

Faculty of Information and Communication Technologies

Classification and Relationship of Double Faults Related to Terms in Boolean Expressions

Technical Report: SUTICT-TR2006.01

Man Fai Lau and Ying Liu

1st July 2006



SWIN
BUR
NE

SWINBURNE UNIVERSITY
OF TECHNOLOGY

This page is intentionally left blank.

Classification and Relationship of Double Faults Related to Terms in Boolean Expressions

Table of contents

1	Introduction	1
2	Preliminary	4
2.1	Notation	4
2.2	Fault Classes	5
3	Double Faults without Ordering	6
3.1	ENF with Other Faults	8
3.2	TNF with Other Faults	11
3.3	TOF with Other Faults	13
3.4	DORF with Other Faults	14
3.5	CORF with Other Faults	15
4	Double Faults with Ordering	16
4.1	ENF First, then Other Faults	17
4.2	TNF First, then Other Faults	20
4.3	TOF First, then Other Faults	22
4.4	DORF First, then Other Faults	24
4.5	CORF First, then Other Faults	27
5	Relation between Double Faults with and without Ordering	31
6	Conclusion and Future Work	35

This page is intentionally left blank.

Abstract

Fault-based testing strategies have been proposed to detect hypothesized faults in a program for many years. Most fault-based testing research assumes that only one hypothesized faults may occur in a program. However, empirical investigations show that multiple faults occur more frequently in practice. Hence, it is important to study the behaviour of multiple faults occurred in a program. Most research on multiple faults are specialized in double fault, which is defined as the occurrence of two single faults in a program. Moreover, in these studies, the researchers assumed that the two single faults occurred independent to each other. It is also possible that two single faults may occur one after another, resulting in the first fault actually affecting the occurrence of the second fault. Therefore, we need to further study these two situations of double fault being occurred to see whether there are differences between these two groups of double faults. During the study, the relationships between these two groups of double faults are identified and reported. The results will help us to better understand on the occurrence of two single faults in a double fault and how they interact with each other.

1 Introduction

Fault-based testing techniques have been proposed which aim at detecting hypothesized faults [1, 4, 5, 7, 9, 11]. More precisely, if any hypothesized fault exists in the program under test, test cases generated by fault-based testing techniques can detect it. Most studies related to fault-based testing techniques assume only single occurrence of any hypothesized fault [7]. However, it is quite common that, during software development, programmers make mistakes which in turn inject more than one fault in program. Moreover, empirical investigations show that programs with multiple occurrences of faults (or, simply programs with multiple faults) happen more often than those with one occurrence of fault [6, 10]. Hence, studies related to multiple faults may give further insights on errors committed by programmers.

Previous studies on multiple faults focus mainly on fault coupling with double faults [2, 3, 8]. A program is said to have a *double fault* if there are two occurrences of single faults. Offutt [8] has performed an empirical study on fault coupling via mutation analysis. A *mutant* is a program that differs from the original program by small syntactic changes. A mutant is a *1-order* (respectively, *2-order*) mutant if it differs from the original program by 1 syntactic change (respectively, 2 syntactic changes). Three programs whose sizes ranges from 16 to 28 lines of code are studied. Test sets that can kill all 1-order mutants are generated and used to kill 2-order mutants. As mentioned by Offutt, the experiment performed is a study of the *mutation* coupling effect to be precise. It is found that test sets so generated can kill approximately 99.9% of 2-order mutants. He concluded that the effect of two faults coupled together rarely occurred.

On the other hand, How Tai Wah [2, 3] studies fault coupling from a theoretical perspective. Unlike Offutt, he does not have particular types of faults in mind. In his study, a function is considered to be faulty if it produces an incorrect output. He models a program as a composition of finite functions. A program with single fault is modelled as a composition of functions in which exactly one function is faulty, while a program with double faults is a composition of functions in which exactly two of them are faulty. In other words, a program with double fault is a combination of two individual faulty functions. He investigates the problem of detecting double faults by test sets that can detect the two faults in isolation. All such test sets considered in the study have at most two elements. It should be noted that if such a test set has 1 element, the only test case in the set can detect both faults in isolation. Test sets that can detect individual faults of a double fault are defined as *proper test sets*. Among proper test sets, those that cannot detect the double fault are defined as *coupled test sets*. He calculates the *coupling ratio*, defined as the ratio of the number of coupled test sets to that of proper test sets. The coupling ratio is approximately $1/|D|$ and $1/|D|^2$ for test sets of sizes 1 and 2, respectively, where $|D|$ is the size of the input domain D . As $|D|$ is usually very large, the ratio is very small. He concludes that fault coupling rarely occurs.

Our study of double faults is different from previous studies in two ways. First, we model faults from software specifications. This is different from Offutt's approach, which is code based, and How

Tai Wah's approach, which is from functional viewpoint. Since programs are developed based on specifications, modelling faults from specifications provides better insights on how faults are being injected in programs. Second, we study the effect of different order of occurrences of two individual faults in a double fault because different orderings of these two faults may result in different double faults. Hence, it is important to study the relationship between these double faults for better insights on how these faults interact with each other and, hence, for better double fault detection. This is different from previous studies because they have not considered this.

In this report, we study the relationships of double faults based on program specifications that can be expressed in or modelled as Boolean expressions. Based on various types of single faults that may occur in Boolean expressions in research literature [1, 5], we define different types of double faults that may occur in Boolean expressions and model their corresponding faulty implementations.

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 [8] or the functions that model a program [2, 3]. In other words, they have not explicitly considered how a single fault can affect another single fault. In fact, it is possible that, for a double fault, the occurrence of a single fault may affect the other single fault. In essence, the former case refers to the situation where single faults are independent of each other whereas the latter case refers to the situation where the ordering of the two single faults may actually make a difference. Hence, it is interesting to study the differences between these two cases. Furthermore, since two individual faults committed in different orderings may result in two different faulty implementations, we also study the relationship of these faulty implementations. After the study, we will have better understanding on how double faults occur within Boolean expressions and how they interact with each other based on faulty implementations.

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 ‘ \cdot ’, ‘ $+$ ’ and ‘ $-$ ’ to represent Boolean operators, AND, OR and NOT, respectively. Usually, ‘ \cdot ’ 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 disjunctive normal form

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

where m is the number of terms, $p_i = x_1^i \cdots x_{k_i}^i$ 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* 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.

Let S be a Boolean expression having n variables, the input domain is the n -dimensional Boolean space \mathbb{B}^n . True points are those that cause 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 that makes (1) p_i evaluate to 1; and (2) all other terms evaluate to 0. The set of all unique true points of p_i in S is denoted by $UTP_i(S)$. The set of all unique true points of S is denoted by $UTP(S)$ and $UTP(S) = \bigcup_i UTP_i(S)$.

