A Detectability Analysis of Fault Classes for Boolean Specifications∗

Zhenyu Chen, Baowen Xu, Changhai Nie
School of Computer Science and Engineering
Southeast University, Nanjing, China
{zychen, bwxu, changhainie}@seu.edu.cn

ABSTRACT
The detectability hierarchies of fault classes for specification-based testing have been established to prioritize test cases so as to achieve earlier detection of more faults. This paper extends and complements the existing studies by analyzing detectability of fault classes for Boolean specifications in general form. The monotonicity of detectability is discovered that a fault occurs at literal is more difficult to detect than the corresponding one occurs at other positions. Furthermore, a strong detectability relationship is introduced to overcome the flaw of traditional approach. It can help identify stronger faults and skip weaker faults during testing. As a result, two detectability hierarchies are established on the detection conditions of fault classes.

Keywords
specification-based testing, fault class, detection condition

1. INTRODUCTION
Fault-based testing is first to hypothesize certain common types of faults that may be committed by the programmers, and then automatically generate test cases that can detect these faults. Fault-based testing can guarantee the absence of certain faults in software. It is an important advantage over other testing methods. Mutation analysis is a commonly accepted technique of fault-based testing. Mutation analysis typically considers the faults that cause small changes to the correct version, which is based on the competent programmer hypothesis [4].

There is a close relationship between fault class analysis and mutation analysis, because the mutation operators are often designed to model the common faults. Various fault classes based on mutation analysis have been defined and studied theoretically and empirically in [5] [12]. The key fault classes are identified to help prioritize test cases to be executed so as to achieve earlier detection of more faults. Kuhn [7] discovered the relationships among the conditions that detect three types of faults. It establishes a hierarchy that implies that some faults may be skipped during testing. Lau and Yu [8] extended Kuhn’s hierarchy by analyzing the relationships between variable faults and literal faults. Their analysis can be applied to the design and evaluation of test methods. It informs the way that test cases should be prioritized for earlier detection of faults.

The existing efforts studied Boolean specifications in redundant disjunctive normal form (IDNF) [7] [8] [11]. However, Boolean expressions in a realistic program or specification are often not in the restricted form, but the faults are introduced in general form [3]. Moreover, the DNF transformation and minimization is an NP-hard problem [6]. Okun et al. [9] removed the restriction of IDNF and presented a more general analysis of literal faults w.r.t. negation, reference and insertion (conjunction or disjunction) using the RELAY model [10].

In this paper we present a more comprehensive analysis and produce richer results to identify the fault hierarchies for Boolean specifications in general form. We first extend the earlier work [9] to describe the detection conditions of fault classes and establish a detectability hierarchy based on the conditions. To reason about more detectability relationships between operator faults and operand faults, a syntax tree of Boolean expression is introduced. The monotonicity of detectability of fault classes is shown by the syntax tree. This implies that a fault occurring at the leaf is more difficult to detect than the corresponding one at other positions of syntax tree.

The detectability hierarchies described above [7] [8] [9] is called a weak detectability relationship in this paper. A limitation of weak detectability relationship comes from the difference between mutant and fault. A mutant may happen to be logically equivalent to the original Boolean specification and an equivalent mutant is not regarded as a fault. If a mutant in a stronger fault class is equivalent and a corresponding mutant in a weaker fault class is not equivalent, then it could not guarantee the detection of the weaker fault.
Hence, it could not skip weaker fault classes during testing based on the weak detectability relationship. A strong detectability relationship is introduced to overcome this problem of the weak one. It implies that detection of all faults in a stronger fault class guarantees the detection of all faults in a weaker fault class.

This paper is organized as follows. In the next section, we introduce logic notations and the fault classes of Boolean specifications. The detection conditions of various fault classes are represented and derived from a Boolean difference model in section 3. Section 4 presents a monotonicity analysis of fault classes and section 5 gives a further detectability analysis. The conclusion is drawn in the last section.

