CHAPTER 3

EFFECTIVENESS OF THE FAULT CONDITIONS

3.1 INTRODUCTION

Software testing aims at detecting software faults that are the result of human errors during software development. Specification-based testing derives test cases from specification rather than the actual implementation. A fault-based approach of generating test cases aims at detecting certain types of faults.

In the past, many fault based approaches on selecting test cases from the specification has been proposed (Chen et al 1997, Chilenski et al 1994, Foster 1984, Tai 1990, Tai 1996, Vouk et al 1994, Weyuker et al 1994). Because the difference between an implementation and its specification is the result of human error, some type of faults may be virtually impossible to predict in advance (Kuhn 1999). Nevertheless, some fault classes can be hypothesized and test sets can be derived to detect them.

Various fault classes were discussed by Chen et al (1997) and Lau et al (2001). A new fault is introduced and the generalized fault condition presented by Kuhn (1999) is applied to generate test cases for all the faults. For larger sized formulae such as the 14 literal specification given by Weyuker et al (1994), the generalized fault condition proposed by Kuhn (1999) is very complex. The effectiveness of the method proposed by Kuhn (1999) and Tsuchiya et al (2002) to generate test cases for various faults is analyzed and simplified fault conditions for some faults are proposed.
Section 3.2 explains the various faults that have been taken for analysis. Section 3.3 discusses the methodology carried out in conducting the experiments and analyzing the results. Section 3.4 gives an overview of the fault condition given by Kuhn (1999) and later gives the fault condition for various faults. Section 3.5 discusses the actual results obtained. The results are applied to TCAS II specifications in Section 3.6. Section 3.7 discusses the significance of the results.

3.2 FAULT CLASSES

Eight fault classes have been analyzed extensively. Some of these faults have been discussed by Chen et al (1997) and Lau et al (2001). The fault classes are:

- Expression Negation Fault (ENF)
- Term Negation Fault (TNF)
- Literal Negation Fault (LNF)
- Literal Reference Fault (LRF)
- Term Omission Fault (TOF)
- Literal Omission Fault (LOF)
- Literal Insertion Fault (LIF)

Kuhn (1999) proposed a fault condition to detect various faults. The fault condition was applied to these faults and test sets were derived. Apart from these, a new fault class is proposed namely,

- Term Insertion Fault (TIF)

Kuhn (1999) studied several fault classes that may occur during the implementation of a boolean expression $S$ in Disjunctive Normal Form (DNF). The faults he analysed were Expression Negation Fault, Variable Negation Fault and Variable Reference Fault. Although Kuhn (1999) used the same terms as those used...
by Weyuker et al (1994), he actually used them in a different sense. For example, for Variable Negation Fault, Kuhn (1999) refers to the fault which negates every occurrence of a particular variable in the boolean expression, whereas Weyuker et al (1994) refers to the fault which negates only one occurrence of a particular variable.

Similarly, for Variable Reference Fault, Kuhn (1999) refers to the fault which replaces every occurrence of a particular variable by another variable, whereas Weyuker et al (1994) refers to the fault which replaces only one occurrence of the variable by another variable. Practically, the fault models proposed by Weyuker et al (1994) and Lau et al (2001) have a greater impact than the one proposed by Kuhn (1999) because these fault classes more closely model the type of fault that humans are likely to make.

These faults are related to the operands in the boolean expression that may occur during implementation due to human error. The faults are defined below:

3.2.1 Expression Negation Fault (ENF)

In an Expression Negation Fault, the boolean specification is implemented as its negation. Let $S$ denote a boolean specification in DNF and $ENF$ denote its faulty implementation. The expressions for $S$ and $ENF$ are given below.

$$S = P_1 \lor P_2 \lor \ldots \lor P_1 \lor \ldots \lor P_m \quad (3.1)$$

$$ENF = \overline{P_1} \lor \overline{P_2} \lor \ldots \lor \overline{P_1} \lor \ldots \lor \overline{P_m} \quad (3.2)$$

where $P_1, P_2, \ldots, P_m$ refers to the terms in $S$. $m$ is the number of terms in $S$ and $m>1$. The specification thus implemented will be equivalent to $\overline{S}$. For example, the boolean specification $((a \land b) \lor (c \land d))$ is implemented as $\overline{((a \land b) \lor (c \land d))}$. 
When \( m = 1 \), the fault effectively becomes a Term Negation Fault. For example, the boolean specification \((a \land b)\) is implemented as \((\overline{a} \land b)\), which is same as negating the first term.

### 3.2.2 Term Negation Fault (TNF)

In a Term Negation Fault, a term of the boolean specification is implemented as its negation. Let \( S \) be a boolean specification in DNF and \( TNF \) denote its faulty implementation. The expression for \( S \) is given in Equation (3.1) and the expression for \( TNF \) is given below.

\[
TNF = P_1 \lor P_2 \lor \ldots \lor \overline{P_i} \lor \ldots \lor P_m \tag{3.3}
\]

where the \( i^{th} \) term \( P_i \) alone is negated. \( P_1, P_2, \ldots, P_m \) refers to the terms in \( S \). \( m \) is the number of terms in \( S \) and \( m \geq 1 \). For example, the boolean specification \(((a \land b) \lor (c \land d))\) is implemented as \(((\overline{a} \land b) \lor (c \land d))\).