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 that makes (1) x_j^i evaluate 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 Fault Classes

In this report, we only consider five fault classes related to terms in a Boolean expression. Let S be a Boolean expression in irredundant disjunctive normal form. Suppose a fault F changes a subexpression E into a subexpression E' . The resulting faulty implementation, referred to as *single-fault expression*, is denoted by $I_{F(E \rightarrow E')}$. The single-fault expression differs from the original expression by one syntactic change and is not equivalent to the original expression. The following five fault classes related to terms in Boolean expressions are studied in this report:

1. *Expression Negation Fault* (ENF): The entire Boolean expression or its subexpression is wrongly implemented as its negation. Suppose that the entire Boolean expression is wrongly negated. The implementation, denoted by $I_{ENF(S \rightarrow \bar{S})}$, is then equivalent to \bar{S} (or, simply the implementation is then equivalent to $I_{ENF(S \rightarrow \bar{S})} = \bar{S}$). If the subexpression $p_i + \dots + p_h$ ($1 \leq i < h \leq m$) is wrongly negated, the implementation is then equivalent to $I_{ENF(p_i + \dots + p_h \rightarrow \overline{p_i + \dots + p_h})} = p_1 + \dots + p_{i-1} + \overline{p_i + \dots + p_h} + p_{h+1} + \dots + p_m$. It should be noted that the former case is a special instance of the latter case when $i = 1$ and $h = m$. For example, the Boolean expression $ab + cd + ef$ may be wrongly implemented as $\overline{ab + cd + ef}$ or $ab + \overline{cd + ef}$.
2. *Term Negation Fault* (TNF): A particular term in the Boolean expression is wrongly implemented as its negation. If the i -th term p_i of S ($1 \leq i \leq m$ and $m > 1$) is negated, the implementation is then equivalent to $I_{TNF(p_i \rightarrow \bar{p}_i)} = p_1 + \dots + \bar{p}_i + \dots + p_m$. For example, the Boolean expression $ab + cd + ef$ may be implemented as $ab + cd + \bar{e}f$. As documented in [5], when S contains just one term (that is $m = 1$), the negation fault is considered as a ENF.
3. *Term Omission Fault* (TOF): A particular term in the Boolean expression is omitted in its implementation. If the i -th term p_i of S ($1 \leq i \leq m$ and $m > 1$) is omitted, the implementation

is then equivalent to $I_{TOF}(p_i \rightarrow) = p_1 + \dots + p_{i-1} + p_{i+1} + \dots + p_m$. For example, the Boolean expression $ab + cd + ef$ may be implemented as $ab + cd$.

4. *Disjunctive Operator Reference Fault (DORF)*: The Boolean disjunctive operator '+' is wrongly implemented as the conjunctive operator '·'. If the + operator between the i -th and $i + 1$ -th terms in S is implemented as the · operator where $1 \leq i < m$, the implementation is then equivalent to $I_{DORF}(p_i + p_{i+1} \rightarrow p_i p_{i+1}) = p_1 + \dots + p_i p_{i+1} + \dots + p_m$. For example, the Boolean expression $ab + cd + ef$ may be implemented as $ab + cdef$.
5. *Conjunctive Operator Reference Fault (CORF)*: The Boolean conjunctive operator '·' is wrongly implemented as the disjunctive operator '+'. If the · operator between the j -th and $j + 1$ -th literals in the i -th term of S is implemented as the + operator where $1 \leq j < k_i$ and $1 \leq i \leq m$, the implementation is then equivalent to $I_{CORF}(p_i \rightarrow p_{i,1,j} + p_{i,j+1,k_i}) = p_1 + \dots + p_{i-1} + p_{i,1,j} + p_{i,j+1,k_i} + p_{i+1} + \dots + p_m$, where $p_{i,1,j} = x_1^i \dots x_j^i$, $p_{i,j+1,k_i} = x_{j+1}^i \dots x_{k_i}^i$ and $p_i = p_{i,1,j} \cdot p_{i,j+1,k_i}$. For example, the Boolean expression $ab + cd + ef$ may be implemented as $ab + c + d + ef$.

3 Double Faults without Ordering

Multiple occurrences of any fault classes may result in a faulty expression which differs from the original expression by several syntactic changes. Such an expression is considered to contain multiple faults. In this report, we focus on *double faults* which are defined as two occurrences of single faults.

It should be noted that when two single faults are committed based on a given Boolean expression, the resulting expression may be equivalent to the original expression or to an expression which is different from the original expression by a syntactic change. For example, if the first term of the expression $p_1 + p_2 + \dots + p_m$ is negated twice, the resulting expression may be given by $\overline{\overline{p_1}} + p_2 + \dots + p_m$, which is equivalent to the original expression. On the other hand, if the first term

of the expression $p_1 + p_2 + \dots + p_m$ is negated first and then removed accidentally, the resulting expression may be given by $p_2 + \dots + p_m$, which is equivalent to the faulty expression where the first term is being omitted. We will exclude all these two types of faulty expressions for double faults. In this report, when a Boolean expression is said to have a double fault, we mean that the faulty Boolean expression (1) differs from the original expression by two syntactic changes, (2) is not equivalent to the original expression, and (3) is not equivalent to any faulty expression that results from any single fault class as discussed in Section 2. For ease of reference, such expression is referred to as *double-fault expression*.

When two single faults are committed in a given Boolean expression, different order of their occurrences may result in the same double-fault expression. For example, when two different terms are negated, the resulting expression is always the same, irrespectively of which term being negated first. Since the ordering of the faults is not important, this is referred to as *double fault without ordering*. On the other hand, it is also possible that different order of occurrences of two faults will result in different expressions, which are not equivalent to each other. This situation is referred to as *double fault with ordering*. In the rest of this section, we first discuss the case of double fault without ordering. The case of double fault with ordering will be discussed in next section.

We now introduce different types of double faults without ordering. Given a Boolean expression S , suppose that two single fault classes F_1 and F_2 are committed in S changing E_1 and E_2 in S to E'_1 and E'_2 , respectively. Since their order of occurrences will result in the same faulty implementation, the corresponding double fault class is simply referred to as F_1 and F_2 . The resulting double-fault implementation is denoted by $I_{F_1(E_1 \rightarrow E'_1), F_2(E_2 \rightarrow E'_2)}$. Table 1 lists all 15 types of double faults without ordering for those five single fault classes discussed in Section 2.2.

Table 1: Types of double faults without ordering

	ENF	TNF	TOF	DORF	CORF
ENF	✓	✓	✓	✓	✓
TNF		✓	✓	✓	✓
TOF			✓	✓	✓
DORF				✓	✓
CORF					✓

3.1 ENF with Other Faults

ENF and ENF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ are negated. We use $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}}), ENF(p_{i_2} + \dots + p_{h_2} \rightarrow \overline{p_{i_2} + \dots + p_{h_2}})}$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ are mutually exclusive, that is $\{i_1, \dots, h_1\} \cap \{i_2, \dots, h_2\} = \emptyset$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + \overline{p_{i_2} + \dots + p_{h_2}} + \dots + p_m \quad (1)$$

Case 2. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ are such that one subexpression contains another, but they are not equal. Without loss of generality, we can assume $\{i_2, \dots, h_2\} \subsetneq \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_m \quad (2)$$

We do not consider the following two situations. First, the two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ are exactly the same, that is $\{i_1, \dots, h_1\} = \{i_2, \dots, h_2\}$, because the implementation

is equivalent to S . Second, the two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ have some common terms, but one does not contain the other. For example, if $S = ab + cd + ef + gh$, we do not consider the situation that the subexpressions $ab + cd$ and $cd + ef$ can both be negated.

ENF and TNF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the subexpression $p_{i_1} + \dots + p_{h_1}$ and the term p_{i_2} are negated. We use $I_{ENF}(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}}), TNF(p_{i_2} \rightarrow \overline{p_{i_2}})$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$.

Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + \overline{p_{i_2}} + \dots + p_m \quad (3)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + \overline{p_{i_2}} + \dots + p_{h_1}} + \dots + p_m \quad (4)$$

ENF and TOF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the subexpression $p_{i_1} + \dots + p_{h_1}$ is negated and the term p_{i_2} is omitted. We use $I_{ENF}(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}}), TOF(p_{i_2} \rightarrow)$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$.

Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m \quad (5)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_{h_1}} + \dots + p_m \quad (6)$$

ENF and DORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the subexpression $p_{i_1} + \dots + p_{h_1}$ is negated and the subexpression $p_{i_2} + p_{i_2+1}$ is implemented as $p_{i_2}p_{i_2+1}$. We use $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})}, DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ to denote the corresponding faulty implementation, which can be further classified into the following four cases:

Case 1. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + p_{i_2+1}$ are mutually exclusive, that is $\{i_1, \dots, h_1\} \cap \{i_2, i_2 + 1\} = \emptyset$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2}p_{i_2+1} + \dots + p_m \quad (7)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains exactly one of the two terms p_{i_2} and p_{i_2+1} , that is either $h_1 = i_2$ or $i_2 + 1 = i_1$. Without loss of generality, we can assume the latter case, that is $i_2 = h_1$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}}p_{h_1+1} + \dots + p_m \quad (8)$$

Case 3. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the subexpression $p_{i_2} + p_{i_2+1}$, but they are not equal, that is $\{i_2, i_2 + 1\} \subsetneq \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2}p_{i_2+1} + \dots + p_{h_1}} + \dots + p_m \quad (9)$$

Case 4. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + p_{i_2+1}$ are exactly same, that is $\{i_2, i_2 + 1\} = \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1}p_{i_1+1}} + \dots + p_m \quad (10)$$

ENF and CORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the subexpression $p_{i_1} + \dots + p_{h_1}$ is wrongly negated and the term p_{i_2} is wrongly implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. We use $I_{ENF(p_{i_1}+\dots+p_{h_1} \rightarrow \overline{p_{i_1}+\dots+p_{h_1}}), CORF(p_{i_2} \rightarrow p_{i_2,1,j_2}+p_{i_2,j_2+1,k_{i_2}})}$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$.

Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m \quad (11)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_{h_1}} + \dots + p_m \quad (12)$$

3.2 TNF with Other Faults

TNF and TNF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose two different terms p_{i_1} and p_{i_2} are negated. We use $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}}), TNF(p_{i_2} \rightarrow \overline{p_{i_2}})}$ to denote the corresponding faulty implementation. We do not consider the situation where the same term is negated twice (that is, $i_1 = i_2$) because the implementation is then equivalent to the original expression S . Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2}} + \dots + p_m \quad (13)$$

TNF and TOF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the term p_{i_1} is wrongly negated and the term p_{i_2} is wrongly omitted. We use

$I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}}), TOF(p_{i_2} \rightarrow)}$ to denote the corresponding faulty implementation. We do not consider the situation where both TNF and TOF occur at the same term (that is $i_1 = i_2$). Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m \quad (14)$$

TNF and DORF Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the term p_{i_1} is negated and the subexpression $p_{i_2} + p_{i_2+1}$ is implemented as $p_{i_2}p_{i_2+1}$. We use $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}}), DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})}$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The subexpression $p_{i_2} + p_{i_2+1}$ does not contain the term p_{i_1} , that is $i_1 \notin \{i_2, i_2 + 1\}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2}p_{i_2+1} + \cdots + p_m \quad (15)$$

Case 2. The subexpression $p_{i_2} + p_{i_2+1}$ contains the term p_{i_1} , that is $i_1 \in \{i_2, i_2 + 1\}$. Hence, there are two possible cases, namely $i_1 = i_2$ and $i_1 = i_2 + 1$. Without loss of generality, we can assume $i_1 = i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}}p_{i_1+1} + \cdots + p_m \quad (16)$$

TNF and CORF Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the term p_{i_1} is negated and the term p_{i_2} is implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. We use $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}}), CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})}$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The 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

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (17)$$

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

$$p_1 + \cdots + \overline{p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}} + \cdots + p_m \quad (18)$$

3.3 TOF with Other Faults

TOF and TOF Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose two different terms p_{i_1} and p_{i_2} are omitted. We use $I_{TOF(p_{i_1} \rightarrow), TOF(p_{i_2} \rightarrow)}$ to denote the corresponding faulty implementation. We do not consider the situation where the same term is omitted twice (that is, $i_1 = i_2$) because the implementation is then equivalent to the term being omitted which is a single TOF. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m \quad (19)$$

TOF and DORF Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the term p_{i_1} is omitted and the subexpression $p_{i_2} + p_{i_2+1}$ is implemented as $p_{i_2}p_{i_2+1}$. We use $I_{TOF(p_{i_1} \rightarrow), DORF(p_{i_2}+p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})}$ to denote the corresponding faulty implementation. We do not consider the situation where the subexpression contains the omitted term (that is, $i_1 = i_2$ or $i_1 = i_2 + 1$) because the implementation is then equivalent to the term being omitted which is a single TOF. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_{i_2}p_{i_2+1} + \cdots + p_m \quad (20)$$

TOF and CORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the term p_{i_1} is omitted and the term p_{i_2} is implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. We use $I_{TOF}(p_{i_1} \rightarrow \cdot), CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ to denote the corresponding faulty implementation. We do not consider the situation where both TOF and CORF occur at the same term (that is, $i_1 = i_2$) because the resulting expressions are not the same. Hence, it should be discussed in Section 4 when we consider double faults with ordering. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m \quad (21)$$

3.4 DORF with Other Faults

DORF and DORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ are implemented as $p_{i_1}p_{i_1+1}$ and $p_{i_2}p_{i_2+1}$, respectively. We use $I_{DORF}(p_{i_1} + p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}), DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ are mutually exclusive, that is $\{i_1, i_1 + 1\} \cap \{i_2, i_2 + 1\} = \emptyset$. Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1}p_{i_1+1} + \dots + p_{i_2}p_{i_2+1} + \dots + p_m \quad (22)$$

Case 2. The two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ have exactly one term in common, that is $\{i_1, i_1 + 1\} \cap \{i_2, i_2 + 1\} = A$ and there is only one element in set A . Hence, there are two possible cases, namely $i_1 + 1 = i_2$ and $i_2 + 1 = i_1$. Without loss of generality, we can assume $i_1 + 1 = i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1}p_{i_1+1}p_{i_1+2} + \dots + p_m \quad (23)$$

We do not consider the situation where two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ are exactly the same, because the implementation is equivalent to a single DORF.

DORF and CORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose the subexpression $p_{i_1} + p_{i_1+1}$ of S is implemented as $p_{i_1}p_{i_1+1}$ and the term p_{i_2} of S is implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. We use $I_{DORF}(p_{i_1}+p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}), CORF(p_{i_2} \rightarrow p_{i_2,1,j_2}+p_{i_2,j_2+1,k_{i_2}})$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The subexpression $p_{i_1} + p_{i_1+1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, i_1 + 1\}$. Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1}p_{i_1+1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m \quad (24)$$

Case 2. The subexpression $p_{i_1} + p_{i_1+1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, i_1 + 1\}$. Hence, there are two possible cases, namely $i_1 = i_2$ and $i_1 + 1 = i_2$. Without loss of generality, we can assume $i_1 = i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_2}}p_{i_1+1} + \dots + p_m \quad (25)$$

3.5 CORF with Other Faults

CORF and CORF Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose two terms p_{i_1} and p_{i_2} are implemented as $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$ and $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, respectively, where $p_{i_1} = p_{i_1,1,j_1} \cdot p_{i_1,j_1+1,k_{i_1}}$ and $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. We use $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1}+p_{i_1,j_1+1,k_{i_1}}), CORF(p_{i_2} \rightarrow p_{i_2,1,j_2}+p_{i_2,j_2+1,k_{i_2}})$ to denote the corresponding faulty implementation, which can be further classified into the following two cases:

Case 1. The term p_{i_1} and the term p_{i_2} are different terms, 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

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (26)$$

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 where the split positions are same (that is, $j_1 = j_2$) because the implementation is then equivalent to a single CORF. Without loss of generality, we can assume $j_1 < j_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,j_2} + p_{i_1,j_2+1,k_{i_1}} + \cdots + p_m \quad (27)$$

4 Double Faults with Ordering

In this section, we discuss different types of double fault with ordering and their corresponding faulty implementations. As a reminder, for the five single fault classes described in Section 2, there are altogether 25 double fault classes with ordering. As discussed previously, double fault with ordering refers to the situation that, when two individual faults of a double fault occur in different orders, the resulting faulty implementations may be different.

