$  and  $x_{l_2}$  be two missing literals of  $p_{i_1}$  and  $p_{i_2}$ , respectively. Suppose that both literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are replaced by literals  $x_{l_1}$  and  $x_{l_2}$ , respectively. We use  $I_{LRF(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})}$ ,  $LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two terms  $p_{i_1}$  and  $p_{i_2}$  are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \hat{j}_1} x_{l_1} + \dots + p_{i_2, \hat{j}_2} x_{l_2} + \dots + p_m \quad (18)$$

Case 2. The two terms  $p_{i_1}$  and  $p_{i_2}$  are exactly the same, that is  $i_1 = i_2$ . We do not consider the situation that  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are exactly the same (that is,  $j_1 = j_2$ ). It is because when the literal  $x_{j_1}^{i_1}$  is first replaced by another literal, it is impossible to replace  $x_{j_1}^{i_1}$  again. Therefore,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are different. Without loss of generality, we can assume  $j_1 < j_2$ . Furthermore, we do not consider the situation when  $x_{l_1}$  and  $x_{l_2}$  are negations of each other (that is  $x_{l_1} = \bar{x}_{l_2}$ ). It is because when  $x_{l_1}$  replaces  $x_{j_1}^{i_1}$  and  $\bar{x}_{l_1}$  replaces  $x_{j_2}^{i_2}$ , the implementation is then equivalent to a single TOF with respect to the original expression  $S$ . As a result, we have the following two subcases:

(a) The two literals  $x_{l_1}$  and  $x_{l_2}$  are different (that is  $x_{l_1} \neq x_{l_2}$ ). The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_1} x_{l_2} + \dots + p_m \quad (19)$$

(b) The two literals  $x_{l_1}$  and  $x_{l_2}$  are exactly the same (that is  $x_{l_1} = x_{l_2}$ ). The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_1} + \cdots + p_m \quad (20)$$

It should be noted that Expression (20) is equivalent to that of (13), which corresponds to Case 2 of the LOF and LRF.

In summary, there are all together 19 different double fault expressions among the 10 double fault classes without ordering because Expressions (13) and (20) are equivalent.

## 4 Double Faults with Ordering

In this section, we will discuss double faults with ordering and their corresponding double-fault expressions. As mentioned previously, double faults with ordering is such that two single faults occur one after the other in such a way that the occurrence of the first fault may affect the occurrence of the second fault. For four classes of single fault studied in this report, there are 16 classes of double fault with ordering.

Given a Boolean expression  $S$ , let  $F_1$  and  $F_2$  be two single fault classes changing the subexpressions  $E_1$  and  $E_2$  of  $S$  to  $E'_1$  and  $E'_2$ , respectively. Suppose that  $F_1$  is committed before  $F_2$ , the resulting implementation is denoted by  $I_{F_1(E_1 \rightarrow E'_1) \otimes F_2(E_2 \rightarrow E'_2)}$ .

### 4.1 LNF first, then Other Faults

Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  be a literal in the  $i_1$ -th term,  $p_{i_1}$ , of  $S$ . Suppose the literal  $x_{j_1}^{i_1}$  is negated. The corresponding faulty implementation, denoted as  $I_{LNF(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})}$ , is then equivalent to  $p_1 + \cdots + p_{i_1, \bar{j}_1} + \cdots + p_m$ .

**LNF and LNF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ . After the first LNF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then negated. We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1}) \otimes LNF(p_{i_2} \rightarrow p_{i_2, \bar{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1} + \cdots + p_{i_2, \bar{j}_2} + \cdots + p_m \quad (21)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation where the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is  $j_1 = j_2$ ). It is because when a literal is negated twice, the implementation is then equivalent to the original expression  $S$ . Without loss of generality, we assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, \bar{j}_2} + \cdots + p_m \quad (22)$$

**LNF and LOF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ . After the first LNF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then omitted from  $p_{i_2}$ . We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1}) \otimes LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1} + \cdots + p_{i_2, \hat{j}_2} + \cdots + p_m \quad (23)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation where the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is  $j_1 = j_2$ ). It is because when

$x_{j_1}^{i_1}$  is negated and then omitted, the implementation is then equivalent a single LOF with respect to the original expression  $S$ . Without loss of generality, we assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, \hat{j}_2} + \cdots + p_m \quad (24)$$

**LNF and LIF** Let  $x_{i_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ . After the first LNF is made on  $S$ , the literal  $x_{i_2}$  is then inserted into  $p_{i_2}$ . We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1}) \otimes LIF(p_{i_2} \rightarrow p_{i_2, x_{i_2}})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ .

Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1} + \cdots + p_{i_2, x_{i_2}} + \cdots + p_m \quad (25)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, x_{i_2}} + \cdots + p_m \quad (26)$$

**LNF and LRF** Let  $x_{j_2}^{i_2}$  and  $x_{i_2}$  be a literal and a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1})$ , respectively. After the first LNF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  from  $p_{i_2}$  is then replaced by the literal  $x_{i_2}$ . We use  $I_{LNF}(p_{i_1} \rightarrow p_{i_1, \bar{j}_1}) \otimes LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2, x_{i_2}})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ .

Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to

the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1} + \cdots + p_{i_2, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (27)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are exactly the same (that is  $j_1 = j_2$ ). It is because when  $x_{j_1}^{i_1}$  is negated and then replaced by  $x_{l_2}$ , the implementation is then equivalent to a single LRF with respect to the original expression  $S$ . Without loss of generality, we assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (28)$$

## 4.2 LOF first, then Other Faults

Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  be a literal of the  $i_1$ -th term,  $p_{i_1}$ , of  $S$ . Suppose the literal  $x_{j_1}^{i_1}$  is omitted from  $p_{i_1}$ . The corresponding faulty implementation, denoted as  $I_{LOF(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})}$ , is then equivalent to  $p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_m$ .

**LOF and LNF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LOF(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})}$ . After the first LOF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then negated. We use  $I_{LOF(p_{i_1} \rightarrow p_{i_1, \hat{j}_1}) \otimes LNF(p_{i_2} \rightarrow p_{i_2, \bar{j}_2})}$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2, \bar{j}_2} + \cdots + p_m \quad (29)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . Since  $x_{j_1}^{i_1}$  is omitted,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are different ( $j_1 \neq j_2$ ). Without loss of generality, we assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \bar{j}_2} + \cdots + p_m \quad (30)$$

where  $p_{i_1, \hat{j}_1, \bar{j}_2} = x_1^{i_1} \cdots x_{j_1-1}^{i_1} x_{j_1+1}^{i_1} \cdots \bar{x}_{j_2}^{i_1} \cdots x_{k_{i_1}}^{i_1}$  can be obtained from  $p_{i_1}$  by omitting its  $j_1$ -th literal and negating its  $j_2$ -th literal and  $j_1 < j_2$ .

**LOF and LOF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})$ . After the first LOF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then omitted from  $p_{i_2}$ . We use  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1}) \otimes LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$  and  $k_{i_2} > 1$ . Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2, \hat{j}_2} + \cdots + p_m \quad (31)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . Furthermore, we do not consider the situation when  $p_{i_1}$  contains just two literals because the net effect of omitting two literals from  $p_{i_1}$  will be equivalent to a term omission fault, which is a single term fault. Without loss of generality, we assume  $j_1 < j_2$  and  $k_{i_1} > 2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} + \cdots + p_m \quad (32)$$

**LOF and LIF** Let  $x_{l_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})$ . After the first LOF is made on  $S$ , the literal  $x_{l_2}$  is then inserted into  $p_{i_2}$ . We use  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1}) \otimes LIF(p_{i_2} \rightarrow p_{i_2, x_{l_2}})$  to denote the corresponding faulty implementation.

We do not consider the situation when LOF and LIF are committed at the same term (that is  $i_1 = i_2$ ) because of the following three reasons. First, when the omitted literal  $x_{j_1}^{i_1}$  is inserted back into the term (that is,  $x_{l_2} = x_{j_1}^{i_1}$ ), the implementation is then equivalent to the original expression  $S$ . Second, when the negation of the omitted literal  $x_{j_1}^{i_1}$  is inserted into the term (that is,  $x_{l_2} = \bar{x}_{j_1}^{i_1}$ ), the implementation is then equivalent to a single LNF with respect to the original expression  $S$ . Third, when the two literals  $x_{j_1}^{i_1}$  and  $x_{l_2}$  are different, the implementation is then equivalent to a single LRF with respect to the original expression  $S$  because the net result of omitting the literal  $x_{j_1}^{i_1}$  from a term and then inserting  $x_{l_2}$  into that term is the same as replacing  $x_{j_1}^{i_1}$  by  $x_{l_2}$ .