### 3.2.3 Literal Negation Fault (LNF)

In a Literal Negation Fault, a literal in a term of the boolean specification is negated to get the faulty specification. Let \( S \) be a boolean specification as given in Equation (3.1) and \( LNF \) denote its faulty implementation. The expressions for the term \( P_i \) in \( S \) and \( LNF \) is shown below.

\[
S(P_i) = x_{i1} \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land \overline{x_{i,j-1}} \land x_{ij} \land \overline{x_{i,j+1}} \land \ldots \land x_{ik} \tag{3.4}
\]

\[
LNF(P_i) = x_{i1} \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land \overline{x_{i,j-1}} \land x_{ij} \land \overline{x_{i,j+1}} \land \ldots \land x_{ik} \tag{3.5}
\]
where the $j^{th}$ literal in the $i^{th}$ term alone is negated. $x_{i1}, x_{i2}, \ldots, x_{ik_i}$ are the literals in the term $P_i$ and $k_i (k_i > 1)$ is the number of literals in the $i^{th}$ term $P_i$. For example, the boolean specification $((a \land b) \lor (c \land d))$ is implemented as $((\overline{a} \land b) \lor (c \land d))$.

When $k_i = 1$, the fault effectively becomes a Term Negation Fault. For example, the boolean specification $(a \lor (c \land d))$ is implemented as $(a \lor (c \land d))$, which is similar to negating the first term.

### 3.2.4 Term Omission Fault (TOF)

In a Term Omission Fault, a term in the boolean specification is omitted to get the faulty specification. Let $S$ be a boolean specification as given in Equation (3.1) and $TOF$ denote its faulty implementation. The general expression for $TOF$ is given below.

$$TOF = P_1 \lor P_2 \lor \ldots \lor P_{i-1} \lor P_{i+1} \lor \ldots \lor P_m$$

(3.6)

where the $i^{th}$ term $P_i$ alone is omitted. $P_1, P_2, \ldots, P_m$ refers to the terms in $S$. $m$ is the number of terms in $S$ and $m > 1$. The number of terms in $TOF$ is $m-1$. For example, the boolean specification $((a \land b) \lor (c \land d))$ is implemented as $(c \land d)$. When $m = 1$, it indicates that the specification is not implemented at all.

### 3.2.5 Literal Reference Fault (LRF)

In a Literal Reference Fault, a literal from a term in the specification is replaced by another literal from the same specification which satisfy certain conditions. Let the boolean expression $S$ denote a specification in DNF. The general expression for $S$ is given in Equation (3.1). The expression for $S(P_i)$ where $P_i$ is the $i^{th}$ term of the specification is given as Equation (3.4). The expression for the term $LRF(P_i)$ is:
\[ LRF(P_i) = x_{i1} \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land x_{i(k_i-1)} \land x_i \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land x_{i_{k_i}} \]  

(3.7)

where the \( j^{th} \) literal in the \( i^{th} \) term alone is replaced by another literal \( x_t \). \( x_{i1}, x_{i2}, \ldots, x_{i_{k_i}} \) are the literals in the term \( P_i \) and \( k_i \) (\( k_i \geq 1 \)) is the number of literals in the \( i^{th} \) term \( P_i \). For example, the boolean expression \( ((a \land b) \lor (c \land d)) \) is implemented as \( ((c \land b) \lor (c \land d)) \).

For Literal Reference Fault, the \( j^{th} \) literal, \( x_{ij} \) (\( j = 1, \ldots, k_i \)), of the \( i^{th} \) term, \( P_i \) (\( i = 1, \ldots, m \)), in \( S \) is replaced by the literal \( x_t \) whose normal form or its negated form does not originally appear in \( P_i \). For LRF, the following replacement of a literal are not considered.

When the literal replaced for Literal Reference Fault is already present in the term, the faulty specification effectively becomes Literal Omission Fault. For example, consider the boolean expression \( ((a \land b) \lor (c \land d)) \). If the literal \( a \) is replaced by the literal \( b \) which is already present in the same term, the new expression thus obtained is \( ((b \land b) \lor (c \land d)) \), which is similar to omitting the literal \( a \) in the first term to get \( LOF \).

If a literal is replaced by its negation, it effectively becomes Literal Negation Fault. For example, consider the boolean expression \( ((a \land b) \lor (c \land d)) \). If the literal \( a \) is replaced by the literal \( \overline{a} \), the new expression thus obtained is \( ((\overline{a} \land b) \lor (c \land d)) \), which is similar to negating the literal \( a \) in the first term to get \( LNF \).

If a literal is replaced by the negation of another literal present in the same term, it becomes a Term Omission Fault. For example, consider the boolean expression \( ((a \land b) \lor (c \land d)) \). If the literal \( a \) is replaced by the literal \( \overline{b} \), the new expression thus obtained is \( ((\overline{b} \land b) \lor (c \land d)) \), which is similar to omitting the first
term in the specification to get TOF. Further, $LRF(P_i)$ should not be equal to any of the terms in the original specification.