2. THEORETICAL FOUNDATION

2.1 Logic Notation

A Boolean expression is a string which involves some variables and logical operators \( \neg \) (NOT), \( \cdotp \) (AND) and \(+\) (OR). \( \cdotp \) is omitted if it is clear in the context. An assignment \( t \) of a Boolean expression \( S \) is a mapping from the variables to \( \{0, 1\} \), i.e. \{true, false\}. An assignment \( t \) satisfies \( S \) if it evaluates \( S \) to \textit{true}. \( S \) is said to be satisfiable if there is at least one satisfying assignment. For two Boolean expressions \( S \) and \( S' \), if each satisfying assignment of \( S \) is also a satisfying assignment of \( S' \), then \( S \) implies \( S' \), denoted by \( S \models S' \). \( S \) is equivalent to \( S' \), denoted by \( S \equiv S' \), if and only if \( S \models S' \) and \( S' \models S \).

A syntax tree \( T \) of \( S \) is a finite binary tree. The intermediate nodes of \( T \) are the logical connectives \( \cdotp \) or \(+\) and the branches may bind the negation operator \( \neg \). The leaves of \( T \) are the literals, i.e. positive variables or negative variables. For clearness, a unique index is given for each node, because the different nodes may have the same label. For example, the syntax tree of \( S_0 = (a+b)\cdotp \cdotp + (a+d) \) is shown in Figure 1. \( S(i) \) denotes the node of index \( i \). \( S[i] \) denotes the sub-expression, i.e. sub-tree, of index \( i \). \( S[i/X] \) denotes the resulting expression by replacing sub-expression \( S[i] \) with \( X \). For example, \( S_0[2] = (a+d) \); \( S_0[3] = a+b \); \( S_0[3/(b+d)] = (b+d)\cdotp \cdotp + (a+d) \). \( ch1(i) \) and \( ch2(i) \) denote two immediate child indexes of \( i \) in \( T \).

A test case \( t \) of Boolean expression \( S \) is a complete assignment of \( S \). A test set \( T \) is a set of test cases. In the following sections, the theoretical results of Boolean specification are established on the syntax tree.

2.2 Fault Classes of Boolean Specifications

Various fault classes of Boolean specifications have been defined and studied in [7] [8] [9]. The most common faults could be modeled by mutation. That is, the faulty is the version differs from the original Boolean expression by exactly one syntactic change. In this paper we introduce the fault classes classified as operator faults and operand faults. \( F_i(S) \) is used to denote the mutant w.r.t. \( S[i] \) in the fault class \( F \) and \( F(S) = \{F_i(S)\} \) for all \( i \).

**Operator Faults** occur due to incorrect logical operators and can be classified as follows:

- **Operator Negation Fault (ONF):** A sub-expression \( S[i] \) of \( S \) is replaced by its negation, denoted by \( ONF_i(S) \). \( ONF \) has a special sub-class.
  - \( \Box \) Literal Negation Fault (LNF): A literal \( S[i] \) of \( S \) is replaced by its negation, denoted by \( LNF_i(S) \). \( ONF^+(S) = ONF(S) - LNF(S) \).
- **Operator Reference Fault (ORF):** A binary logical operator \( S(i) \) of \( S \) is replaced by another binary logical operator. ORF can be divided into two disjoint sub-classes.
  - \( \Box \) Operator Reference Fault (ORF[i]): An operator \( \cdot \) (index \( i \)) of \( S \) is replaced by \( + \), denoted by \( ORF[i,+](S) \).
  - \( \Box \) Operator Reference Fault (ORF[+]): An operator \( + \) (index \( i \)) of \( S \) is replaced by \( \cdot \), denoted by \( ORF[+,-](S) \).