Hence, we only consider the situation when LOF and LIF are committed at two different terms, say, the  $i_1$ -th and  $i_2$ -th terms (that is  $i_1 \neq i_2$ ). It should be noted that  $k_{i_1} > 1$ . Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2} x_{l_2} + \cdots + p_m \quad (33)$$

**LOF and LRF** Let  $x_{j_2}^{i_2}$  and  $x_{l_2}$  be a literal and a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1})$ , respectively. After the first LOF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  from  $p_{i_2}$  is then replaced by  $x_{l_2}$ . We use  $I_{LOF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1}) \otimes LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_1} > 1$ .

Without loss of generality, we assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} + \cdots + p_{i_2, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (34)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . Since  $x_{j_1}^{i_1}$  is omitted,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are different (that is  $j_1 \neq j_2$ ). Without loss of generality, we assume  $j_1 < j_2$ . We do not consider the situation when  $x_{j_1}^{i_1}$  is first omitted, and then another literal  $x_{j_2}^{i_1}$  is replaced by

the omitted literal  $x_{j_1}^{i_1}$  (that is,  $x_{l_2}$  is the same as  $x_{j_1}^{i_1}$ ) because the implementation is then equivalent to a single LOF with respect to the original expression  $S$  with  $x_{j_2}^{i_1}$  being omitted.

As a result, we have the following two subcases:

- (a) The literal  $x_{j_2}^{i_1}$  is replaced by the negation of  $x_{j_1}^{i_1}$  (that is  $x_{l_2} = \bar{x}_{j_1}^{i_1}$ ). The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, \hat{j}_2} + \cdots + p_m \quad (35)$$

- (b) The literal  $x_{j_2}^{i_1}$  is replaced by a literal different from  $x_{j_1}^{i_1}$  (that is,  $x_{l_2} \neq x_{j_1}^{i_1}$  and  $x_{l_2} \neq \bar{x}_{j_1}^{i_1}$ ). The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (36)$$

### 4.3 LIF first, then Other Faults

Let  $S = p_1 + \cdots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{l_1}$  be a missing literal of the  $i_1$ -th term,  $p_{i_1}$ , of  $S$ . Suppose the literal  $x_{l_1}$  is inserted into  $p_{i_1}$ . The corresponding faulty implementation, denoted as  $I_{LIF(p_{i_1} \rightarrow p_{i_1} x_{l_1})}$ , is then equivalent to  $p_1 + \cdots + p_{i_1} x_{l_1} + \cdots + p_m$ .

**LIF and LNF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LIF(p_{i_1} \rightarrow p_{i_1} x_{l_1})}$ . After the first LIF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then negated. We use  $I_{LIF(p_{i_1} \rightarrow p_{i_1} x_{l_1}) \otimes LNF(p_{i_2} \rightarrow p_{i_2, \bar{j}_2})}$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_2} > 1$ .

Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1} x_{l_1} + \cdots + p_{i_2, \bar{j}_2} + \cdots + p_m \quad (37)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider  $x_{l_1}$  being negated because the implementation will then be equivalent to a single LIF with respect to the original specification  $S$  with  $\bar{x}_{l_1}$  being inserted into  $p_{i_1}$  of  $S$ . Therefore,  $x_{j_2}^{i_1}$  is a literal in  $p_{i_1}$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_2} x_{l_1} + \cdots + p_m \quad (38)$$

**LIF and LOF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LIF}(p_{i_1} \rightarrow p_{i_1} x_{l_1})$ . After the first LIF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then omitted from  $p_{i_2}$ . We use  $I_{LIF}(p_{i_1} \rightarrow p_{i_1} x_{l_1}) \otimes LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to denote the corresponding faulty implementation. We do not consider the situation when the LIF and LOF are committed at the same term (that is,  $i_1 = i_2$ ) because of the following two reasons. First, when the inserted literal  $x_{l_1}$  is omitted from  $p_{i_1}$ , the implementation then is equivalent to original expression  $S$ . Second, when any literal in  $p_{i_1}$  is omitted, the implementation is then equivalent to a single LRF with respect to the original expression  $S$  because the net result of inserting a literal  $x_{l_1}$  into a term and then omitting another literal  $x_{j_2}^{i_1}$  in  $p_{i_1}$  is the same as replacing  $x_{j_2}^{i_1}$  by  $x_{l_1}$ .