### 3.2.6 Literal Omission Fault (LOF)

In a Literal Omission Fault, a literal from a term of the specification is omitted to get the faulty specification. Let the boolean expression $S$ denote a specification in DNF. The general expression for $S$ is given in Equation (3.1). The expression for $S(P_i)$ where $P_i$ is the $i^{th}$ term of the specification is given in Equation (3.4). The expression for the term $LOF(P_i)$ is given below.

$$LOF(P_i) = x_{i1} \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land x_{i(j-1)} \land x_{i(j+1)} \land \ldots \land x_{ik}\quad (3.8)$$

where the $j^{th}$ literal in the $i^{th}$ term alone is omitted. $x_{i1}, x_{i2}, \ldots, x_{ik}$ are the literals in the term $P_i$ and $k_i \ (k_i > 1)$ is the number of literals in the term $P_i$. For example, the boolean specification $((a \land b) \lor (c \land d))$ is implemented as $(c \land d)$, which is same as omitting the first term to obtain Term Omission Fault.

### 3.2.7 Literal Insertion Fault (LIF)

In a Literal Insertion Fault, a literal is inserted in a term of the boolean specification to get the faulty specification. Let $S$ be a boolean specification as given in Equation (3.1) and $LIF$ denote its faulty implementation. The term $LIF(P_i)$ is:

$$LIF(P_i) = x_{i1} \land x_{i2} \land x_{i3} \land x_{i4} \land \ldots \land x_{ik} \land x_k\quad (3.9)$$
where $P_i$ is the $i^{th}$ term in which the literal $x_t$ is inserted. $x_{t1}, x_{t2}, \ldots, x_{tk}$ are the literals in the term $P_i$ and $k_t (k_t \geq 1)$ is the number of literals in the term $P_i$. For example, the boolean specification $((a \land b) \lor (c \land d))$ is implemented as $((a \land b \land c) \lor (c \land d))$.

There is a Literal Insertion Fault in the $i^{th}$ term $P_i$ in $S$ with the literal $x_t$ whose normal form or its negated form does not originally appear in $P_i$. For LIF, the following insertion of a literal in a term is not considered.

If the literal inserted in Literal Insertion Fault is the same literal that is already present in the term, then the faulty specification effectively becomes the original specification. For example, consider the boolean expression $((a \land b) \lor (c \land d))$. If the literal $a$ is inserted in the first term, the new expression thus obtained is $((a \land a \land b) \lor (c \land d))$, which is same as the original specification.

If the literal inserted in Literal Insertion Fault is the negation of a literal that is already present in the term, then the faulty specification effectively results in Term Omission Fault. For example, consider the boolean expression $((a \land b) \lor (c \land d))$. If the literal $\overline{a}$ is inserted in the first term, the new expression thus obtained is $(\overline{a} \land a \land b) \lor (c \land d))$, which results in Term Omission Fault in which the first term is omitted.

3.2.8 Term Insertion Fault (TIF)

In a Term Insertion Fault, a term is inserted in the boolean specification such that it satisfies certain conditions. Let $S$ be a boolean specification as given in
Equation (3.1) and TIF denote its faulty implementation. The general expression for TIF is given below.

\[ TIF = P_1 \lor P_2 \lor \ldots \lor P_i \lor \ldots \lor P_m \lor P_t \] (3.10)

where \( P_1, P_2, \ldots, P_m \) refers to the terms in \( S \). \( P_t \) is the term inserted in \( S \) to get TIF. \( m \) is the number of terms in \( S \) and \( m > 1 \). The number of terms in TIF is \( m+1 \). The term \( P_t \) satisfies the following condition.

\[ P_t = LRF(P_i) \]

Here, \( P_i \) is the term in which the Literal Reference Fault occurs to get \( P_t \), which is the term inserted in \( S \) to get TIF. The term \( P_t \) does not exist in the original specification. For example, the boolean specification \(((a \land b) \lor (c \land d))\) is implemented as \(((a \land b) \lor (c \land d) \lor (c \land b))\).

### 3.3 EXPERIMENTAL STUDY

For this study, many specifications have been taken for analysis from various sources including the Traffic Alert and Collision Avoidance Subsystem (TCAS II) specifications given by Weyuker et al (1994). The specifications varied in size from 5 to 14 literals. They were then converted into DNF.

Various faults were created for those specifications as described in the fault classes. Each literal was considered for generating the faults. For example, in Literal Reference Fault, each and every literal in the specification in DNF was replaced by all the possible literals that satisfy the fault class. For example, in a 14 literal specification given by Weyuker et al (1994), a total of 1793 mutants were created, for a 11 literal specification, a total of 849 mutants were created, for a 8 literal specification, a total of 313 mutants were created etc. The specification was tested with all possible
combinations of the input. For example, if the specification consists of 8 distinct literals, $2^8$ combinations are possible. The specification was tested for all these combinations.

All the possible input combinations were stored in a database and for each fault that occurs on a literal, a table was maintained. On an average, each table had 154 boolean fields. More than 500 tables were created for the complete analysis of various specifications. Test sets were generated to detect various faults using the fault condition given by Kuhn (1999). The test cases were applied to the faulty implementations to check if the fault was detected. The test cases were analyzed and results were derived.