**Operand Faults** occur due to incorrect literals and can be classified as follows:

- **Literal Stuck Fault (LSF):** A literal \( S[i] \) of \( S \) is replaced by 0 or 1, denoted by \( LSF(S) \). LSF can be divided into two disjoint sub-classes.
  - \( \Box \) Literal Stuck Fault (LSF0): A literal \( S[i] \) is replaced by 0, denoted by \( LSF_0(S) \).
  - \( \Box \) Literal Stuck Fault (LSF1): A literal \( S[i] \) is replaced by 1, denoted by \( LSF_1(S) \).
- **Literal Reference Fault (LRF):** A literal \( S[i] \) of \( S \) is replaced by another literal \( l \) whose variable is in \( S \), denoted by \( LRF_l(S) \).
- **Literal Insertion Fault (LIF):** A literal is incorrectly inserted into the Boolean expression. LIF can be divided into two disjoint sub-classes.
  - \( \Box \) Literal Insertion Fault (LIF[i]): A literal \( l' \) (index \( i \)) of \( S \) is replaced by \( l' \cdot l \), denoted by \( LIF[i,+,-](S) \).
  - \( \Box \) Literal Insertion Fault (LIF[+]): A literal \( l' \) (index \( i \)) of \( S \) is replaced by \( l' + l \), denoted by \( LIF[+,-,+](S) \).

Literal Omission Fault (LOF) are introduced and studied in [2]. It is clear that LOF is a subset of LSF, because a literal \( l \) omission with \( \cdot \) (resp. to \(+\)) is equivalent to replacing \( l \) with 1 (resp. to 0). LOF would not be discussed additionally here.

3. DETECTION CONDITIONS

In general, a mutant \( M \) may happen to be logically equivalent to the specification \( S \) and hence it cannot be distinguished by any test case. A non-equivalent mutant is called a \textit{fault} and an equivalent mutant is not regarded as a fault. A fault \( M \) is said to be killed (or detected) by a test case \( t \) if \( M(t) \neq S(t) \). That is, \( t \) is a satisfying assignment of \( M \oplus S \) (\( \oplus \) is an exclusive-or operator). \( M \oplus S \) is called the detection condition of \( M \) w.r.t. \( S \).

The formula \( S \oplus M \) is not easy to manipulate, especially when one would like to compare different fault classes of a general Boolean specification. In this section, we adopt the RELAY model [10] and introduce origination condition and...
propagation condition [9] to define Boolean difference model to compute detection conditions for various fault classes.

Given a Boolean specification S, let us consider the mutant M = S[i/X]. ∂S[i/X] denotes S ⊕ M. That is, X is the mutated sub-expression of S corresponding M. S[i] ⊕ X, denoted by ∂S[i, X], could be regarded as the origination condition of M. Obviously, if t can detect M from S, then S[i](t) ≠ X(t). However, if S[i](t) = X(t), t cannot guarantee to detect M from S, because the different values of S[i](t) and X(t) may be masked by other parts of S.

Another key point of detection condition is to formulate the propagation condition in a Boolean expression. The Boolean difference [1] is introduced and extended with respect to one sub-expression as ∂S[i] := S[i/0] ⊕ S[i/1]. If ∂S[i](t) = 0, then the mutant of S[i] will be masked. If ∂S[i](t) = 1, then the mutant of S[i] will be propagated to affect the value of S. This could be formulated as the following theorem.

**Theorem 1 (Boolean Difference Model).**

∂S[i/X] ⇔ ∂S[i, X] · ∂S[i]

**Proof.** S ≈ S[i] · S[i/1] + S[i] · S[i/0]. Therefore

∂S[i/X] = S ⊕ S[i/X]
≈ (S[i] · S[i/1] + S[i] · S[i/0]) + (X · S[i/1] + X · S[i/0])
≈ (S[i] · X) · (S[i/1] ⊕ S[i/0])
∂S[i/X] · ∂S[i] = 0

∂S[i/X] could be regarded as the set of test cases which can evaluate M to a different value from S. Based on the Boolean difference model, we can reformulate and simplify the detection conditions of fault classes as follows.