Hence, we only consider the situation when LOF and LIF are committed at two different terms (that is  $i_1 \neq i_2$ ). It should be noted that  $k_{i_2} > 1$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1} x_{l_1} + \cdots + p_{i_2, \hat{j}_2} + \cdots + p_m \quad (39)$$

**LIF and LIF** Let  $x_{l_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LIF}(p_{i_1} \rightarrow p_{i_1} x_{l_1})$ . After the first LIF is made on  $S$ , the literal  $x_{l_2}$  is then inserted into  $p_{i_2}$ . We use  $I_{LIF}(p_{i_1} \rightarrow p_{i_1} x_{l_1}) \otimes LIF(p_{i_2} \rightarrow p_{i_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can

assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}x_{l_1} + \cdots + p_{i_2}x_{l_2} + \cdots + p_m \quad (40)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . Since  $x_{l_2}$  is a missing literal of  $p_{i_1}x_{l_1}$ , the  $i_1$ -th term of  $I_{LIF}(p_{i_1} \rightarrow p_{i_1}x_{l_1})$ , the two literals  $x_{l_1}$  and  $x_{l_2}$  are different. It should be noted that  $x_{l_2}$  is also a missing literal of  $p_{i_1}$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}x_{l_1}x_{l_2} + \cdots + p_m \quad (41)$$

**LIF and LRF** Let  $x_{j_2}^{i_2}$  and  $x_{l_2}$  be a literal and a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LIF}(p_{i_1} \rightarrow p_{i_1}x_{l_1})$ , respectively. After the first LIF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  from  $p_{i_2}$  is then replaced by the literal  $x_{l_2}$ . We use  $I_{LIF}(p_{i_1} \rightarrow p_{i_1}x_{l_1}) \otimes LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2}x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}x_{l_1} + \cdots + p_{i_2, \hat{j}_2}x_{l_2} + \cdots + p_m \quad (42)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation where the two literals  $x_{j_2}^{i_1}$  and  $x_{l_1}$  are exactly the same (that is  $x_{j_2}^{i_1} = x_{l_1}$ ). It is because when  $x_{l_1}$  is inserted into  $p_i$  of  $S$  and then  $x_{l_1}$  is replaced by another literal  $x_{l_2}$ , the implementation is then equivalent to a single LIF with respect to  $S$  with  $x_{l_2}$  being inserted into  $p_{i_1}$ . It should be noted that  $x_{l_2}$  is also a missing literal of  $p_{i_1}$  because it is a missing literal of  $p_{i_1}x_{l_1}$ , the  $i_1$ -th term of  $I_{LIF}(p_{i_1} \rightarrow p_{i_1}x_{l_1})$ . As a result, both  $x_{l_1}$  and  $x_{l_2}$  are two different missing literals of  $p_{i_1}$  and the implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_2}x_{l_1}x_{l_2} + \cdots + p_m \quad (43)$$

## 4.4 LRF first, then Other Faults

Let  $S = p_1 + \dots + p_m$  be a Boolean specification in irredundant disjunctive normal form. Let  $x_{j_1}^{i_1}$  be a literal of the  $i_1$ -th term,  $p_{i_1}$ , of  $S$  and  $x_{l_1}$  be a missing literal of  $p_{i_1}$ . Suppose the literal  $x_{j_1}^{i_1}$  is replaced by the literal  $x_{l_1}$ . The corresponding faulty implementation, denoted as  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ , is then equivalent to  $p_1 + \dots + p_{i_1, \hat{j}_1} x_{l_1} + \dots + p_m$ .

**LRF and LNF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ . After the first LRF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then negated. We use  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1}) \otimes LNF(p_{i_2} \rightarrow p_{i_2, \bar{j}_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_2} > 1$ .

Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \hat{j}_1} x_{l_1} + \dots + p_{i_2, \bar{j}_2} + \dots + p_m \quad (44)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation when the two literals  $x_{j_2}^{i_2}$  and  $x_{l_1}$  are exactly the same (that is  $x_{j_2}^{i_2} = x_{l_1}$ ). It is because when  $x_{j_1}^{i_1}$  is replaced by  $x_{l_1}$  and then  $x_{l_1}$  is negated, the implementation is then equivalent to a single LRF with respect to the original expression  $S$  with  $x_{j_1}^{i_1}$  being replaced by  $\bar{x}_{l_1}$ . Since  $x_{j_1}^{i_1}$  does not exist in  $p_{i_1, \hat{j}_1} x_{l_1}$  in  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$  and  $x_{j_2}^{i_2}$  is different from  $x_{l_1}$ ,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_2}$  are two different literals of  $p_{i_1}$  (that is  $j_1 \neq j_2$ ). Without loss of generality, we can assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1, \hat{j}_1, \bar{j}_2} x_{l_1} + \dots + p_m \quad (45)$$

**LRF and LOF** Let  $x_{j_2}^{i_2}$  be a literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ . After the first LRF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  is then omitted from  $p_{i_2}$ . We use  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1}) \otimes LOF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2})$  to

denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . It should be noted that  $k_{i_2} > 1$ .

Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} x_{l_1} + \cdots + p_{i_2, \hat{j}_2} + \cdots + p_m \quad (46)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation

when the literal  $x_{l_1}$  is omitted because the implementation is then equivalent to a single LOF with respect to the original expression  $S$  with  $x_{j_1}^{i_1}$  being omitted. Hence,  $x_{j_2}^{i_1}$  is a literal in  $p_{i_1, \hat{j}_1}$  and the two literals  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are different (that is  $j_1 \neq j_2$ ). Without loss of generality, we can assume  $j_1 < j_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_1} + \cdots + p_m \quad (47)$$

**LRF and LIF** Let  $x_{l_2}$  be a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ . After the first LRF is made on  $S$ , the literal  $x_{l_2}$  is then inserted into  $p_{i_2}$ . We use  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1}) \otimes LIF(p_{i_2} \rightarrow p_{i_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} x_{l_1} + \cdots + p_{i_2} x_{l_2} + \cdots + p_m \quad (48)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation

when  $x_{j_1}^{i_1}$  is inserted back to  $p_{i_1, \hat{j}_1} x_{l_1}$ , the  $i_1$ -th term of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$  (that is  $x_{l_2} = x_{j_1}^{i_1}$ ),

because the implementation is then equivalent to a single LIF with respect to the original expression  $S$  with  $x_{l_1}$  being inserted into  $p_{i_1}$ . As a result, we have the following two subcases:

- (a) The negation of the literal  $x_{j_1}^{i_1}$  is inserted (that is  $x_{l_2} = \bar{x}_{j_1}^{i_1}$ ). The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1} x_{l_1} + \cdots + p_m \quad (49)$$

- (b) The two literals  $x_{l_2}$  and  $x_{j_1}^{i_1}$  are different (that is,  $x_{l_2} \neq x_{j_1}^{i_1}$  and  $x_{l_2} \neq \bar{x}_{j_1}^{i_1}$ ). Thus,  $x_{l_2}$  is a missing literal of  $p_{i_1}$  because it is a missing literal of  $p_{i_1, \hat{j}_1} x_{l_1}$ , the  $i_1$ -th term of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ . As a result, both  $x_{l_1}$  and  $x_{l_2}$  are two different missing literals of  $p_{i_1}$  and the implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} x_{l_1} x_{l_2} + \cdots + p_m \quad (50)$$

**LRF and LRF** Let  $x_{j_2}^{i_2}$  and  $x_{l_2}$  be a literal and a missing literal of the  $i_2$ -th term,  $p_{i_2}$ , of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ , respectively. After the first LRF is made on  $S$ , the literal  $x_{j_2}^{i_2}$  from  $p_{i_2}$  is then replaced by  $x_{l_2}$ . We use  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1}) \otimes LRF(p_{i_2} \rightarrow p_{i_2, \hat{j}_2} x_{l_2})$  to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The  $i_1$ -th and  $i_2$ -th terms are different, that is  $i_1 \neq i_2$ . Without loss of generality, we can assume  $i_1 < i_2$ . The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1} x_{l_1} + \cdots + p_{i_2, \hat{j}_2} x_{l_2} + \cdots + p_m \quad (51)$$

Case 2. The  $i_1$ -th and  $i_2$ -th terms are the same, that is  $i_1 = i_2$ . We do not consider the situation when  $x_{l_1}$  is replaced by  $x_{l_2}$  because of the following three reasons. During the discussion, please noted that, in the first LRF,  $x_{j_1}^{i_1}$  is replaced by  $x_{l_1}$ . First, when  $x_{l_1}$  is now replaced by  $x_{j_1}^{i_1}$ , the implementation is then equivalent to the original expression  $S$ . Second, when

$x_{l_1}$  is now replaced by the negation of  $x_{j_1}^{i_1}$ , the implementation is then equivalent to a single LNF with respect to the original expression  $S$  with  $x_{j_1}^{i_1}$  being negated. Third, when  $x_{l_1}$  is now replaced by  $x_{l_2}$  which is different from  $x_{j_1}^{i_1}$  (that is,  $x_{l_2} \neq x_{j_1}^{i_1}$  and  $x_{l_2} \neq \bar{x}_{j_1}^{i_1}$ ), the implementation is then equivalent to a single LRF with respect to the original expression  $S$  with  $x_{j_1}^{i_1}$  being replaced by  $x_{l_2}$ .