### 3.4 TEST SET REGION

A test set region is the region in the venn diagram where the test cases to detect a particular fault is present. Let $S$ be a boolean specification and let $S'$ be its faulty implementation. If a test causes $S \oplus S'$ to evaluate to true, then the fault will be detected. Test cases to detect a fault can be obtained by computing $S \oplus S'$ (Kuhn 1999, Tsuchiya et al 2002).

The work of Kuhn (1999) was analyzed for various faults mentioned in Section 3.2. The detection condition proposed by him could be used to detect all the faults. The work of Kuhn (1999) was further analyzed using a venn diagram. The following venn diagram depicts the test set region.
In the above venn diagram, $S$ represents the set of true points for which the specification evaluates to True and $S'$ represents the set of true points for which the faulty specification evaluates to True. The shaded region in the venn diagram shows the region where test cases can be found. It is the region where the output of the original specification differs from the output of the faulty specification for the same input. So, Kuhn (1999) proposed that to detect any fault, test cases should be chosen from the shaded region.

Kuhn (1999) and Tsuchiya et al (2002) have mentioned a generalized fault condition for various faults such as Missing Condition Fault, Variable Reference Fault, Variable Negation Fault etc. Although the fault defined by Chen et al (1997) and Lau et al (2001) are different from those defined by Kuhn (1999), the fault condition proposed by Kuhn (1999) was applied to the faults defined by Chen et al (1997) and Lau et al (2001). Let $m$ be the number of terms in the specification in DNF and let $k_i$ be the number of literals in the $i^{th}$ term $P_i$ of $S$. The fault condition for various faults are defined below.

\[ S_{\text{enf}} = S \oplus \overline{S} \text{ where } S \text{ is the boolean specification and } m > 1. \]

\[ S_{\text{tnf}} = S \oplus S_{\overline{P_i}} \text{ where } P_i \text{ is the term being negated and } m \geq 1. \]
$SLNF = S \oplus S_{x_i}^{x_i}$ where the $j^{th}$ literal $x_{ij}$ in the $i^{th}$ term $P_i$ is negated, $k_i > 1$ and $m \geq 1$.

$STOF = S \oplus S_{x_i}^{x_i}$ where $P_i$ is the term excluded and $m \geq 1$.

$SLRF = S \oplus S_{x_i}^{x_i}$ where $x_{ij}$ is the literal replaced by $x_i (x_i \neq x_{ij})$ and $k_i \geq 1$ and $m \geq 1$.

$SLOF = S \oplus S_{x_i}^{x_i}$ where $x_{ij}$ is the literal excluded and $k_i \geq 1$ and $m \geq 1$.

$SLIF = S \oplus S_{x_i}^{x_i}$ where $x_i$ is the literal inserted in the $i^{th}$ term $P_i$ and $k_i \geq 1$ and $m \geq 1$.

$STIF = S \oplus S_{x_i}^{x_i}$ where $P_i$ is the term inserted in $S, P_i = LRF(P_i)$ and $m \geq 1$.

While analyzing the fault condition for various faults, it was observed that certain faults generated test cases that satisfied a particular condition. Those conditions were analyzed and the results are given below.

3.5 RESULTS

The results obtained have been formulated as theorems and the analytical proof of the test results are given.

**Theorem 3.1:** A test case $x$ will detect a Literal Omission Fault if and only if $x \notin S$ and $x \in LOF$ in the fault condition $SLOF$.

**Proof:** Let $S$ be a boolean specification in DNF. Let the literal $a_i$ in the term $P_i$ be omitted in $S$ to get $LOF$. The general expression for $S$ and the expression for $LOF$ in the above situation are given below.

\[
S = (a_1 \land a_2 \land \ldots \land a_l) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_p)
\]

\[
LOF = (a_1 \land a_2 \land \ldots \land a_l) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_p)
\]
The theorem can be proved by the method of contradiction. Let there exist a test case \( x \) such that \( x \in S \) and \( x \notin \text{LOF} \). From the above premises, it is clear that \( x \in (S \land \overline{\text{LOF}}) \). From Equation (3.11) and Equation (3.12), we get that

\[
S \land \overline{\text{LOF}} = ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_j))
\]

\[
= ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_j))
\]

\[
= \text{Null}
\]

Thus, it is observed that the point \( x \notin (S \land \overline{\text{LOF}}) \). Hence by the method of contradiction, it is found that a test case to detect Literal Omission Fault will be obtained if and only if \( x \notin S \) and \( x \in \text{LOF} \). From the above proof, it is clear that the expression \( S \land \overline{\text{LOF}} \) is always Null. This is further illustrated by a venn diagram.

![Venn Diagram](image-url)

**Figure 3.2 Test set region of SLOF**

Here, \( S \) is the set of true points for which the specification evaluates to True and \( \text{LOF} \) is the set of true points for which the faulty implementation of \( S \) evaluates to True. The fault condition \( SLOF \) is applied and the test set region is indicated in the above venn diagram. The region B in Figure 3.2 is always Null in the fault condition \( SLOF \).
Theorem 3.2: A test case $x$ will detect a Term Omission Fault if and only if $x \in S$ and $x \not\in \text{TOF}$ in the fault condition $\text{STOF}$.