**Theorem 2 (Detection Condition).**

1. ∂ONF₁(S) ≈ ∂S[i]
2. ∂ORF₁(S) ≈ ∂S[i] · (S[ch1(i)] ⊕ S[ch2(i)])
3. ∂LSF₁(S) ≈ ∂S[i] · S[i]
4. ∂LSF₂(S) ≈ ∂S[i] · S[i]
5. ∂LRF₁(S) ≈ ∂S[i] · (S[i] ⊕ 1)
6. ∂LIF₁[+], i(S) ≈ ∂S[i] · (S[i] ⊕ 1)
7. ∂LIF₂[+], i(S) ≈ ∂S[i] · (S[i] ⊕ 1)

4. MONOTONICITY ANALYSIS

For two indexes i and j in T, i is less than j w.r.t. T, denoted by i <ₜ j, if node i is a successor of node j. Note that <ₜ is transitive. For example, in the case of S₀, 6 <ₜ 3, 3 <ₜ 1, and 6 <ₜ 1. 3 and 2 can not be compared in <ₜ. Each index of leaf is a minimal one and the index of root is the maximum one.

(A · B) ⊕ (A · B) = A · (B ⊕ B), i.e., A is the propagation condition of the difference between B and B. Similarly, (A + B) ⊕ (A + B) = A · (B ⊕ B), i.e., A is the propagation condition of the difference between B and B. Then we can compute the propagation conditions from the S[k₀] to the root of syntax tree T. Let (k₀) · · · (k₀) be the index path of T from the node k₀ to the root k₀. p(S[k₀, k₀]) is the immediate successive index of k₀, which is not on the path of k₀ in S. The propagation condition of node k₀ is defined as follows:

PC(S, k₀, k₀) :=

| S[p(S[k₀, k₀]) ] : S(k₀) = ·
| S[p(S[k₀, k₀]) ] : S(k₀) = +

For example, (7)(3)(1)(0) is the path of the node 7 of S₀. For the path, p(S₀, 7, 3) = 8, p(S₀, 7, 1) = 4 and p(S₀, 7, 0) = 2. Then PC(S₀, 7, 3) = S₀[8] = 5, PC(S₀, 7, 1) = S₀[4] = 7, PC(S₀, 7, 0) = S₀[2] = a + d. Hence, the Boolean difference model can be reformulated as the conjunction of the propagation conditions of each node kᵢ on the path.

**Lemma 3.** Let (k₀) · · · (kᵢ) be the index path of kᵢ in T of S, then

∂S[k₀] ≈ PC(S[k₀, k₁] · · · PC(S[k₀, kᵢ])

**Proof.** Please notice that ∂S[k₀] = S[k₀/0] ⊕ S[k₀/1], this lemma can be proved by i inductively for each kᵢ.

Given two indexes i and j in syntax tree T such that i is less than j w.r.t. T, if the Boolean difference of i is satisfied, then the Boolean difference of j is guaranteed to be satisfied. This is stated formally as the following Lemma.

**Lemma 4 (Monotonicity of Boolean Difference).**

T is a syntax tree of S, then

i <ₜ j ⇒ (∂S[i] ≈ ∂S[j])

**Proof.** If i <ₜ j, then S[i] is a sub-expression S[j]. Let S[j] = X, then S[i] = X[i]. ∂S[i] = S[i/0] ⊕ S[i/1].

Clearly, we could conclude the following theorem by lemma 4, because ∂ONF₁(S) ≈ ∂S[i].

**Theorem 5 (Monotonicity of ONF).**

T is a syntax tree of S, then

i <ₜ j ⇒ (∂ONF₁(S) |∂ONF₁(S))

An interesting question is whether other fault classes have the similar monotonicity. Unfortunately, the monotonicity may be destroyed by the negation operator. For example, S = (a + b) + c, S' = (a · b) + c and S'' = (a + b) · c are two mutants of ORF[+]. It is not difficult to see that the test case t = 001, i.e., (c = 0, b = 0, a = 1), can detect S' from S but it cannot detect S'' from S. For S(t) = 0, S'(t) = 1, and S''(t) = 0.

The problem described above would disappear for the Boolean expressions in negation normal form (NNF). An expression is in NNF if and only if every negation operation occurs immediately in front of a variable. An expression can be transformed into an equivalent one in NNF efficiently with De Morgan rules [6]. For NNF expressions, the monotonicity of ORF is formulated as the following theorem.

