A Revisit of Fault Class Hierarchies in General Boolean Specifications

ZHENYU CHEN
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
Software Institute, Nanjing University, Nanjing 210093, China
TSONG YUEH CHEN
Centre for Software Analysis and Testing, Swinburne University of Technology, Victoria 3122, Australia
and
BAOWEN XU
State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China

Recently, Kapoor and Bowen [2007] have extended the works by Kuhn [1999], Tsuchiya and Kikuno [2002], and Lau and Yu [2005]. However, their proofs overlook the possibility that a mutant of the Boolean specifications under test may be equivalent. Hence, each of their fault relationships is either incorrect or has an incorrect proof. In this paper, we give counterexamples to the incorrect fault relationships and provide new proofs for the valid fault relationships. Furthermore, a co-stronger fault relation is introduced to establish a new fault class hierarchy for general Boolean specifications.

Categories and Subject Descriptors: D.2.4 [Software Engineering]: Software/Program Verification; D.2.1 [Software Engineering]: Requirements/Specifications; D.2.5 [Software Engineering]: Testing and Debugging

General Terms: Theory, Verification

Additional Key Words and Phrases: Fault-based testing, Boolean specifications, Fault class

The work described in this article was partially supported by the National Natural Science Foundation of China (60818027, 60803007), the National High Technology Research and Development Program of China (863 Program: 2009AA01Z147), the Major State Basic Research Development Program of China (973 Program: 2009CB320703), and the Australian Research Council (ARC DP0771733).

Authors’ addresses: Zhenyu Chen, Software Institute, Nanjing University, Nanjing 210093, China; email: zychen@software.nju.edu.cn; Tsong Yueh Chen, Centre for Software Analysis and Testing, Swinburne University of Technology, Hawthorn 3122, Australia; email: tychen@swin.edu.au; Baowen Xu, Department of Computer Science and Technology, Nanjing University, China; email: bwxu@nju.edu.cn.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 $5.00

1. INTRODUCTION

Fault-based testing methods first hypothesize certain types of faults that may be committed by programmers, and then design test cases targeted at these faults [Morell 1990]. In contrast to other testing methods, fault-based testing can demonstrate the absence of hypothesized faults. There has been an increasing interest on the use of a fault-based approach to generate test cases from software specifications in the past years [Ammann et al. 1998] [Offutt et al. 2003] [Gargantini 2007]. Since logic expressions are extensively used in both software specifications and codes, many investigations of fault-based testing have been conducted on logic expressions [Foster 1984] [Tai et al. 1994], in particular, Boolean expressions [Weyuker et al. 1994] [Chen et al. 1999] [Stamelos 2003] [Kaminski et al. 2008].

There is a close relationship between fault-based testing and mutation analysis [Demillo et al. 1978]. Mutation operators are often designed to model the potential faults. To facilitate the development of fault-based testing methods, faults induced by the same mutation operator are classified into the same category, namely fault class or fault type. Several empirical studies have observed that some fault classes are normally more difficult to detect than others [Weyuker et al. 1994] [Kobayashi et al. 2002]. These observations have motivated researchers to conduct analytical investigations into the relationships among various fault classes [Kuhn 1999] [Tsuchiya and Kikuno 2002] [Okun et al. 2004] [Lau and Yu 2005] [Kapoor and Bowen 2007].

Kuhn [Kuhn 1999] identified the relationships between three types of faults in Boolean specifications. His study was the first attempt to build a hierarchy of fault classes for Boolean specifications. Such a hierarchy could be used to determine the order in which the fault classes should be dealt with in order to achieve more cost-effective testing. Tsuchiya and Kikuno [Tsuchiya and Kikuno 2002] extended Kuhn’s three fault classes to include the fault class of a missing condition. Lau and Yu further extended Kuhn’s hierarchy by analyzing the relationships between variable faults and literal faults [Lau and Yu 2005]. Their analysis can also be applied to the design and evaluation of testing strategies [Kaminski et al. 2008].

All these studies [Kuhn 1999] [Tsuchiya and Kikuno 2002] [Lau and Yu 2005] assumed Boolean specifications in disjunctive normal form (DNF). However, Boolean expressions in a realistic program or specification are normally not in DNF. It has been shown that a single fault in a general expression may correspond to more than one fault in its corresponding DNF [Chen et al. 2007]. Okun et al. [Okun et al. 2004] presented a more general analysis of literal faults with regard to negation, reference of conjunction and disjunction, as well as insertion of conjunction and disjunction, using the RELAY model [Richardson and Thompson 1993] instead of the DNF model.