Proof: Let $S$ be a boolean specification as given in Equation (3.11). Let the term $P_i$ in $S$ be omitted to get $\text{TOF}$. The expression for $\text{TOF}$ is given below.

$$\text{TOF} = (b_1 \land b_2 \land \ldots \land b_d) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_d)$$ (3.13)

The theorem can be proved by the method of contradiction. Let there exist a test case $x$ such that $x \not\in S$ and $x \in \text{TOF}$. From the above premises, it is clear that $x \in (S' \land \text{TOF})$. From Equation (3.11) and (3.13), we get that

$$S \land \text{TOF} = ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_d) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_d)) \land$$

$$((b_1 \land b_2 \land \ldots \land b_d) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_d))$$

$$= ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_d) \land \ldots \land (z_1 \land z_2 \land \ldots \land z_d)) \land$$

$$((b_1 \land b_2 \land \ldots \land b_d) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_d))$$

$$= \text{Null}$$

Thus, it is observed that the point $x \not\in (S \land \text{TOF})$. Hence by the method of contradiction, it is found that a test case to detect Term Omission Fault will be obtained if and only if $x \in S$ and $x \not\in \text{TOF}$. From the above proof, it is clear that the expression $\overline{S} \land \text{TOF}$ is always Null. This is further illustrated by a venn diagram.
Here, $S$ is the set of true points for which the specification evaluates to True and $TOF$ is the set of true points for which the faulty implementation of $S$ evaluates to True. The fault condition $Stof$ is applied and the test set region is indicated in the above venn diagram. The region B in Figure 3.3 is always Null in the fault condition $Stof$.

**Theorem 3.3:** A test case $x$ will detect a Literal Insertion Fault if and only if $x \in S$ and $x \not\in LIF$ in the fault condition $Slif$.

**Proof:** Let $S$ be a boolean specification as given in Equation (3.11). Let the literal $b_1$ be inserted in the term $P_1$ in $S$ to get $LIF$. The expression for $LIF$ is given below.

$$LIF = (b_1 \land a_1 \land a_2 \land \ldots \land a_q) \lor (b_1 \land b_2 \land \ldots \land b_q) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)$$

(3.14)

The theorem can be proved by the method of contradiction. Let there exist a test case $x$ such that $x \not\in S$ and $x \in LIF$. From the above premises, it is clear that $x \in (\bar{S} \land LIF)$. From Equation (3.11) and Equation (3.14), we get that
Thus, it is observed that the point $x \not\in (S \land LIF)$. Hence by the method of contradiction, it is found that a test case to detect Literal Insertion Fault will be obtained if and only if $x \in S$ and $x \not\in LIF$. From the above proof, it is clear that the expression $\bar{S} \land LIF$ is always Null. This is further illustrated by a venn diagram.

![Venn Diagram](image_url)

Figure 3.4 Test set region of $SLIF$

Here, $S$ is the set of true points for which the specification evaluates to True and $LIF$ is the set of true points for which the faulty implementation of $S$ evaluates to True. The fault condition $SLIF$ is applied and the test set region is indicated in the above venn diagram. The region B in Figure 3.4 is always Null in the fault condition $SLIF$.

**Theorem 3.4:** A test case $x$ will detect a Term Insertion Fault if and only if $x \not\in S$ and $x \in TIF$ in the fault condition $STIF$. 
Proof: Let $S$ be a boolean specification as given in Equation (3.11) Let the Term $P_i$ be inserted in $S$ to get $TIF$. The expression for $TIF$ is given below.

$$TIF = (b_1 \land a_2 \land \ldots \land a_k) \lor (a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_n) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)$$

(3.15)

The theorem can be proved by the method of contradiction. Let us assume there exist a test case $x$ such that $x \in S$ and $x \notin TIF$. From the above premises, it is clear that $x \in (S \land \overline{TIF})$. From Equation (3.11) and Equation (3.15), we get

$$S \land \overline{TIF} = ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_n) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)) \land$$

$$((b_1 \land a_2 \land \ldots \land a_k) \lor (a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_n) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)) \land$$

$$((b_1 \land a_2 \land \ldots \land a_k) \land (a_1 \land a_2 \land \ldots \land a_k) \land (b_1 \land b_2 \land \ldots \land b_n) \land \ldots \land (z_1 \land z_2 \land \ldots \land z_y))$$

= Null

Thus, it is observed that the point $x \notin (S \land \overline{TIF})$. Hence by the method of contradiction, it is found that a test case to detect Term Insertion Fault will be obtained if and only if $x \notin S$ and $x \in TIF$. From the above proof, it is clear that the expression $S \land \overline{TIF}$ is always Null. This is further illustrated by a venn diagram.

Figure 3.5 Test set region of $S_{TIF}$
Here, $S$ is the set of true points for which the specification evaluates to True and $TIF$ is the set of true points for which the faulty implementation of $S$ evaluates to True. The fault condition $S_{TIF}$ is applied and the test set region is indicated in the above venn diagram. The region B in Figure 3.5 is always Null in the fault condition $S_{TIF}$.