As a result, we only need to consider literals in  $p_{i_1, \hat{j}_1}$  being replaced by  $x_{l_2}$ . Therefore,  $x_{j_1}^{i_1}$  and  $x_{j_2}^{i_1}$  are two different literals of  $p_{i_1}$ . Without loss of generality, we can assume  $j_1 < j_2$ . Furthermore, we do not consider the situation when  $x_{j_2}^{i_1}$  is replaced by  $x_{j_1}^{i_1}$  because the implementation is then equivalent to a single LRF with respect to the original expression  $S$  with  $x_{j_2}^{i_1}$  being replaced by  $x_{l_1}$ . Hence, we have the following two subcases:

- (a) The literal  $x_{j_2}^{i_1}$  is replaced by  $\bar{x}_{j_1}^{i_1}$ , the implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \bar{j}_1, \hat{j}_2} x_{l_1} + \cdots + p_m \quad (52)$$

- (b) The literal  $x_{j_2}^{i_1}$  is replaced by a literal different from  $x_{j_1}^{i_1}$  (that is,  $x_{l_2} \neq x_{j_1}^{i_1}$  and  $x_{l_2} \neq \bar{x}_{j_1}^{i_1}$ ). Since  $x_{l_2}$  is also a missing literal of  $p_{i_1, \hat{j}_1} x_{l_1}$ , the  $i_1$ -th term of  $I_{LRF}(p_{i_1} \rightarrow p_{i_1, \hat{j}_1} x_{l_1})$ , and it is different from  $x_{j_1}^{i_1}$ ,  $x_{l_2}$  is a missing literal of  $p_{i_1}$ . As a result, both  $x_{l_1}$  and  $x_{l_2}$  are two different missing literals of  $p_{i_1}$  and the implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1, \hat{j}_1, \hat{j}_2} x_{l_1} x_{l_2} + \cdots + p_m \quad (53)$$

In summary, there are all together 33 double-fault expressions among the 16 double fault classes with ordering considered in this report. In the next section, we show that these 33 double-fault expressions can be reduced to 19 double fault expressions discussed in Section 3.

Table 2: Comparison of faulty implementations of LNF and other faults

Fault type	Double fault without ordering (Expression number)	Double fault with ordering - LNF first (Expression number)
LNF and LNF	1	21
	2	22
LNF and LOF	3	23
	4	24
LNF and LIF	5	25
	6	26
LNF and LRF	7	27
	8	28

## 5 Relation between Double Faults with and without Ordering

In this section, we analyse the relation of double faults with and without ordering. We compare the possible faulty expressions of double faults without ordering with respect to those with ordering in the same fault category related to LNF, LOF, LIF and LRF. Table 2 (respectively, 3, 4 and 5) summarizes the situations of double faults with LNF (respectively LOF, LIF, and LRF) and other faults. Each row in these tables shows those faulty expressions of a particular type of double fault without ordering and their counterparts in double faults with ordering.

Let us consider Table 2. In the first row, there are two subcases for LNF and LNF without ordering which are given by Expressions (1) and (2) in Section 3. For the first subcase, Expression (1) is equivalent to Expression (21) which corresponds to the first subcase of LNF and LNF with ordering as discussed in Section 4. Similarly, other rows in Table 2 show the equivalent faulty expressions of double faults of LNF and other faults with and without ordering.

For the rows in Table 3, there are three different situations. First, some of them can be interpreted in a similar manner as those in Table 2. For example, for LOF and LOF in Table 3, Expression (9) is equivalent to Expression (31). Second, for some rows, the faulty expressions of double faults with

Table 3: Comparison of faulty implementations of LOF and other faults

Fault type	Double fault without ordering (Expression number)	Double fault with ordering - LOF first (Expression number)
LOF and LNF	3	29
	4	30
LOF and LOF	9	31
	10	32
LOF and LIF	11	33
LOF and LRF	12	34
	-	35 (4)
	13	36

ordering have their counterparts in double faults without ordering in the same double fault class. For example, let us consider the first row in Table 3, there are two subcases for LOF and LNF. In the first subcase, Expression (3) is derived from LNF and LOF without ordering with LNF being committed at the  $i_1$ -th term and LOF committed at the  $i_2$ -th term. However, Expression (29) is derived from LOF and LNF with ordering with LOF being committed at  $i_1$ -th term and LNF being committed at the  $i_2$ -th term. By interchanging the position of these terms, Expressions (3) and (29) are equivalent. Hence, they are considered as counterparts of each other. Similar situations occurred at Expressions (4) and (30) in second subcase of LOF and LNF, and other double fault classes in Tables 3, 4 and 5. In later discussions, we assume that the readers can make the adjustment accordingly. Third, for some rows, the faulty expressions of double faults with ordering do not have their counterparts in double faults without ordering in the same double fault class, but they are equivalent to other faulty expressions of double faults without ordering in a different double fault class. For example, let us consider the second subcase of LOF and LRF of double fault with ordering, the faulty expression is given by Expression (35). It does not have its counterpart in LOF and LRF without ordering. However, it is equivalent to Expression (4) in the double fault LNF and LOF without ordering in Section 3.