Kapoor and Bowen [Kapoor and Bowen 2007] extended the analysis to general Boolean specifications with a more complete fault class hierarchy. A novel contribution of their work is the use of a more comprehensible and useful fault relation (a subsumption relation), which is different from the fault relation (a detection capability relation) used in [Kuhn 1999] [Tsuchiya and Kikuno 2002] [Okun et al. 2004] [Lau and Yu 2005]. However, in their proofs, Kapoor and Bowen have overlooked that a mutant of the Boolean specifications under test might be equivalent. Hence,
each of their fault relationships is either incorrect or has an incorrect proof. In this paper, we use counterexamples to reveal the incorrect fault relationships and provide new proofs for the valid fault relationships. Furthermore, we introduce the notion of co-stronger fault relation for the development of a new fault class hierarchy.

This paper is organized as follows. Section 2 presents the fault classes of Boolean specifications and the definition of Kapoor and Bowen’s fault relation. In Section 3, we give counterexamples for six fault relationships in Kapoor and Bowen’s hierarchy. Also presented are a new notion of co-stronger fault relation and an enhanced fault class hierarchy. The conclusion is presented in Section 4.

2. PRELIMINARIES

We follow the definitions and notations used by Kapoor and Bowen [Kapoor and Bowen 2007]. Here a condition is a Boolean variable or a negated Boolean variable. We use the same Boolean expression \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\) to explain Kapoor and Bowen’s ten fault classes which are categorized into operator faults and operand faults, as follows:

(1) Operator Faults

(a) Operator Reference Fault (ORF). An occurrence of a logical connective \(\land\) replaced by \(\lor\) or vice versa. For example, \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is an ORF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(b) Expression Negation Fault (ENF). An ENF is a mutant with a sub-expression (except conditions) replaced by its negation. For example, \(\neg(x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is an ENF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(c) Variable Negation Fault (VNF). An occurrence of a condition is replaced by its negation. For example, \((\neg x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is a VNF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(d) Associative Shift Fault (ASF). ASF is caused by omission of the brackets because of the misunderstanding about operator evaluation priorities. For example, \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is an ASF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(2) Operand Faults

(a) Missing Variable Fault (MVF). An occurrence of a condition is omitted in the expression. For example, \((x_1 \lor \neg x_2) \land (x_3)\) is an MVF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\). Note that a condition of MVF may be connected by \(\land\) or \(\lor\).

(b) Variable Reference Fault (VRF). An occurrence of a condition is replaced by another possible condition. A condition is said to be possible if its variable has already appeared in the expression. For example, \((x_1 \lor \neg x_2) \land (\neg x_1 \land x_4)\) is a VRF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(c) Clause Conjunction Fault (CCF). An occurrence of condition \(c\) is replaced by \(c \land c'\), in which \(c'\) is a possible condition. For example, \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is a CCF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(d) Clause Disjunction Fault (CDF). An occurrence of condition \(c\) is replaced by \(c \lor c'\), in which \(c'\) is a possible condition. For example, \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\) is a CDF of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

(e) Stuck-At-0 Fault (SA0). An occurrence of a condition is replaced by 0 in the expression. For example, \((x_1 \lor 0) \land (x_3 \land x_4)\) is an SA0 of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).
(f) Stuck-At-1 Fault (SA1). An occurrence of a condition is replaced by 1 in the expression. For example, \((x_1 \lor 1) \land (x_3 \land x_4)\) is an SA1 of \((x_1 \lor \neg x_2) \land (x_3 \land x_4)\).

In the sequel, we use \(F\) to denote a Boolean expression, and \(F_\delta\) to denote a possible mutant of \(F\) for the fault class \(\delta\). The fault relation used by Kapoor and Bowen is defined as follows [Kapoor and Bowen 2007].

**Definition 1 Fault Relation \(\geq_f\).**

For any Boolean expression \(F\) and two fault classes \(\delta_1\) and \(\delta_2\), if any test set detecting all possible faulty implementations of \(\delta_1\) for \(F\) guarantees to detect all faulty implementations of \(\delta_2\) for \(F\), then \(\delta_1\) is said to be stronger than \(\delta_2\), denoted by \(\delta_1 \geq_f \delta_2\).