**Theorem 6 (Monotonicity of ORF in NNF).**

T is a syntax tree of S in NNF, then

i <ₜ j ⇒ (∂ORF₁[+], i(S) |∂ORF₁[+], j(S))

**Proof.** In the case of ORF[+], ∂ORF₁[+], i(S) = ∂S[i] · (S[ch1(i)] ⊕ S[ch2(i)]). If t is a satisfying assignment of ∂ORF₁[+], i(S), then (S[ch1(i)] ⊕ S[ch2(i)])(t) = 1, thus (S[ch1(i)] + S[ch2(i)])(t) = 1, i.e., S[i](t) = 1. Since i <ₜ j, one of immediate successive indexes of j should be on the path from i to j, suppose it is ch1(j). Following by the definition of propagation condition, S[ch1(j)](t) = 1, because S[i](t) = 1. S[ch2(j)](t) should be 0, since node j is “+” by computing the propagation conditions for node i. Hence
The weak detectability relationship is defined on the same position. Hence the detectability relationship can be compared with a satisfying assignment of the detection condition. Therefore, $\partial ORF^+[i](S) = \partial ORF^+[\cdot](S)$.

In the case of $ORF^-[\cdot]$, the proof is similar.

The operand faults could also be extended on non-leaf indexes to reason about the monotonicity. Stuck at Faults (STF) occur due to a sub-expression (index $i$) of $S$ is replaced with 0 or 1, denoted by $STF_0(S)$ and $STF_1(S)$, respectively. The detection conditions can be described as $\partial STF_0(S) = \partial S[i] \cdot \partial S[i]$ and $\partial STF_1(S) = \partial S[i] \cdot \partial S[i]$, respectively.

**Theorem 7 (Monotonicity of STF in NNF).**

$T$ is a syntax tree of $S$ in NNF, then

$$i < j \implies (\partial STF_0(S) = \partial STF_0(S))$$

$$i < j \implies (\partial STF_1(S) = \partial STF_1(S))$$

5. **DETECTABILITY ANALYSIS**

This section uses detection conditions to derive the weak detectability relationships between several fault classes. A strong detectability relationship is introduced to enhance and reason about the fault-based testing.

5.1 **Weak Detectability Relationship**

For any two fault classes $F^A$ and $F^B$, if a test case $t$ detects a fault in $F^A$, then $t$ will also detect a corresponding fault in $F^B$, then we say that $F^A$ weak-implies $F^B$, denoted by $F^A \rightarrow F^B$. The weak detectability hierarchy can be neatly summarized in Figure 2. We now formally state and prove the detectability relationship between different pairs of fault classes in the hierarchy.

A test case $t$ can detect a fault $M$ of $S$ if and only if $t$ is a satisfying assignment of the detection condition $S \sqsupset M$. Hence the detectability relationship can be compared with the detection conditions of fault classes. The first type of weak detectability relationship is defined on the same position between various fault classes w.r.t. literal.

**Theorem 8 (1-Weak Detectability Relationship).**

$i$ is a leaf index in $T$ and $l$ is a literal in $S$, then

1. $\partial LIF^-[\cdot](S) = \partial ORF^+[\cdot](S)$
2. $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$
3. $\partial LIF^+[\cdot](S) = \partial ORF^+[\cdot](S)$
4. $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$
5. $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$

6. $\partial LSF_0(S) = \partial ORF^+[\cdot](S)$
7. $\partial LSF_1(S) = \partial ORF^+[\cdot](S)$
8. $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$
9. $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$

**Proof.** (1-2) $S[i] \oplus l$ can be represented equivalently as $(S[i] \cdot T) + (\overline{S[i]} \cdot l)$. $\partial LIF^-[\cdot](S) = \partial LRF^+[\cdot](S)$ for $(S[i] \cdot T) = (S[i] \cdot l)$. $\partial LIF^+[\cdot](S) = \partial LRF^+[\cdot](S)$ for $(\overline{S[i]} \cdot l) = (S[i] \cdot l)$.