Given a Boolean expression S , let F_1 and F_2 be two single fault classes changing 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 $I_{F_1(E_1 \rightarrow E'_1) \otimes F_2(E_2 \rightarrow E'_2)}$. For example, let S be $abc + cd + ef$. When TNF occurs at the first term abc of S and then CORF occurs at the ‘.’ operator between a and b , the corresponding implementation is $I_{TNF(abc \rightarrow \overline{abc}) \otimes CORF(\overline{abc} \rightarrow \overline{a+bc})}$ which is equivalent to $\overline{a+bc} + cd + ef$. This expression is considered to differ from S by two syntactic changes.

4.1 ENF First, then Other Faults

Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose an ENF is committed first by negating the subexpression $p_{i_1} + \dots + p_{h_1}$. The corresponding faulty implementation is $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})} = p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_m$.

ENF and ENF After the first ENF is made on S , another subexpression $p_{i_2} + \dots + p_{h_2}$ is then negated. Let $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}}) \otimes ENF(p_{i_2} + \dots + p_{h_2} \rightarrow \overline{p_{i_2} + \dots + p_{h_2}})}$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + \dots + p_{h_2}$ are mutually exclusive, that is $\{i_1, \dots, h_1\} \cap \{i_2, \dots, h_2\} = \emptyset$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + \overline{p_{i_2} + \dots + p_{h_2}} + \dots + p_m \quad (28)$$

Case 2. One subexpression is contained in another, but they are not the same.

(a) The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the subexpression $p_{i_2} + \dots + p_{h_2}$. That is, $\{i_2, \dots, h_2\} \subsetneq \{i_1, \dots, h_1\}$. The implementation is then equivalent to

$$p_1 + \dots + \overline{p_{i_1} + \dots + \overline{p_{i_2} + \dots + p_{h_2}} + \dots + p_{h_1}} + \dots + p_m \quad (29a)$$

(b) The subexpression $p_{i_2} + \dots + p_{h_2}$ contains the subexpression $p_{i_1} + \dots + p_{h_1}$. That is, $\{i_1, \dots, h_1\} \subsetneq \{i_2, \dots, h_2\}$. The implementation is then equivalent to

$$p_1 + \dots + \overline{p_{i_2} + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{h_2}} + \dots + p_m \quad (29b)$$

As discussed in Section 3.1, we do not consider the situations where (1) the two subexpressions are exactly same, and (2) the two subexpressions have some common terms, but one does not contain the other.

ENF and TNF After the ENF is made on S , the i_2 -th term p_{i_2} is then negated. Let $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})} \otimes TNF(p_{i_2} \rightarrow \overline{p_{i_2}})$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + \overline{p_{i_2}} + \dots + p_m \quad (30)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + \overline{p_{i_2}} + \dots + p_{h_1}} + \dots + p_m \quad (31)$$

ENF and TOF After the ENF is made on S , the i_2 -th term p_{i_2} is then omitted. Let $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})} \otimes TOF(p_{i_2} \rightarrow)$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m \quad (32)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_{h_1}} + \dots + p_m \quad (33)$$

ENF and DORF After the ENF is made on S , another subexpression $p_{i_2} + p_{i_2+1}$ is then wrongly implemented as $p_{i_2}p_{i_2+1}$. Let $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})} \otimes DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ be the corresponding faulty implementation. We have the following four cases:

Case 1. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + p_{i_2+1}$ are mutually exclusive, that is $\{i_1, \dots, h_1\} \cap \{i_2, i_2 + 1\} = \emptyset$. Without loss of generality, we can assume $h_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2}p_{i_2+1} + \dots + p_m \quad (34)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains either the term p_{i_2} or the term p_{i_2+1} . Without loss of generality, we can assume that the subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} (that is, $i_2 = h_1$). The implementation is then equivalent to

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}p_{h_1+1}} + \dots + p_m \quad (35)$$

Case 3. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the subexpression $p_{i_2} + p_{i_2+1}$, but they are not same, that is $\{i_2, i_2 + 1\} \subsetneq \{i_1, \dots, h_1\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2}p_{i_2+1} + \dots + p_{h_1}} + \dots + p_m \quad (36)$$

Case 4. The two subexpressions $p_{i_1} + \dots + p_{h_1}$ and $p_{i_2} + p_{i_2+1}$ are exactly same, that is $\{i_2, i_2 + 1\} = \{i_1, \dots, h_1\}$. Hence, $i_1 = i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1}p_{i_1+1}} + \dots + p_m \quad (37)$$

ENF and CORF After the ENF is made on S , the term p_{i_2} is then implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. Let $I_{ENF(p_{i_1} + \dots + p_{h_1} \rightarrow \overline{p_{i_1} + \dots + p_{h_1}})} \otimes CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_1} + \dots + p_{h_1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, \dots, h_1\}$.

Without loss of generality, we can assume $h_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{h_1}} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m \quad (38)$$

Case 2. The subexpression $p_{i_1} + \dots + p_{h_1}$ contains the term p_{i_2} , that is $i_2 \in \{i_1, \dots, h_1\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_{h_1}} + \dots + p_m \quad (39)$$

4.2 TNF First, then Other Faults

Let $S = p_1 + \dots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose a TNF is committed first by negating the term p_{i_1} . The corresponding faulty implementation is

$$I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}})} = p_1 + \dots + \overline{p_{i_1}} + \dots + p_m.$$

TNF and ENF After the TNF is made on S , the subexpression $p_{i_2} + \dots + p_{h_2}$ is then negated. Let

$I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}}) \otimes ENF(p_{i_2} + \dots + p_{h_2} \rightarrow \overline{p_{i_2} + \dots + p_{h_2}})}$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_2} + \dots + p_{h_2}$ does not contain the term p_{i_1} , that is $i_1 \notin \{i_2, \dots, h_2\}$.

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

$$p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2} + \dots + p_{h_2}} + \dots + p_m \quad (40)$$

Case 2. The subexpression $p_{i_2} + \dots + p_{h_2}$ contains the term p_{i_1} , that is $i_1 \in \{i_2, \dots, h_2\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_2} + \dots + \overline{p_{i_1}} + \dots + p_{h_2}} + \dots + p_m \quad (41)$$

TNF and TNF After the first TNF is made on S , the i_2 -th term p_{i_2} is then negated. Let $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}})} \otimes TNF(p_{i_2} \rightarrow \overline{p_{i_2}})$ be the corresponding faulty implementation. As discussed in Section 3.2, we do not consider the situation that the two terms are exactly the same (that is, $i_1 = i_2$). Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + \overline{p_{i_2}} + \cdots + p_m \quad (42)$$

TNF and TOF After the TNF is made on S , the i_2 -th term p_{i_2} is then omitted. Let $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}})} \otimes TOF(p_{i_2} \rightarrow)$ be the corresponding faulty implementation. We do not consider the situation where both TNF and TOF occur at the same term (that is, $i_1 = i_2$). Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m \quad (43)$$

TNF and DORF After the TNF is made on S , the subexpression $p_{i_2} + p_{i_2+1}$ is then wrongly implemented as $p_{i_2}p_{i_2+1}$. Let $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}})} \otimes DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_2} + p_{i_2+1}$ does not contain the term p_{i_1} , that is $i_1 \notin \{i_2, i_2 + 1\}$.