Kapoor and Bowen [Kapoor and Bowen 2007] defined an implementation as faulty if some failure-causing test cases exist. In other words, \(F_\delta\) is a faulty implementation of \(F\) if and only if \(F_\delta\) is a non-equivalent mutant of \(F\), that is \(F_\delta \oplus F\) is satisfiable. Any satisfying assignment of \(F_\delta \oplus F\) is a failure-causing test case of \(F_\delta\). We use \(K(F, \delta)\) to denote all test sets which can kill all possible faulty implementations \(F_\delta\) of \(F\). That is, \(T \in K(F, \delta)\) if and only if for each faulty implementation \(F_\delta\), there exists a failure-causing test case \(t \in T\) which can kill \(F_\delta\). The following relationship is immediate after the definition of \(\geq_f\):

\[
\delta_1 \geq_f \delta_2 \iff \forall F : K(F, \delta_1) \subseteq K(F, \delta_2)
\]

3. FAULT CLASS HIERARCHIES

3.1 Kapoor and Bowen’s Hierarchy

Kapoor and Bowen used the fault relation as defined in Definition 1 to build a hierarchy for the ten fault classes. They claimed that \(VNF \geq_f ENF\), \(MVF \geq_f VNF\), \(VRF \geq_f VNF\), \(CCF \geq_f VRF\), \(CDF \geq_f VRF\), \(CCF \geq_f SA0 \geq_f VNF\), and \(CDF \geq_f SA1 \geq_f VNF\) in their theorem 4 [Kapoor and Bowen 2007]. Their hierarchy is depicted in Figure 1, where \(\delta_1 \rightarrow \delta_2\) means that \(\delta_1 \geq_f \delta_2\). Immediately following the definition of \(\geq_f\), this hierarchy implies that if a test set can detect all possible faults related to \(ASF\), \(ORF\), \(MVF\), \(CCF\) and \(CDF\), this test set is also guaranteed to detect all possible faults related to the other five fault classes.
3.2 Counterexamples

However, in their proofs [Kapoor and Bowen 2007], Kapoor and Bowen have overlooked that a mutant of the Boolean specifications under test might be equivalent. Six of their fault relationships are in fact incorrect. These incorrect relationships are: \( MVF \geq_f VNF \), \( VNF \geq_f ENF \), \( SA0 \geq_f VNF \), \( SA1 \geq_f VNF \), \( CCF \geq_f VRF \) and \( CDF \geq_f VRF \). We give counterexamples to justify our claim.

**Example 1.** \( MVF \not\geq_f VNF \).

Consider the Boolean expression \((x_1 \land \neg x_2) \lor x_2\). All mutants of \( MVF \), one mutant of \( VNF \) and their failure-causing test cases are shown in Table I. A mutant is an equivalent one if and only if there is no failure-causing test case for it. Hence, we do not need to consider the case of equivalent mutants. The test set \{00, 01\}\(^1\) kills all faulty \( MVF \)s, but it cannot kill the faulty \( VNF \) \((x_1 \land x_2) \lor x_2\). Therefore, \( MVF \not\geq_f VNF \).

**Example 2.** \( CCF \not\geq_f VRF \), \( SA0 \not\geq_f VNF \) and \( VNF \not\geq_f ENF \).

Consider the Boolean expression \((x_1 \land x_2) \land x_3 \lor x_1 \lor x_3 \lor x_2 \land x_3\). All mutants of \( CCF \), \( SA0 \) and \( VNF \), one mutant of \( VRF \), one mutant of \( ENF \), and their failure-causing test cases are shown in Table II.

The test set \{011, 101\} can kill all faulty \( CCF \)’s but it cannot kill the faulty \( VRF \) \((x_1 \land x_2) \land x_3 \lor \neg x_2 \lor x_1 \lor x_3 \lor x_2 \land x_3\). Hence, \( CCF \not\geq_f VRF \).

The test set \{011, 101\} can kill all faulty \( SA0 \)’s but it cannot kill the faulty \( VNF \) \((x_1 \land x_2) \land \neg x_3 \lor x_1 \lor x_3 \lor x_2 \land x_3\). Hence, \( SA0 \not\geq_f VNF \).