(3-4) $\partial LIF^+[\cdot](S) = \partial LSF_0(S)$, for $(S[i] \cdot T) = S[i]$. $\partial LIF^+[\cdot](S) = \partial LSF_1(S)$, for $(\overline{S[i]} \cdot l) = S[i]$. (5-9) They are obvious by the detection conditions.

The second type of weak detectability relationship is based on the monotonicity of ONF (theorem 3). That is, the LNF of index $i$ weak-implies ONF of index $j$ if $i$ is less than $j$.

**Theorem 9 (2-Weak Detectability Relationship).**

$i$ is a leaf index and $j$ is a non-leaf index of $T$ of $S$, such that $i < j$, then

10. $\partial LNF^+[\cdot](S) = \partial ORF^+[\cdot](S)$

It is interesting to reason about the relationship between ORF and other fault classes. The following lemma shows that detection of STF on the child indexes of $i$ can guarantee detection of ORF on the index $i$.

**Lemma 10.** $i$ is a non-leaf index in $T$ of $S$, $ch(i)$ is an immediate successive index of $i$, then

$$\partial STF_{ch(i)}(S) = \partial ORF^+[\cdot](S)$$

$$\partial STF_{ch(i)}(S) = \partial ORF^+[\cdot](S)$$

**Proof.** In the case of $STF_0(ch(i))$ and $ORF^+[\cdot]$, $\partial STF_{ch(i)}(S) = \partial S[ch(i)] \cdot S[ch(i)]$. Suppose $t$ is a satisfying assignment of $STF_{ch(i)}(S)$, then $S[ch(i)](t) = 1$. Since the node $i$ is a leaf, $S[ch(i)](t) = 0$ by computing the propagation condition of $S[ch(i)]$. That is $(S[ch(i)] \oplus S[ch(i)])(t) = 1$. Hence $\partial ORF^+[\cdot](S) = \partial ORF^+[\cdot](S)$.

In other cases, the proofs are similar.

In the above lemma, if $ch(i)$ is a leaf index, then $STF_{ch(i)}(S)$ is a literal stuck fault. The monotonicity of ORF and STF only holds for Boolean expressions in NNF.

**Theorem 11 (3-Weak Detectability Relationship).**

$T$ is a syntax tree of $S$ in NNF, $i$ is a leaf index and $j$ is a non-leaf index of $T$, such that $i < j$, then

11. $\partial LSF_0(S) = \partial ORF^+[\cdot](S)$
12. $\partial LSF_1(S) = \partial ORF^+[\cdot](S)$

5.2 **Strong Detectability Relationship**

It must be noted that $F^A \rightarrow F^B$ does not guarantee the existence of a test case for the fault $M^B$ in $F^B$. The detection condition $M^A \sqsubseteq S$ of $F^A$ may be unsatisfiable. That is, the mutant $M^A$ is equivalent to the original Boolean specification $S$ and $M^A$ should not be considered as a fault. Hence it is not necessary to test $M^A$. If $M^B$ is not equivalent to $S$, i.e. $M^B$ is a fault, but it could not generate a test case $t$ from $F^A$ to detect $M^B$.

Given a test set $T = \{t_1, \ldots, t_n\}$, we say that one fault class $F$ of $S$ is killed by $T$ if and only if at least one $t_i$ can detect $M$ for each $M$ in $F(S)$. The set of all such test sets is denoted by $K_F(S)$. It can be formulated as follows.

$$T \in K_F(S) \iff \forall M \in F(S) \exists t \in T : M(t) \neq S(t)$$ (1)
For example, \( S = a + b \). \( \{00\} \in K_{LSF}(S) \); \( \{01,10\} \in K_{LRF}(S) \). Note that the test set is always unique and it does not required to be minimum one.