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

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2}p_{i_2+1} + \cdots + p_m \quad (44)$$

Case 2. The subexpression $p_{i_2} + p_{i_2+1}$ contains the term p_{i_1} , that is $i_1 \in \{i_2, i_2 + 1\}$. Without loss of generality, we can assume $i_1 = i_2$, the implementation is equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}}p_{i_1+1} + \cdots + p_m \quad (45)$$

TNF and CORF After the TNF is made on S , the term p_{i_2} is then implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. Let $I_{TNF(p_{i_1} \rightarrow \overline{p_{i_1}})} \otimes CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ be the corresponding faulty implementation. We have 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 equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1}} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (46)$$

Case 2. The two terms p_{i_1} and p_{i_2} are the same, that is $i_1 = i_2$. The implementation is equivalent to the following expression $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$

$$p_1 + \cdots + \overline{p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}} + \cdots + p_m \quad (47)$$

4.3 TOF First, then Other Faults

Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose a TOF is committed first by omitting the term p_{i_1} . The corresponding faulty implementation is $I_{TOF(p_{i_1} \rightarrow)} = p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_m$.

TOF and ENF After the TOF is made on S , the subexpression $p_{i_2} + \cdots + p_{h_2}$ is then negated. Let $I_{TOF(p_{i_1} \rightarrow)} \otimes ENF(p_{i_2} + \cdots + p_{h_2} \rightarrow \overline{p_{i_2} + \cdots + p_{h_2}})$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_2} + \cdots + p_{h_2}$ does not contain the term p_{i_1} , that is $i_1 \notin \{i_2, \dots, h_2\}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + \overline{p_{i_2} + \cdots + p_{h_2}} + \cdots + p_m \quad (48)$$

Case 2. The subexpression $p_{i_2} + \dots + p_{h_2}$ contains the term p_{i_1} , that is $i_1 \in \{i_2, \dots, h_2\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_2} + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{h_2}} + \dots + p_m \quad (49)$$

TOF and TNF After the TOF is made on S , the i_2 -th term p_{i_2} is then negated. Let $I_{TOF(p_{i_1} \rightarrow)} \otimes TNF(p_{i_2} \rightarrow \overline{p_{i_2}})$ be the corresponding faulty implementation. We do not consider the situation where the same term is omitted and then negated (that is, $i_1 = i_2$). Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + \overline{p_{i_2}} + \dots + p_m \quad (50)$$

TOF and TOF After the first TOF is made on S , the i_2 -th term p_{i_2} is then omitted. Let $I_{TOF(p_{i_1} \rightarrow)} \otimes TOF(p_{i_2} \rightarrow)$ be the corresponding faulty implementation. We do not consider the situation where the same term is omitted first and then omitted again. Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m \quad (51)$$

TOF and DORF After the TOF is made on S , the subexpression $p_{i_2} + p_{i_2+1}$ is then wrongly implemented as $p_{i_2}p_{i_2+1}$. Let $I_{TOF(p_{i_1} \rightarrow)} \otimes DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ be the corresponding faulty implementation. We have the following two cases:

Case 1. The subexpression $p_{i_2} + p_{i_2+1}$ involves any subexpression of two consecutive terms other than $p_{i_1-1} + p_{i_1+1}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2}p_{i_2+1} + \dots + p_m \quad (52)$$

Case 2. The subexpression $p_{i_1-1} + p_{i_1+1}$ is implemented as $p_{i_1-1}p_{i_1+1}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1}p_{i_1+1} + \cdots + p_m \quad (53)$$

TOF and CORF After the TOF is made on S , the term p_{i_2} is wrongly implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. Let $I_{TOF}(p_{i_1} \rightarrow) \otimes CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ be the corresponding faulty implementation. We do not consider the situation where a term is omitted first and then a CORF is committed at the same term (that is, $i_1 = i_2$). Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1+1} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (54)$$

4.4 DORF First, then Other Faults

Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose a DORF is committed first by concatenating two terms p_{i_1} and p_{i_1+1} using the '+' operator. The corresponding faulty implementation is $I_{DORF}(p_{i_1} + p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}) = p_1 + \cdots + p_{i_1}p_{i_1+1} + \cdots + p_m$.

DORF and ENF After the DORF is made on S , the subexpression $p_{i_2} + \cdots + p_{h_2}$ is then negated. Let $I_{DORF}(p_{i_1} + p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}) \otimes ENF(p_{i_2} + \cdots + p_{h_2} \rightarrow \overline{p_{i_2} + \cdots + p_{h_2}})$ be the corresponding faulty implementation. We have the following four cases:

Case 1. The subexpressions $p_{i_2} + \cdots + p_{h_2}$ and $p_{i_1} + p_{i_1+1}$ are mutually exclusive, that is $\{i_1, i_1 + 1\} \cap \{i_2, \dots, h_2\} = \emptyset$. Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1}p_{i_1+1} + \cdots + \overline{p_{i_2} + \cdots + p_{h_2}} + \cdots + p_m \quad (55)$$

Case 2. The subexpression $p_{i_2} + \dots + p_{h_2}$ contains either p_{i_1} or p_{i_1+1} . Without loss of generality, we can assume the subexpression $p_{i_2} + \dots + p_{h_2}$ contains p_{i_1} , that is $i_1 = h_2$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_2} + \dots + p_{h_2}} p_{h_2+1} + \dots + p_m \quad (56)$$

Case 3. The subexpression $p_{i_2} + \dots + p_{h_2}$ contains the subexpression $p_{i_1} + p_{i_1+1}$, but they are not same, that is $\{i_1, i_1 + 1\} \subsetneq \{i_2, \dots, h_2\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_2} + \dots + p_{i_1} p_{i_1+1} + \dots + p_{h_2}} + \dots + p_m \quad (57)$$

Case 4. The two subexpressions $p_{i_2} + \dots + p_{h_2}$ and $p_{i_1} + p_{i_1+1}$ are exactly same, that is $\{i_1, i_1 + 1\} = \{i_2, \dots, h_2\}$. The implementation is equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} p_{i_1+1}} + \dots + p_m \quad (58)$$

DORF and TNF After the DORF is made on S , the i_2 -th term p_{i_2} is then negated. Let $I_{DORF(p_{i_1}+p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}) \otimes TNF(p_{i_2} \rightarrow \overline{p_{i_2}})}$ be the corresponding faulty implementation. We have the following three cases:

Case 1. The subexpression $p_{i_1} + p_{i_1+1}$ does not contain the term p_{i_2} . Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \dots + p_{i_1} p_{i_1+1} + \dots + \overline{p_{i_2}} + \dots + p_m \quad (59)$$

Case 2. The subexpression $p_{i_1} + p_{i_1+1}$ contains the term p_{i_2} . Without loss of generality, we can assume $i_2 = i_1$ following expression

$$p_1 + \dots + \overline{p_{i_1}} p_{i_1+1} + \dots + p_m \quad (60)$$

Case 3. The term p_{i_2} is the newly created term $p_{i_1} p_{i_1+1}$. The implementation is then equivalent to the following expression

$$p_1 + \dots + \overline{p_{i_1} p_{i_1+1}} + \dots + p_m \quad (61)$$

DORF and TOF After the DORF is made on S , the i_2 -th term p_{i_2} is then omitted. Let $I_{DORF(p_{i_1}+p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}) \otimes TOF(p_{i_2} \rightarrow)}$ be the corresponding faulty implementation. We do not consider the two situations where the term p_{i_2} is either p_{i_1} or p_{i_1+1} because the net effect is a single TOF. Therefore, we have the following two cases:

Case 1. The subexpression $p_{i_1} + p_{i_1+1}$ does not contain the term p_{i_2} . Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1}p_{i_1+1} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m \quad (62)$$

Case 2. The term p_{i_2} is the newly created term $p_{i_1}p_{i_1+1}$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1+2} + \cdots + p_m \quad (63)$$

DORF and DORF After the first DORF is made on S , another subexpression $p_{i_2} + p_{i_2+1}$ is then implemented as $p_{i_2}p_{i_2+1}$. Let $I_{DORF(p_{i_1}+p_{i_1+1} \rightarrow p_{i_1}p_{i_1+1}) \otimes DORF(p_{i_2}+p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})}$ be the corresponding faulty implementation. We do not consider the situation where the two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ are exactly the same because the implementation is equivalent to a single DORF. Therefore, we have the following two cases:

Case 1. The two subexpressions $p_{i_1} + p_{i_1+1}$ and $p_{i_2} + p_{i_2+1}$ are mutually exclusive, that is $\{i_1, i_1 + 1\} \cap \{i_2, i_2 + 1\} = \emptyset$. Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1}p_{i_1+1} + \cdots + p_{i_2}p_{i_2+1} + \cdots + p_m \quad (64)$$

Case 2. Either the term p_{i_2} or the term p_{i_2+1} is the newly created term $p_{i_1}p_{i_1+1}$. Without loss of generality, we can assume the term p_{i_2} is the newly created term $p_{i_1}p_{i_1+1}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1}p_{i_1+1}p_{i_1+2} + \cdots + p_m \quad (65)$$

DORF and CORF After the DORF is made on S , the term p_{i_2} is implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. Let $I_{DORF}(p_{i_1} + p_{i_1+1} \rightarrow p_{i_1} p_{i_1+1}) \otimes CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ be the corresponding faulty implementation. We do not consider the situations where the newly created term $p_{i_1} p_{i_1+1}$ is splitted into $p_{i_1} + p_{i_1+1}$ because the resulting expression is equivalent to the original expression. Therefore, we have the following two cases:

Case 1. The subexpression $p_{i_1} + p_{i_1+1}$ does not contain the term p_{i_2} , that is $i_2 \notin \{i_1, i_1 + 1\}$.

Without loss of generality, we can assume $i_1 + 1 < i_2$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1} p_{i_1+1} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (66)$$

Case 2. The subexpression $p_{i_1} + p_{i_1+1}$ contains the term p_{i_2} . Without loss of generality, we can assume $i_2 = i_1$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_2} + p_{i_1,j_2+1,k_{i_1}} \cdot p_{i_1+1} + \cdots + p_m \quad (67)$$

4.5 CORF First, then Other Faults

Let $S = p_1 + \cdots + p_m$ be a Boolean specification in irredundant disjunctive normal form. Suppose a CORF is committed first by splitting the term p_{i_1} into $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$, where $p_{i_1} = p_{i_1,1,j_1} \cdot p_{i_1,j_1+1,k_{i_1}}$. The corresponding faulty implementation is $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) = p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_m$.

CORF and ENF After the CORF is made on S , the subexpression $p_{i_2} + \cdots + p_{h_2}$ is then negated.

Let $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) \otimes ENF(p_{i_2} + \cdots + p_{h_2} \rightarrow \overline{p_{i_2} + \cdots + p_{h_2}})$ be the corresponding faulty implementation. We have the following four cases:

Case 1. The subexpression $p_{i_2} + \cdots + p_{h_2}$ does not contain $p_{i_1,1,j_1}$ and $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is equivalent to the following

expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + \overline{p_{i_2} + \cdots + p_{h_2}} + \cdots + p_m \quad (68)$$

Case 2. The subexpression $p_{i_2} + \cdots + p_{h_2}$ contains $p_{i_1,1,j_1}$ and $p_{i_1,j_1+1,k_{i_1}}$. The implementation is equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_2} + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{h_2}} + \cdots + p_m \quad (69)$$

Case 3. The subexpression $p_{i_2} + \cdots + p_{h_2}$ contains exactly one of the two newly created term. Without loss of generality, we can assume that the subexpression $p_{i_2} + \cdots + p_{h_2}$ contains $p_{i_1,j_1+1,k_{i_1}}$. Hence, the term p_{i_2} actually refers to $p_{i_1,j_1+1,k_{i_1}}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + \overline{p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{h_2}} + \cdots + p_m \quad (70)$$

Case 4. The subexpression $p_{i_2} + \cdots + p_{h_2}$ is exactly the newly created subexpression $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$. That is, the newly created subexpression is then negated. The implementation is equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}} + \cdots + p_m \quad (71)$$

CORF and TNF After the CORF is made on S , the i_2 -th term p_{i_2} is then negated. Let $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) \otimes TNF(p_{i_2} \rightarrow \overline{p_{i_2}})$ be the corresponding faulty implementation. We have the following three cases:

Case 1. The term p_{i_2} is neither $p_{i_1,1,j_1}$ nor $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + \overline{p_{i_2}} + \cdots + p_m \quad (72)$$

Case 2. The negated term is either $p_{i_1,1,j_1}$ or $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume that the negated term is $p_{i_1,j_1+1,k_{i_1}}$. The implementation is equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + \overline{p_{i_1,j_1+1,k_{i_1}}} + \cdots + p_m \quad (73)$$

Case 3. The negated term is p_{i_1} . The implementation is equivalent to the following expression

$$p_1 + \cdots + \overline{p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}} + \cdots + p_m \quad (74)$$

CORF and TOF After the CORF is committed on S , the i_2 -th term p_{i_2} is then omitted. Let $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) \otimes TOF(p_{i_2} \rightarrow)$ be the corresponding faulty implementation. We do not consider the situation where $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$ (originally belongs to the term p_{i_1}) is then omitted because the net effect is a single TOF. We have the following two cases:

Case 1. The term p_{i_2} is neither $p_{i_1,1,j_1}$ nor $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{i_2-1} + p_{i_2+1} + \cdots + p_m \quad (75)$$

Case 2. The term p_{i_2} is either $p_{i_1,1,j_1}$ or $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume that the omitted term is $p_{i_1,j_1+1,k_{i_1}}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1-1} + p_{i_1,1,j_1} + p_{i_1+1} + \cdots + p_m \quad (76)$$

CORF and DORF After the CORF is made on S , another subexpressions $p_{i_2} + p_{i_2+1}$ is wrongly implemented as $p_{i_2}p_{i_2+1}$. Let $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) \otimes DORF(p_{i_2} + p_{i_2+1} \rightarrow p_{i_2}p_{i_2+1})$ be the corresponding faulty implementation. We do not consider the situation where a DORF is committed on $p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}$ because the implementation is equivalent to the original expression S . We have the following two cases:

Case 1. The subexpression $p_{i_2} + p_{i_2+1}$ does not contain $p_{i_1,1,j_1}$ and $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{i_2}p_{i_2+1} + \cdots + p_m \quad (77)$$

Case 2. The subexpression $p_{i_2} + p_{i_2+1}$ contains either $p_{i_1,1,j_1}$ or $p_{i_1,j_1+1,k_{i_1}}$. Without loss of generality, we can assume that the subexpression $p_{i_2} + p_{i_2+1}$ contains $p_{i_1,j_1+1,k_{i_1}}$. Hence, the term p_{i_2+1} actually refers to $p_{i_1,j_1+1,k_{i_1}}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}p_{i_1+1} + \cdots + p_m \quad (78)$$