The test set \{011, 101, 110\} can kill all faulty \( VNF \)’s but it cannot kill the faulty \( ENF \) \((\neg x_1 \land x_2) \land x_3 \lor x_1 \lor x_3 \lor x_2 \land x_3\). Hence, \( VNF \not\geq_f ENF \).

**Example 3.** \( CDF \not\geq_f VRF \) and \( SA1 \not\geq_f VNF \).

Consider the Boolean expression \((x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3\). All mutants of \( CDF \) and \( SA1 \), one mutant of \( VNF \), one mutant of \( VRF \), and their failure-causing test cases are shown in Table III.

The test set \{010, 100\} can kill all faulty \( CDF \)’s but it cannot kill the faulty \( VRF \) \((x_1 \lor x_2 \lor x_3) \land (\neg x_2 \lor x_3) \land (x_2 \lor x_3\). Hence, \( CDF \not\geq_f VRF \).

The test set \{010, 100\} can kill all faulty \( SA1 \)’s but it cannot kill the faulty \( VNF \) \((x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3\). Hence, \( SA1 \not\geq_f VNF \).

---

1\(^{1}\)“01” denotes the assignment of \( x_1 = 0 \) and \( x_2 = 1 \)
Table II. Mutants of \((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3\)

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Mutants</th>
<th>Failure-causing Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>all CCFs</td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_1 \land x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_2 \land x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_2 \land x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_3 \land x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_3 \land x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_1 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_2 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_1 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_2 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_3 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_3 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_1 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_1 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_2 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_2 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_3 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_3 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>011</td>
</tr>
<tr>
<td>all SA0s</td>
<td>((0 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land 0) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land 0 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor 0 \lor x_3 \lor x_2 \land x_3)</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land 0 \lor x_2 \land x_3)</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor 0 \lor x_3)</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land 0)</td>
<td>011</td>
</tr>
<tr>
<td>all VNFs</td>
<td>((\neg x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land \neg x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land \neg x_3 \land x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>001, 101</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land \neg x_3 \land x_2 \land x_3)</td>
<td>100, 101, 110</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor \neg x_2 \land x_3)</td>
<td>001, 011</td>
</tr>
<tr>
<td></td>
<td>((x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land \neg x_2)</td>
<td>010, 011, 110</td>
</tr>
<tr>
<td>one VRF</td>
<td>((x_1 \land x_2) \land x_3 \lor \neg x_2 \land x_3 \lor x_2 \land x_3)</td>
<td>001</td>
</tr>
<tr>
<td>one ENF</td>
<td>((\neg x_1 \land x_2) \land x_3 \lor x_1 \land x_3 \lor x_2 \land x_3)</td>
<td>01</td>
</tr>
</tbody>
</table>

Fig. 2. Kapoor and Bowen's Hierarchy after Correction

ACM Transactions on Software Engineering and Methodology, Vol. V, No. N, Month 20YY.
Table III. Mutants of \((x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3)\)

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Mutants</th>
<th>Failure-causing Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>all CDPs</td>
<td>((x_1 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_1 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_2 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_2 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((\neg x_3 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_3 \lor x_1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_2 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>((x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td>all SA1s</td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>100</td>
</tr>
<tr>
<td>one (VNF)</td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>110</td>
</tr>
<tr>
<td>one (VRF)</td>
<td>((1 \lor x_2 \lor x_3) \land (x_1 \lor x_3) \land (x_2 \lor x_3))</td>
<td>001</td>
</tr>
</tbody>
</table>

Originally, there are nine fault relationships in Kapoor and Bowen’s hierarchy as shown in Figure 1. In view of our counterexamples to the six fault relationships given above, their fault class hierarchy turns out to be a very disconnected one as shown in Figure 2. This has motivated us to further investigate whether there are other less explicit but useful or interesting relationships among these fault classes.

### 3.3 An Enhanced Hierarchy

\(F_3\) is a faulty implementation if and only if \(F_3 \oplus F\) is satisfiable. If \(F_3 \oplus F\) is satisfiable, then any satisfying assignment of \(F_3 \oplus F\) is a failure-causing test case of \(F_3\). Hence \(F_3 \oplus F\) is also called the detection condition of \(F_3\). Given two fault classes \(\delta_1\) and \(\delta_2\), it is not difficult to see that \(\delta_1 \geq \delta_2\) if and only if for any Boolean expression \(F\) and any faulty implementation \(F_{\delta_2}\), there exists a faulty implementation \(F_{\delta_1}\) such that \(F \oplus F_{\delta_1} \Rightarrow F \oplus F_{\delta_2}\), where \(F \Rightarrow F_2\) means that any satisfying assignment of \(F_1\) is also a satisfying assignment of \(F_2\).