**Fault condition for Literal Reference Fault:** Let $S$ be a boolean specification as given in Equation (3.11) Let the literal $a_i$ in the term $P_1$ in $S$ be replaced by the literal $b_j$. The expression for $LRF$ is given below.

$$LRF = (b_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_j)$$  \hspace{1cm} (3.16)

The fault condition for Literal Reference Fault is given below.

$$S_{LRF} = S \oplus LRF$$

$$= ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_j)) \oplus$$

$$((b_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_j))$$

$$= (((a_1 \lor \bar{b}_i) \lor (\bar{a}_1 \lor b_i)) \land a_2 \land \ldots \land a_k) \land (\bar{b}_i \lor \bar{b}_j \lor \ldots \lor \bar{b}_k)$$

$$\land \ldots \land (\bar{z}_1 \lor \bar{z}_2 \lor \ldots \lor \bar{z}_j)$$  \hspace{1cm} (3.17)

Thus it is observed that a test set for Literal Reference Fault can be derived using the above expression. Test cases to detect the Literal Reference Fault are available in the shaded area of the venn diagram shown below.
Here, $S$ is the set of true points for which the specification evaluates to True and $LRF$ is the set of true points for which the faulty implementation of $S$ evaluates to True. The fault condition $SLRF$ is applied and the test set region is indicated in the above venn diagram.

Fault condition for Literal Negation Fault: Let $S$ be a boolean specification as given in Equation (3.11) Let the literal $a_i$ in the term $P_1$ in $S$ be negated to get Literal Negation Fault. The expression for $LNF$ is given below.

$$LNF = (\overline{a_1} \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)$$  \hspace{1cm} (3.18)

The fault condition for Literal Negation Fault is given below.

$$SLNF = S \oplus LNF$$

$$= ((\overline{a_1} \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y)) \oplus$$

$$((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_y))$$

$$= (a_2 \land \ldots \land a_k) \land (\overline{b_1} \lor \overline{b_2} \lor \ldots \lor \overline{b_m}) \land \ldots \land (\overline{z_1} \lor \overline{z_2} \lor \ldots \lor \overline{z_y})$$  \hspace{1cm} (3.19)

Thus it is observed that a test set for Literal Negation Fault can be derived using the above expression. Test cases to detect the Literal Negation Fault are available in the shaded area of the venn diagram shown below.
Here, $S$ is the set of true points for which the specification evaluates to True and $LNF$ is the set of true points for which the faulty implementation of $S$ evaluates to True. The fault condition $S_{lnf}$ is applied and the test set region is indicated in the above venn diagram.

**Fault condition for Term Negation Fault:** Let $S$ be a boolean specification as given in Equation (3.11) Let the term $P_j$ in $S$ be negated to get Term Negation Fault. The expression for $TNF$ is given below.

$$TNF = (a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_l)$$ \hspace{1cm} (3.20)

The fault condition for Term Negation Fault is given below.

$$S_{TNF} = S \oplus TNF$$

$$= ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_l)) \oplus$$

$$= ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_m) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_l))$$

$$= (\bar{a}_1 \lor \bar{a}_2 \lor \ldots \lor \bar{a}_k) \land \ldots \land (\bar{z}_1 \lor \bar{z}_2 \lor \ldots \lor \bar{z}_l)$$ \hspace{1cm} (3.21)
Thus, it is observed that a test set for Term Negation Fault can be derived using the above expression. Test cases to detect the Term Negation Fault are available in the shaded area of the Venn diagram shown below.

![Venn diagram](image)

**Figure 3.8 Test set region of \( S_{\text{TNF}} \)**

Here, \( S \) is the set of true points for which the specification evaluates to True and \( TNF \) is the set of true points for which the faulty implementation of \( S \) evaluates to True. The fault condition \( S_{\text{TNF}} \) is applied and the test set region is indicated in the above Venn diagram.

**Fault condition for Expression Negation Fault:** Although, Lau et al (2001) has defined the fault, no analysis is provided. Here, Expression Negation Fault is analyzed and the test set region to detect Expression Negation Fault is provided.

Let \( S \) be a boolean specification as given in Equation (3.11) Let the specification \( S \) be negated to get Expression Negation Fault. The expression for \( ENF \) is given below.

\[
ENF = ((a_1 \land a_2 \land \ldots \land a_k) \lor (b_1 \land b_2 \land \ldots \land b_k) \lor \ldots \lor (z_1 \land z_2 \land \ldots \land z_n)) \quad (3.22)
\]
The fault condition for Expression Negation Fault is given below.

\[ S_{ENF} = S \oplus ENF \]
\[ = \left( (a_1 \land a_2 \land \cdots \land a_k) \lor (b_1 \land b_2 \land \cdots \land b_n) \lor \cdots \lor (z_1 \land z_2 \land \cdots \land z_m) \right) \]
\[ \oplus \]
\[ = T \]

Thus it is observed that a test set for Expression Negation Fault can be derived using the above fault condition. Test cases to detect the Expression Negation Fault are available in the shaded area of the venn diagram shown below.