CORF and CORF After the CORF is made on S , the term p_{i_2} is implemented as $p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}}$, where $p_{i_2} = p_{i_2,1,j_2} \cdot p_{i_2,j_2+1,k_{i_2}}$. Let $I_{CORF}(p_{i_1} \rightarrow p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}}) \otimes CORF(p_{i_2} \rightarrow p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}})$ be the corresponding faulty implementation. We do not consider the situation where $i_1 = i_2$ and $j_1 = j_2$ because the implementation is then equivalent to a single CORF. We have the following two cases:

Case 1. The term p_{i_2} is neither $p_{i_1,1,j_1}$ nor $p_{i_1,j_1+1,k_{i_1}}$. Without loss the generality, we can assume $i_1 < i_2$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \cdots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \cdots + p_m \quad (79)$$

Case 2. The term p_{i_2} is either $p_{i_1,1,j_1}$ or $p_{i_1,j_1+1,k_{i_1}}$. Without loss the generality, we can assume that the term p_{i_2} is $p_{i_1,j_1+1,k_{i_1}}$. The implementation is then equivalent to the following expression

$$p_1 + \cdots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + p_{i_1,j_2+1,k_{i_1}} + \cdots + p_m \quad (80)$$

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 implementations of double faults without ordering with respect to those with ordering in the same fault category related to ENF, TNF, TOF, DORF, and CORF. Table 2 (respectively, 3, 4, 5 and 6) summarizes the situations of double faults with ENF (respectively TNF, TOF, DORF, and CORF) and other faults. Each row in these tables shows those faulty implementations 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 ENF and ENF without ordering which are given by Expressions (1) and (2) in Section 3. For the first subcase, it is equivalent to Expression (28) which corresponds to the first subcase of ENF and ENF with ordering as discussed in Section 4. For the second subcase of ENF and ENF without ordering, Expressions (2) and (29a) are equivalent. Moreover, it should be noted that, Expressions (2) and (29b)¹ are dual to each other. Hence, they are considered to be equivalent. Other rows in Table 2 show the equivalent faulty implementations of double faults of ENF and other faults with and without ordering.

In Table 3, the rows can also be interpreted in a similar manner as those in Table 2. For example, for the row of TNF and TNF, the possible faulty implementation given by Expression (13) is equivalent the faulty implementation given by Expression (42).

For the rows in Table 4, there are two different situations. First, some of them can be interpreted in a similar manner as those in Tables 2 and 3. Second, for other rows, the expressions of double faults with ordering do not have their counterparts in double faults without ordering in the same double fault class. For example, for the second subcase of TOF and DORF, Expression (53) does not have its counterpart in TOF and DORF without ordering.

¹Expressions (2) and (29b) are $p_1 + \dots + p_{i_1} + \dots + p_{i_2} + \dots + p_{h_2} + \dots + p_{h_1} + \dots + p_m$ (where $\{i_2, \dots, h_2\} \subsetneq \{i_1, \dots, h_1\}$) and $p_1 + \dots + p_{i_2} + \dots + p_{i_1} + \dots + p_{h_1} + \dots + p_{h_2} + \dots + p_m$ (where $\{i_1, \dots, h_1\} \subsetneq \{i_2, \dots, h_2\}$), respectively.

Table 2: Comparison of faulty expressions for double faults, ENF and other faults

Fault type	Double faults without ordering	Double faults with ordering, ENF occurs first
ENF and ENF	1	28
	2	29a, 29b ^a
ENF and TNF	3	30
	4	31
ENF and TOF	5	32
	6	33
ENF and DORF	7	34
	8	35
	9	36
	10	37
ENF and CORF	11	38
	12	39

^aExpressions (2) and (29b) are dual to each other.

Table 3: Comparison of faulty expressions for double faults, TNF and other faults

Fault type	Double faults without ordering	Double faults with ordering, TNF occurs first
TNF and ENF	3	40
	4	41
TNF and TNF	13	42
TNF and TOF	14	43
TNF and DORF	15	44
	16	45
TNF and CORF	17	46
	18	47

Table 4: Comparison of faulty expressions of double faults, TOF and other faults

Fault type	Double faults without ordering	Double faults with ordering, TOF occurs first
TOF and ENF	5	48
	6	49
TOF and TNF	14	50
TOF and TOF	19	51
TOF and DORF	20	52
	-	53
TOF and CORF	21	54

Table 5: Comparison of faulty expressions of double faults, DORF and other faults

Fault type	Double faults without ordering	Double faults with ordering, DORF occurs first
DORF and ENF	7	55
	8	56
	9	57
	10	58
DORF and TNF	15	59
	16	60
	-	61 (10)
DORF and TOF	20	62
	-	63 (19 ^a)
DORF and DORF	22	64
	23	65
DORF and CORF	24	66
	25	67

^aWhen both p_{i_1} and p_{i_1+1} are omitted, Expression (19) is equivalent to Expression (63).

Table 6: Comparison of faulty expressions of double faults, CORF and other faults

Fault type	Double faults without ordering	Double faults with ordering, CORF occurs first
CORF and ENF	11	68
	12	69
	-	70
	-	71 (18)
CORF and TNF	17	72
	18 (47)	-
	-	73
CORF and TOF	21	74 (18)
	-	75
CORF and DORF	24	76
	25	77
CORF and CORF	26	78
	27	79
		80

Similarly, for the rows in Table 5, there are two different situations. However, there are some subtle differences among them and those in Table 4. First, some of them can be interpreted in a similar manner as those in Tables 2 and 3. For example, for DORF and DORF, the faulty implementation given by Expression (22) is equivalent to that given by Expression (64). Second, for other rows, the faulty implementations 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 implementations of double faults without ordering in a different double fault class. For example, for the third subcase of DORF and TNF, the faulty implementation given by Expression (61) does not have its counterpart in DORF and TNF without ordering. However, it is equivalent to the faulty implementation given by Expression (10) in the double fault ENF and DORF without ordering in Section 3. Another example is the second subcase of DORF and TOF with ordering. The faulty implementation given by Expression (63) is such that the subexpression $p_{i_1} + p_{i_1+1}$ is wrongly committed as $p_{i_1} \cdot p_{i_1+1}$ and then this newly created term is omitted. This is, in fact, equivalent to the faulty implementation given by Expression (19) where both terms p_{i_1} and p_{i_1+1} are omitted.

For the rows in Table 6, there are four different situations. Again, there are some subtle differences among them and those in Tables 4 and 5. First, some of them can be interpreted in a similar manner as those in Tables 2 and 3. For example, for CORF and CORF in Table 6, the faulty implementation given by Expression (26) is equivalent to that given by Expression (79). Second, in some rows, the faulty implementations of double faults with ordering do not have their counterparts in double faults without ordering in the same double fault class. For example, for the third subcase of CORF and ENF in Table 6, the faulty implementation given by Expression (70) does not have its equivalent faulty implementation in double faults without ordering. Third, in some rows, the faulty implementations 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 implementations of double faults without ordering in a different double fault class. For example, for the fourth subcase of CORF and ENF in Table 6, the faulty implementation given by Expression (71) does not have its counterpart in CORF and ENF without ordering. However, it is equivalent to the faulty implementation given by

Expression (18) in the double fault TNF and CORF without ordering in Section 3. Fourth, in other rows, the faulty implementations 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 implementations of double faults with ordering in a different double fault class. For example, for the second subcase of CORF and TNF in Table 6, the faulty implementation given by Expression (18), a case of TNF and CORF without ordering, does not have its counterpart in CORF and TNF with ordering. However, it is equivalent to the faulty implementation given by Expression (47) in the double fault TNF and CORF with ordering, which is different from CORF and TNF with ordering.