For the rows in Table 4, there are two different situations. First, some of them can be interpreted in

Table 4: Comparison of faulty implementations of LIF and other faults

Fault type	Double fault without ordering (Expression number)	Double fault with ordering - LIF first (Expression number)
LIF and LNF	5	37
	6	38
LIF and LOF	11	39
LIF and LIF	14	40
	15	41
LIF and LRF	16	42
	17	43

a similar manner as those in Table 2. For example, for LIF and LIF in Table 4, Expression (14) is equivalent to Expression (40). Second, for some rows, the faulty expressions of double faults with ordering have their counterparts in double faults without ordering in the same double fault class. For example, Expressions (5) and (37) are considered to be counterparts of each other as discussed previously for Expressions (3) and (29) in Table 3.

For the rows in Table 5, there are four different situations. First, some of them can be interpreted in a similar manner as those in Table 2. For example, for LRF and LRF in Table 5, Expression (18) is equivalent to Expression (51). Second, for some rows, the faulty expressions of double faults with ordering have their counterparts in double faults without ordering in the same double fault class. For example, Expressions (7) and (44) are considered to be counterparts of each other. Third, for some rows, the faulty expressions of double faults with ordering do not have their counterparts in double faults without ordering in the same double fault class, but they are equivalent to other faulty expressions of double faults without ordering in a different double fault class. For example, for the second subcase of LRF and LRF of double fault with ordering, Expression (52) does not have its counterpart in LRF and LRF without ordering. However, it is equivalent to Expression (8) in the double fault LNF and LRF without ordering in Section 3. Fourth, for some rows, the faulty expressions of double faults without ordering do not have their counterparts in double faults with ordering in the same double fault class, but they are equivalent to other faulty expressions of double

Table 5: Comparison of faulty implementations of LRF and other faults

Fault type	Double fault without ordering (Expression number)	Double fault with ordering - LRF first (Expression number)
LRF and LNF	7	44
	8	45
LRF and LOF	12	46
	13	47
LRF and LIF	16	48
	-	49 (6)
	17	50
LRF and LRF	18	51
	-	52 (8)
	19	53
	20 (same as 13 as discussed in Section 3)	-

faults with ordering in a different double fault class. For example, for the third subcase of LRF and LRF of double fault without ordering, Expression (20) does not have its counterpart in LRF and LRF with ordering. But it is equivalent to Expression (47) in the double fault LOF and LRF without ordering.

In summary, all expressions of double faults with ordering have their counterparts in double faults without ordering. Therefore, for the four single fault classes studied in this report, there are 19 different double-fault expressions and all of them fall into the category of double fault without ordering. Table 6 summarizes all these expressions.

## 6 Conclusion

In this report, we study the relationship of double faults with and without ordering based on four single fault types related to literals in Boolean expressions. The four single fault classes considered in this report, they are literal negation fault (LNF), literal omission fault (LOF), literal insertion fault

Table 6: Double fault and double-fault expression ( $S = p_1 + \dots + p_m$ )