Although Kapoor and Bowen overlooked the case of equivalent mutants in their incorrect proofs of \(CCF \geq_{f} SA0, \ CDF \geq_{f} SA1\) and \(VRF \geq_{f} VNF\) [Kapoor and Bowen 2007], these relationships are in fact valid. We will provide the correct proofs for them in this section.

Given a Boolean expression \(F\) and a condition \(c\), consider six fault classes: \(VNF,\)
SA0, SA1, VRF, CCF and CDF for F. A mutant \( F_3 \) of the six fault classes is to replace an occurrence of a condition \( c \) by \( \neg c \), 0, 1, \( c' \), \( (c \land c') \) or \( (c \lor c') \) where \( c' \) is a possible condition different from \( c \). We use \( F^{c,z} \) to denote the respective mutant, where \( z \) is \( \neg c \), 0, 1, \( c' \), \( (c \land c') \) and \( (c \lor c') \) respectively\(^2\).

In order to facilitate the proofs, a Boolean difference model similar to those used in [Chen et al. 2008] is used to reexpress the detection conditions of fault classes. Given a Boolean expression \( F \), \( \partial F^{c,z} \) denotes \( F \oplus F^{c,z} \), in which \( z \) is the mutated sub-expression of \( c \) in \( F \). And \( \partial F^c \) denotes \( F^c \oplus F^{c,1} \). If \( F_1 \Rightarrow F_2 \) and \( F_2 \Rightarrow F_1 \), then \( F_1 \approx F_2 \), in which \( \approx \) is the logical equivalence.

**Lemma 1 Boolean Difference Model.**

\[
\partial F^{c,z} \approx (c \oplus z) \land \partial F^c
\]

**Proof.** Since \( F \approx (c \land F^{c,1}) \lor (\neg c \land F^{c,0}) \),

\[
\partial F^{c,z} = F \oplus F^{c,z}
\]

\[
\approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land (F^{c,2,z}) \lor (\neg z \land (F^{c,2,z}) \land F^{z,2}))
\]

\[
\approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land F^{c,0}) \lor (\neg z \land (F^{c,1})))
\]

For any assignment \( A \), either \( A(c) = A(z) \) or \( A(c) \neq A(z) \).

**Case 1:** \( A(c) = A(z) = 1 \), then \( (c \oplus z) \land \partial F^c \approx 0 \).

1.1 If \( A(c) = A(z) = 1 \), then

\[
\partial F^{c,z} \approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land F^{c,1}) \lor (\neg z \land F^{c,0}))
\]

\[
\approx ((1 \land F^{c,1}) \lor (0 \land F^{c,0})) \lor ((1 \land F^{c,1}) \lor (0 \land F^{c,0})) \approx F^{c,1} \oplus F^{c,1} \approx 0.
\]

1.2 If \( A(c) = A(z) = 0 \), then

\[
\partial F^{c,z} \approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land F^{c,1}) \lor (\neg z \land F^{c,0}))
\]

\[
\approx ((0 \land F^{c,1}) \lor (1 \land F^{c,0})) \lor ((0 \land F^{c,1}) \lor (1 \land F^{c,0})) \approx F^{c,0} \oplus F^{c,0} \approx 0.
\]

**Case 2:** \( A(c) \neq A(z) \), that is \( A(c \oplus z) = 1 \). Thus \( (c \oplus z) \land \partial F^c \approx F^c \).

2.1 If \( A(c) = 0 \) and \( A(z) = 1 \), then

\[
\partial F^{c,z} \approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land F^{c,1}) \lor (\neg z \land F^{c,0}))
\]

\[
\approx ((0 \land F^{c,1}) \lor (1 \land F^{c,0})) \lor ((1 \land F^{c,1}) \lor (0 \land F^{c,0})) \approx F^{c,0} \oplus F^{c,1} = F^c.
\]

2.2 If \( A(c) = 1 \) and \( A(z) = 0 \), then

\[
\partial F^{c,z} \approx ((c \land F^{c,1}) \lor (\neg c \land F^{c,0})) \lor ((z \land F^{c,1}) \lor (\neg z \land F^{c,0}))
\]

\[
\approx ((1 \land F^{c,1}) \lor (0 \land F^{c,0})) \lor ((0 \land F^{c,1}) \lor (1 \land F^{c,0})) \approx F^{c,1} \oplus F^{c,0} = F^c.
\]

Therefore, we can conclude that

\[
\partial F^{c,z} \approx (c \oplus z) \land (F^{c,0} \oplus F^{c,1}) = (c \oplus z) \land \partial F^c.
\]

Based on the Boolean difference model, detection conditions for VNF, SA0, SA1, VRF, CCF and CDF could be expressed in terms of only \( c \), \( c' \) and \( \partial F^c \) as shown in Table IV.