If \( F^A \rightarrow F^B \), then a test set killing \( F^A \) does not guarantee to kill \( F^B \), i.e. \( K_{F^A}(S) \not\subseteq K_{F^B}(S) \). A definition of strong detectability relationship is introduced to enhance the relationship of fault classes. For any two fault classes \( F^A \) and \( F^B \), we say that \( F^A \) strongly implies \( F^B \), denoted by \( F^A \rightarrow F^B \), if \( K_{F^A}(S) \subseteq K_{F^B}(S) \) for any \( S \).

The strong detectability hierarchy can be neatly summarized in Figure 3. We now formally state and prove the strong detectability relationship between different pairs of fault classes in the hierarchy.

**Theorem 12.** \( LIF \rightarrow LRF \).

**Proof.** Given a Boolean specification \( S \) and a test set \( TS \in K_{LSF}(S) \). For any mutation \( M = LRF_{i,l}(S) \),

- Case 1: \( \partial LIF \}\[i,l\](S) or \( \partial LIF\][+\]\[i,l\](S) is satisfiable, suppose \( t \) is one satisfying assignment, then \( t \) can kill the fault \( M \), because \( LIF \rightarrow LRF \).

- Case 2: \( \partial LIF\]\[i,l\](S) and \( \partial LIF\][+\]\[i,l\](S) is unsatisfiable, then \( \partial S[l] \cdot (S[l] \cdot \overline{F} \cdot T) \) and \( \partial S[l] \cdot (S[l] \cdot \overline{F} \cdot T) \) is unsatisfiable, i.e. \( \partial S[l] \cdot (S[l] \cdot \overline{F} \cdot T + S[l] \cdot \overline{F}) \) is unsatisfiable , thus \( \partial LRF_{i,l} \) is unsatisfiable. \( M \) is not a fault.

Therefore, \( TS \in K_{LRF}(S) \), \( K_{LIF}(S) \subseteq K_{LRF}(S) \).

**Theorem 13.** \( LIF[\cdot] \rightarrow LSF0 \) and \( LIF[+\cdot] \rightarrow LSF1 \).

**Proof.** The proof is similar to theorem 12.

**Theorem 14.** \( LRF \rightarrow LNF \).

**Proof.** The proof is similar to theorem 12.

**Theorem 15.** \( LSF \rightarrow LNF \).

**Proof.** The proof is similar to theorem 12.

In the previous section, we show that the detectability relationships of ONF and STF are monotonic. The facts mean that if a test set \( T \) can detect each mutant w.r.t. leaf (literal) of Boolean specification, then \( T \) can detect other mutants in the corresponding fault class. However, some literals of Boolean specifications may be redundant, that is, they can be removed without changing any value of outcome. For example, \( S_1 = ((a + b) + (\overline{a} + \overline{b}) \cdot c \cdot (a + b) + (\overline{a} + \overline{b}) \) can be removed and \( S_2 \) is equivalent to \( c \). For two mutants of \( ONF \), \( S_{1}' = ((a + b) + (\overline{a} + \overline{b}) \cdot c \cdot (a + b) + (\overline{a} + \overline{b}) \cdot c \). \( S_1 \oplus S_1' \) or \( S_2 \) for the monotonicity of detection conditions of \( ONF \). However, \( S_2' \) is an equivalent mutant and \( S_2' \) is a fault. The monotonicity of strong detectability relationship does not hold. \( LSF \rightarrow ORF \), \( LNF \rightarrow ONF' \) and \( ORF \rightarrow ONF' \) do not hold if there is some redundant literals in Boolean specifications.

6. Conclusion

Much of the published research in fault class analysis was based on empirical evidence. However, Kuhn showed that it is possible to compare fault classes using a theoretical approach [7]. Following by his work, several hierarchies have been recently been established [8] that capture the relationships among the detection conditions of fault classes for IDNF Boolean specifications. In this paper, we extended the existing efforts [9] and established a more complete detectability hierarchy of fault classes for Boolean specifications in general form, which was called a weak one and shown in Figure 2. A strong detectability hierarchy, shown in Figure 3, provided that if all faults of a stronger fault class are killed, then the testing of the weaker faults could be skipped, thus it improved the effectiveness of fault-based testing for Boolean specifications.

7. References