![Venn Diagram](image)

**Figure 3.9 Test set region of S_{ENF}**

Here, \( S \) is the set of true points for which the specification evaluates to True and \( ENF \) is the set of true points for which the faulty implementation of \( S \) evaluates to True. The fault condition \( S_{ENF} \) is applied and the test set region is indicated in the above venn diagram.

### 3.6 APPLICATIONS

The above theorems have been applied to real life Traffic Alert and Collision Avoidance Subsystem (TCAS II) specifications given by Weyuker et al (1994). The results are given below.
Example for Theorem 3.1: For a realistic example, consider the following specification from Weyuker et al (1994).

\[ S = (ac + bd)e(fg + fh) \]  

(3.23)

The specification \( S \) in DNF is given below.

\[ S = ((a \wedge c \wedge e \wedge f \wedge g) \vee (b \wedge d \wedge e \wedge f \wedge g) \vee (a \wedge c \wedge e \wedge \overline{f} \wedge h) \vee (b \wedge d \wedge e \wedge \overline{f} \wedge h)) \]  

(3.24)

Let a literal say \( e \) be missed in the term \( P_1 = (a \wedge c \wedge e \wedge f \wedge g) \) in \( S \) to get \( LOF \). The following table gives the test set for \( S \wedge LOF \) and \( SLOF \).

**Table 3.1 Test set for Literal Omission Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \wedge LOF )</td>
<td>166, 167, 182, 183, 230, 231, 246, 247</td>
</tr>
<tr>
<td>( SLOF )</td>
<td>166, 167, 182, 183, 230, 231, 246, 247</td>
</tr>
</tbody>
</table>

The table shows that the test set for \( SLOF \) and \( S \wedge LOF \) are equal. Thus, it is clear that a test case \( x \) will detect a Literal Omission Fault if and only if \( x \in S \) and \( x \in LOF \) in the fault condition \( SLOF \).

Example for Theorem 3.2: For a realistic example, consider the following specification from Weyuker et al (1994).

\[ S = ((ac + bd)e(f + (ig + hk))) \]  

(3.25)
The specification $S$ in DNF is given below.

$$S = (a \land c \land e \land f) \lor (a \land c \land e \land g \land i \land j) \lor (a \land c \land e \land h \land i \land k) \lor (b \land d \land e \land f) \lor (b \land d \land e \land g \land i \land j) \lor (b \land d \land e \land h \land i \land k)$$  

(3.26)

Let the term $P_i = (a \land c \land e \land f)$ in $S$ be omitted to get $TOF$. The following table gives the test set for $S \land TOF$ and $STOF$.

**Table 3.2 Test set for Term Omission Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \land TOF$</td>
<td>1376, 1377, 1378, 1379, 1380, 1381, 1382, ...</td>
</tr>
<tr>
<td>$STOF$</td>
<td>1376, 1377, 1378, 1379, 1380, 1381, 1382, ...</td>
</tr>
</tbody>
</table>

The table shows that the test set for $STOF$ and $S \land TOF$ are equal. Thus, it is clear that a test case $x$ will detect a Term Omission Fault if and only if $x \in S$ and $x \not\in TOF$ in the fault condition $STOF$.

**Example for Theorem 3.3:** For a realistic example, consider the following specification from Weyuker et al (1994).

$$S = (\overline{a \overline{b} \overline{c} \overline{f} (g + \overline{g}) (e \overline{n} + d) + \overline{n} (j \overline{k} + \overline{j} \overline{m}))$$  

(3.27)

The specification $S$ in DNF is given below.

$$S = ((a \land b \land c \land j \land l \land \overline{m} \land \overline{n}) \lor (a \land b \land c \land \overline{e} \land f \land h \land n) \lor (a \land b \land c \land \overline{e} \land f \land g \land \overline{n}) \lor (a \land b \land c \land \overline{e} \land f \land i \land \overline{n}) \lor (a \land b \land c \land d \land f \land n) \lor (a \land b \land c \land d \land f \land h) \lor (a \land b \land \overline{c} \land d \land f \land g) \lor (a \land b \land \overline{c} \land j \land k \land \overline{n}))$$  

(3.28)
Let a literal say $e$ be inserted in the term $P_j = (a \land b \land c \land d \land f \land g)$ in $S$ to get $\text{LIF}$. The following table gives the test set for $S \land \text{LIF}$ and $\text{SLIF}$.

**Table 3.3 Test set for Literal Insertion Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \land \text{LIF}$</td>
<td>9601, 9603, 9605, 9607, 9609, 9611, ...</td>
</tr>
<tr>
<td>$\text{SLIF}$</td>
<td>9601, 9603, 9605, 9607, 9609, 9611, ...</td>
</tr>
</tbody>
</table>

The table shows that the test set for $\text{SLIF}$ and $S \land \text{LIF}$ are equal. Thus, it is clear that a test case $x$ will detect a Literal Insertion Fault if and only if $x \in S$ and $x \notin \text{LIF}$ in the fault condition $\text{SLIF}$.

**Example for Theorem 3.4:** Let $S$ be a boolean specification as given in Equation (3.24). Let the term $P_z = (b \land c \land e \land f \land g)$, obtained by replacing the literal $a$ by $b$ in the term $P_j = (a \land c \land e \land f \land g)$ be inserted in $S$ to get $\text{TIF}$. The following table gives the test set for $S \land \text{TIF}$ and $\text{STIF}$.