**Theorem 1.** (1) CCF \( \geq_f \) SA0 (2) CDF \( \geq_f \) SA1 (3) VRF \( \geq_f \) VNF.

**Proof.**

(1) Given a Boolean expression \( F \), for any \( T \in K(F, CCF) \), consider an SA0 mutant \( F^{c,0} \) and two corresponding CCF mutants \( F^{c,(c \land c')} \) and \( F^{c,(c \land c')} \).

\(^2\)There may be more than one occurrence of the same condition in an expression. An index can be used to refer to a specific occurrence, but this will make the presentation obscure and in fact such an explicit reference is unnecessary.

ACM Transactions on Software Engineering and Methodology, Vol. V, No. N, Month 20YY.
Table IV. Detection Conditions

<table>
<thead>
<tr>
<th>Fault Class</th>
<th>Detection Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNF</td>
<td>$\partial F^{c,\neg c} \approx (c \oplus \neg c) \land \partial F^c \approx \partial F^c$</td>
</tr>
<tr>
<td>SA0</td>
<td>$\partial F^{c,0} \approx (c \oplus 0) \land \partial F^c \approx c \land \partial F^c$</td>
</tr>
<tr>
<td>SA1</td>
<td>$\partial F^{c,1} \approx (c \oplus 1) \land \partial F^c \approx \neg c \land \partial F^c$</td>
</tr>
<tr>
<td>VRF</td>
<td>$\partial F^{c,e} \approx (c \oplus e) \land \partial F^c$</td>
</tr>
<tr>
<td>CCF</td>
<td>$\partial F^{c,(c \land e')} \approx (c \oplus (c \land e')) \land \partial F^c \approx (c \land \neg e') \land \partial F^c$</td>
</tr>
<tr>
<td>CDF</td>
<td>$\partial F^{c,(c \lor e')} \approx (c \oplus (c \lor e')) \land \partial F^c \approx (\neg c \land e') \land \partial F^c$</td>
</tr>
</tbody>
</table>

Case 1: Assume $F^{c,(c \land e')}$ or $F^{c,(c \land \neg e')}$ is faulty. There exists a failure-causing test case $t \in T$ for $F^{c,(c \land e')}$ or $F^{c,(c \land \neg e')}$. It follows immediately from the detection conditions in Table IV that $\partial F^{c,(c \land e')} \approx (c \land \neg e') \land \partial F^c \Rightarrow c \land \partial F^c \approx \partial F^{c,0}$. Similarly, $\partial F^{c,(c \land \neg e')} \approx \partial F^{c,0}$. Therefore, $t$ is also a failure-causing test case for $F^{c,0}$.

Case 2: Assume both $F^{c,(c \land e')}$ and $F^{c,(c \land \neg e')}$ are equivalent mutants. Then both $\partial F^{c,(c \land e')}$ and $\partial F^{c,(c \land \neg e')}$ are unsatisfiable. Since $\partial F^{c,(c \land e')} \lor \partial F^{c,(c \land \neg e')} \approx c \land \neg e' \land \partial F^c \lor c \land e' \land \partial F^c \approx c \land \partial F^c \approx \partial F^{c,0}$, $\partial F^{c,0}$ is unsatisfiable. Hence $F^{c,0}$ is an equivalent mutant.

It follows from Cases 1 and 2 that for any SA0 faulty implementation $F^{c,0}$, there exists a failure-causing test case $t \in T$ for the faulty $F^{c,0}$. Therefore $T \subseteq K(F,SA0)$. Hence $K(F,CCF) \subseteq K(F,SA0)$. Thus we have $CCF \geq_f SA0$.

(2) The proof is similar to the above proof of $CCF \geq_f SA0$.