As discussed in Sections 3 and 4, there are 27 possible faulty expressions of double faults without ordering and 53 possible faulty expressions of double faults with ordering. After comparing all these faulty expressions, 49 out of the 53 faulty expressions of double faults with ordering have their equivalent counterparts in double faults without ordering. The remaining 4 faulty implementations are given by Expressions (53), (70), (73) and (76). Hence, for the five single classes studied in this report, there are altogether 31 different double-fault expressions, 27 of them are from double fault classes without ordering and the remaining 4 are from double fault classes with ordering. Table 7 summarizes all these expressions.

6 Conclusion and Future Work

In this report, we study double faults the relationship of double faults with and without ordering based on five single fault types related to terms in Boolean expressions. A double fault is defined as the occurrence of two single faults. Since different order of occurrences of two single faults in a double fault may result in different faulty implementations, we further divide double faults into two categories, namely double faults with and without ordering. Double fault without ordering refers to the situation that different orderings of the two single faults will result in the same faulty implementation whereas double fault with ordering refers to the situation where the resulting faulty implementations

Table 7: Double fault and double-fault expression ($S = p_1 + \dots + p_m$)

(a) Double-fault expressions (1 – 27)

Fault class	Double-fault expression
ENF \times ENF	Case 1 ($i_1 < h_1 < i_2 < h_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} + \dots + \overline{p_{i_2}} + \dots + \overline{p_{h_2}} + \dots + p_m$ (1)
	Case 2 ($i_1 \leq i_2 < h_2 \leq h_1$ and $\{i_2, \dots, h_2\} \not\subseteq \{i_1, \dots, h_1\}$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2}} + \dots + \overline{p_{h_2}} + \dots + \overline{p_{h_1}} + \dots + p_m$ (2)
ENF \times TNF	Case 1 ($i_1 < h_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} + \dots + \overline{p_{i_2}} + \dots + p_m$ (3)
	Case 2 ($i_1 \leq i_2 \leq h_1$ and $i_1 < h_1$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2}} + \dots + \overline{p_{h_1}} + \dots + p_m$ (4)
ENF \times TOF	Case 1 ($i_1 < h_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m$ (5)
	Case 2 ($i_1 \leq i_2 \leq h_1$ and $i_1 < h_1$): $p_1 + \dots + \overline{p_{i_1}} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + \overline{p_{h_1}} + \dots + p_m$ (6)
ENF \times DORF	Case 1 ($i_1 < h_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} + \dots + p_{i_2} p_{i_2+1} + \dots + p_m$ (7)
	Case 2 ($i_1 < h_1 < m$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} p_{h_1+1} + \dots + p_m$ (8)
	Case 3 ($i_1 < m$): $p_1 + \dots + \overline{p_{i_1}} p_{i_1+1} + \dots + p_m$ (9)
	Case 4 ($i_1 \leq i_2 < h_1$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2}} p_{i_2+1} + \dots + \overline{p_{h_1}} + \dots + p_m$ (10)
ENF \times CORF	Case 1 ($i_1 < h_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{h_1}} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m$ (11)
	Case 2 ($i_1 \leq i_2 \leq h_1$ and $i_1 < h_1$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2,1,j_2}} + \overline{p_{i_2,j_2+1,k_{i_2}}} + \dots + \overline{p_{h_1}} + \dots + p_m$ (12)
TNF \times TNF	($i_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + \overline{p_{i_2}} + \dots + p_m$ (13)
TNF \times TOF	($i_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m$ (14)
TNF \times DORF	Case 1 ($i_1 < i_2 < m$): $p_1 + \dots + \overline{p_{i_1}} + \dots + p_{i_2} p_{i_2+1} + \dots + p_m$ (15)
	Case 2 ($i_1 < m$): $p_1 + \dots + \overline{p_{i_1}} p_{i_1+1} + \dots + p_m$ (16)
TNF \times CORF	Case 1 ($i_1 < i_2$): $p_1 + \dots + \overline{p_{i_1}} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m$ (17)
	Case 2 (both faults occur at p_{i_2}): $p_1 + \dots + \overline{p_{i_2,1,j_2}} + \overline{p_{i_2,j_2+1,k_{i_2}}} + \dots + p_m$ (18)
TOF \times TOF	($i_1 < i_2$): $p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2-1} + p_{i_2+1} + \dots + p_m$ (19)
TOF \times DORF	($i_1 < i_2 < m$): $p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2} p_{i_2+1} + \dots + p_m$ (20)
TOF \times CORF	($i_1 < i_2$): $p_1 + \dots + p_{i_1-1} + p_{i_1+1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m$ (21)
DORF \times DORF	Case 1 ($i_1 < i_2 < m$): $p_1 + \dots + p_{i_1} p_{i_1+1} + \dots + p_{i_2} p_{i_2+1} + \dots + p_m$ (22)
	Case 2 ($i_1 < m - 1$): $p_1 + \dots + p_{i_1} p_{i_1+1} p_{i_1+2} + \dots + p_m$ (23)
DORF \times CORF	Case 1 ($i_1 < i_2 - 1$): $p_1 + \dots + p_{i_1} p_{i_1+1} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m$ (24)
	Case 2 ($i_1 < m$): $p_1 + \dots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} p_{i_1+1} + \dots + p_m$ (25)
CORF \times CORF	Case 1 ($i_1 < i_2$): $p_1 + \dots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + \dots + p_{i_2,1,j_2} + p_{i_2,j_2+1,k_{i_2}} + \dots + p_m$ (26)
	Case 2 (both faults occur at p_{i_1}): $p_1 + \dots + p_{i_1,1,j_1} + p_{i_1,j_1+1,k_{i_1}} + p_{i_1,j_2+1,k_{i_1}} + \dots + p_m$ (27)

(b) Four double-fault expressions (53, 70, 73, and 76)^a due to double fault with ordering

Fault class	Double-fault expression
TOF \times DORF	($1 < i_1 < m$): $p_1 + \dots + p_{i_1-1} p_{i_1+1} + \dots + p_m$ (53)
CORF \times ENF	($i_1 < h_2$): $p_1 + \dots + p_{i_1,1,j_1} + \overline{p_{i_1,j_1+1,k_{i_1}}} + \dots + \overline{p_{h_1}} + \dots + p_m$ (70)
CORF \times TNF	$p_1 + \dots + p_{i_1,1,j_1} + \overline{p_{i_1,j_1+1,k_{i_1}}} + \dots + p_m$ (73)
CORF \times TOF	$p_1 + \dots + p_{i_1,1,j_1} + \dots + p_m$ (76)

are different. The five single fault classes considered are expression negation fault (ENF), term negation fault (TNF), term omission fault (TOF), disjunctive operator reference fault (DORF), and conjunctive operator reference fault (CORF).

Among the five single fault classes, there are 15 types of double fault without ordering resulting in 27 distinct non-equivalent faulty implementations. On the other hand, there are 25 types of double fault with ordering resulting in 53 faulty implementations. It is 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. As a result, altogether there are 31 different faulty expressions for double fault classes considered in this report.

Such an analysis of the relationship of double faults helps us to better understand how these double faults occur within Boolean expressions and how they interact with each other. Based on these 31 faulty implementations, detection conditions can be derived to design test cases aiming at the detection of the corresponding double faults. Moreover, further analysis of double faults related to literals in Boolean expressions are needed to have a more comprehensive understanding on double faults related to Boolean expressions.

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. 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. T. Yu. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 14(3):247 – 276, 2005.
- [6] B. Marick. Two experiments in software testing. Technical Report Technical Report UIUCDCS-R-90-1644, University of Illinois at Urbana-Champaign, 1990.
- [7] L. J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844 – 857, 1990.
- [8] A. J. Offutt. Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology*, 1(1):5–20, 1992.
- [9] 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.
- [10] 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.
- [11] 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.