**Table 3.4 Test set for Term Insertion Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \land \text{TIF}$</td>
<td>110, 111</td>
</tr>
<tr>
<td>$\text{STIF}$</td>
<td>110, 111</td>
</tr>
</tbody>
</table>

The table shows that the test set for $\text{STIF}$ and $S \land \text{TIF}$ are equal. Thus, it is clear that a test case $x$ will detect a Term Insertion Fault if and only if $x \notin S$ and $x \in \text{TIF}$ in the fault condition $\text{STIF}$.

**Fault condition for Literal Reference Fault - An Example:** Let $S$ be a boolean specification as given in Equation (3.26). Let the literal $a$ in the term
\( P_i = (a \land c \land e \land g \land i \land j) \) in \( S \) be replaced by the literal \( b \) to get \( LRF \). The following table gives the test set to detect the Literal Reference Fault in this case.

**Table 3.5 Test set for Literal Reference Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SLRF )</td>
<td>854, 855, 862, 863, 886, 887, 894, 895, 1366, 1367, 1374, 1494, 1495, 1502</td>
</tr>
</tbody>
</table>

The table shows the test set generated to detect Literal Reference Fault for the specification given in Equation (3.26).

**Fault condition for Literal Negation Fault - An Example:** Let \( S \) be a boolean specification as given in Equation (3.28). Let the literal \( a \) in the term \( P_i = (a \land b \land c \land j \land k \land \overline{n}) \) in \( S \) be negated to get \( LNF \). The following table gives the test set to detect the Literal Negation Fault in this case.

**Table 3.6 Test set for Literal Negation Fault**

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SLNF )</td>
<td>24, 26, 28, 30, 56, 58, 60, 62, 88, 90, 92, 94, 120, 122, 124, 126, 152, 154, ...</td>
</tr>
</tbody>
</table>

The table shows the test set generated to detect Literal Negation Fault for the specification given in Equation (3.28).

**Fault condition for Term Negation Fault - An Example:** Let \( S \) be a boolean specification as given in Equation (3.24). Let the term \( P_i = (a \land c \land e \land \overline{f} \land h) \) in \( S \) be negated to get \( TNF \). The following table gives the test set to detect the Term Negation Fault in this case.
Table 3.7 Test set for Term Negation Fault

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_tnf</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ...</td>
</tr>
</tbody>
</table>

The table shows the test set generated to detect Term Negation Fault for the specification given in Equation (3.24).

Fault condition for Expression Negation Fault – An Example: Let S be a boolean specification as given in Equation (3.26). Let the specification S be negated to get ENF. The following table gives the test set to detect the Expression Negation Fault in this case.

Table 3.8 Test set for Expression Negation Fault

<table>
<thead>
<tr>
<th>Fault condition</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_enf</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ...</td>
</tr>
</tbody>
</table>

The table shows the test set generated to detect Expression Negation Fault for the specification given in Equation (3.26).

3.7 SIGNIFICANCE OF THE RESULTS

The generalized fault condition given by Kuhn (1999) is applied to different fault classes to generate test cases to detect various faults. In this chapter, computationally efficient simplified fault condition for some faults has been presented. It has been shown that it suffices to compute the simplified fault condition for generating test sets for various faults rather than computing the fault condition proposed by Kuhn (1999) and Tsuchiya et al (2002). Their method always results in
Null for a part of the test set region for faults such as Literal Insertion Fault, Term Insertion Fault, Literal Omission Fault and Term Omission Fault. The results of the chapter are summarized below:

### Table 3.9 Simplified Fault Condition for various faults

<table>
<thead>
<tr>
<th>Fault</th>
<th>Kuhn’s method</th>
<th>Simplified Fault Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal Omission Fault</td>
<td>$SLOF$</td>
<td>$\overline{S} \wedge LOF$</td>
</tr>
<tr>
<td>Term Omission Fault</td>
<td>$STOF$</td>
<td>$S \wedge TOF$</td>
</tr>
<tr>
<td>Term Insertion Fault</td>
<td>$STIF$</td>
<td>$S \wedge TIF$</td>
</tr>
<tr>
<td>Literal Insertion Fault</td>
<td>$SLIF$</td>
<td>$S \wedge LIF$</td>
</tr>
<tr>
<td>Literal Reference Fault</td>
<td>$SLRF$</td>
<td>-</td>
</tr>
<tr>
<td>Literal Negation Fault</td>
<td>$SLNF$</td>
<td>-</td>
</tr>
<tr>
<td>Term Negation Fault</td>
<td>$STNF$</td>
<td>-</td>
</tr>
<tr>
<td>Expression Negation Fault</td>
<td>$SENF$</td>
<td>-</td>
</tr>
</tbody>
</table>

In the next chapter, test set generation from a single fault class namely Literal Reference Fault is explained. Also, the results proposed by Lau et al (2001) are analyzed empirically and the hierarchy of fault classes proposed by Lau et al (2001) is extended by adding the new fault class namely, Term Insertion Fault.