Fault Class	Double-fault Expression
LNF $\times$ LNF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \bar{j}_2} + \dots + p_m$ (1)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1, \bar{j}_2} + \dots + p_m$ (2)
LNF $\times$ LOF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \hat{j}_2} + \dots + p_m$ (3)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1, \hat{j}_2} + \dots + p_m$ (4)
LNF $\times$ LIF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, x_{l_2}} + \dots + p_m$ (5)
	Case 2: $p_1 + \dots + p_{i_1, \bar{j}_1, x_{l_2}} + \dots + p_m$ (6)
LNF $\times$ LRF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, \hat{j}_2, x_{l_2}} + \dots + p_m$ (7)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1, \hat{j}_2, x_{l_2}} + \dots + p_m$ (8)
LOF $\times$ LOF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1} + \dots + p_{i_2, \hat{j}_2} + \dots + p_m$ (9)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1, \hat{j}_2} + \dots + p_m$ (10)
LOF $\times$ LIF	( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \bar{j}_1} + \dots + p_{i_2, x_{l_2}} + \dots + p_m$ (11)
LOF $\times$ LRF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1} + \dots + p_{i_2, \hat{j}_2, x_{l_2}} + \dots + p_m$ (12)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1, \hat{j}_2, x_{l_2}} + \dots + p_m$ (13)
LIF $\times$ LIF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, x_{l_1}} + \dots + p_{i_2, x_{l_2}} + \dots + p_m$ (14)
	Case 2: $p_1 + \dots + p_{i_1, x_{l_1}, x_{l_2}} + \dots + p_m$ (15)
LIF $\times$ LRF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, x_{l_1}} + \dots + p_{i_2, \hat{j}_2, x_{l_2}} + \dots + p_m$ (16)
	Case 2: $p_1 + \dots + p_{i_1, \hat{j}_2, x_{l_1}, x_{l_2}} + \dots + p_m$ (17)
LRF $\times$ LRF	Case 1 ( $i_1 < i_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1, x_{l_1}} + \dots + p_{i_2, \hat{j}_2, x_{l_2}} + \dots + p_m$ (18)
	Case 2 ( $j_1 < j_2$ ): $p_1 + \dots + p_{i_1, \hat{j}_1, \hat{j}_2, x_{l_1}, x_{l_2}} + \dots + p_m$ (19)

(LIF) and literal reference fault (LRF). A double fault is defined as the occurrence of two single faults. Moreover, we constrained that faulty expression with double fault is neither equivalent to its original expression nor faulty expression with single faults. Since different order of occurrences of two single faults in a double fault may result in different faulty implementations, we further study the double faults from two categories, namely double faults with and without ordering.

For the four single fault classes related to literal, there are 10 types of double fault without ordering resulting in 20 possible faulty expressions. Since two faulty expressions in double fault without ordering are equivalent to each other, there are actually 19 distinct non-equivalent faulty expressions in double fault without ordering. On the other hand, there are 16 types of double fault with ordering resulting in 33 possible faulty expressions. Moreover, we show in this report that all the faulty expressions of double fault with ordering are equivalent to those 19 distinct non-equivalent faulty expressions of double fault without ordering.

The study of relationship of double faults not only helps us to better understand how these double faults occur in Boolean expression and how they interact with each other, but also enumerate all possible faulty expressions related to double literal faults. Based on the 19 faulty expressions, detection conditions can be derived to design test cases aiming at the detection of the corresponding double faults.

## References

- [1] T. Y. Chen and M. F. Lau. Test case selection strategies based on boolean specifications. *Software Testing, Verification and Reliability*, 11(3):165 – 180, 2001.
- [2] K. S. How Tai Wah. Fault coupling in finite bijective functions. *Software Testing, Verification and Reliability*, 5(1):3–47, 1995.

- [3] K. S. How Tai Wah. A theoretical study of fault coupling. *Software Testing, Verification and Reliability*, 10(1):3–45, 2000.
- [4] D. R. Kuhn. Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 8(4):411–424, 1999.
- [5] M. F. Lau and Y. Liu. Classification and relationship of double faults related to terms in Boolean expressions. Technical Report SUTICT-TR2006.01, Swinburne University of Technology, 2006.
- [6] M. F. Lau, Y. Liu, and Y. T. Yu. On the detection conditions of double faults related to terms in boolean expressions. To appear in *Proceedings of the 30th International Computer Software and Applications Conference (COMPSAC 2006)*, 2006.
- [7] M. F. Lau and Y. T. Yu. On the relationships of faults for Boolean specification based testing. In *Proceedings of Australian Software Engineering Conference (ASWEC 2001)*, pages 21 – 28, 2001.
- [8] M. F. Lau and Y. T. Yu. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 14(3):247 – 276, 2005.
- [9] B. Marick. Two experiments in software testing. Technical Report Technical Report UIUCDCS-R-90-1644, University of Illinois at Urbana-Champaign, 1990.
- [10] L. J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844 – 857, 1990.
- [11] A. J. Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, 1992.
- [12] T. Tsuchiya and T. Kikuno. On fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 11(1):58 – 62, 2002.

- [13] D. Wallace and D. Kuhn. Failure modes in medical device software: an analysis of 15 years of recall data. *International Journal of Reliability, Quality, and Safety Engineering*, 8(4), 2001.
- [14] E. Weyuker, T. Goradia, and A. Singh. Automatically generating test data from a Boolean specification. *IEEE Transactions on Software Engineering*, 20(5):353–363, 1994.