(3) Given a Boolean expression $F$, for any $T \in K(F,VRF)$, consider a VNF mutant $F^{c,\neg c}$ and two corresponding VRF mutants $F^{c,e}$ and $F^{c,\neg e}$.

Case 1: Assume $F^{c,e}$ or $F^{c,\neg e}$ is faulty. There exists a failure-causing test case $t \in T$ for $F^{c,e}$ or $F^{c,\neg e}$. It follows immediately from the detection conditions in Table IV that $\partial F^{c,e} \approx (c \oplus e) \land \partial F^c \Rightarrow F^c \approx \partial F^{c,\neg c}$. Similarly, $\partial F^{c,\neg e} \Rightarrow \partial F^{c,\neg c}$. Therefore, $t$ is also a failure-causing test case for $F^{c,\neg c}$.

Case 2: Assume both $F^{c,e}$ and $F^{c,\neg e}$ are equivalent mutants. Then both $\partial F^{c,e}$ and $\partial F^{c,\neg e}$ are unsatisfiable. Since $\partial F^{c,e} \lor \partial F^{c,\neg e} \approx (c \oplus e) \land \partial F^c \lor (c \oplus \neg e) \land \partial F^c \approx c \land \partial F^c \approx \partial F^{c,0}$, $\partial F^{c,0}$ is unsatisfiable. Hence $F^{c,0}$ is an equivalent mutant.

It follows from Cases 1 and 2 that for any VNF faulty implementation $F^{c,\neg c}$, there exists a failure-causing test case $t \in T$ for the faulty $F^{c,\neg c}$. Therefore $T \subseteq K(F,VNF)$. Hence $K(F,VRF) \subseteq K(F,VNF)$. Thus we have $VRF \geq_f VNF$.

Furthermore, we find some instances in which two fault classes are collectively stronger or “co-stronger” than another fault class. The formal definition of our co-stronger fault relation is as follows:

**Definition 2 Co-stronger Fault Relation.**

For any Boolean expression $F$ and fault classes $\delta_1$, $\delta_2$ and $\delta_3$, if any test set detecting all possible faulty implementations of $\delta_1$ and $\delta_2$ for $F$ guarantees to detect all faulty implementations of $\delta_3$ for $F$, then $\delta_1$ and $\delta_2$ are said to be co-stronger than $\delta_3$, denoted by $(\delta_1 \cup \delta_2) \geq_f \delta_3$.
It is obvious from the definition that we have the following relationship:

\[(\delta_1 \cup \delta_2) \supseteq \forall F : (K(F, \delta_1) \cap K(F, \delta_2)) \subseteq K(F, \delta_3)\]

With this intuition of co-stronger, we are able to identify some interesting relationships which are stated in Theorem 2.

**Theorem 2.** (1) \((CCF \cup CDF) \geq_f VRF\) (2) \((SA0 \cup SA1) \geq_f MVF\) (3) \((SA0 \cup SA1) \geq_f VNF\).

**Proof.**

(1) Given a Boolean expression \(F\), for any \(T \in K(F, CCF) \cap K(F, CDF)\), consider a \(VRF\) mutant \(F_{c,e}^{\prime}\), a corresponding \(CCF\) mutant \(F_{c,(c\wedge e)}\) and a corresponding \(CDF\) mutant \(F_{c,(c\vee e)}\).

   Case 1: Assume \(F_{c,(c\wedge e)}\) or \(F_{c,(c\vee e)}\) is faulty. There exists a failure-causing test case \(t \in T\) for \(F_{c,(c\wedge e)}\) or \(F_{c,(c\vee e)}\). It follows immediately from the detection conditions in Table IV that \(\partial F_{c,(c\wedge e)} \approx (c \wedge \lnot e) \wedge \partial F_c \Rightarrow (c \oplus e) \wedge \partial F_c \approx \partial F_{c,\lnot e}\). Similarly, \(\partial F_{c,(c\vee e)} \Rightarrow \partial F_{c,\lnot e}\). Therefore, \(t\) is also a failure-causing test case for \(F_{c,\lnot e}^{\prime}\).

   Case 2: Assume both \(F_{c,(c\wedge e)}\) and \(F_{c,(c\vee e)}\) are equivalent mutants. Then both \(\partial F_{c,(c\wedge e)}\) and \(\partial F_{c,(c\vee e)}\) are unsatisfiable. Since \(\partial F_{c,(c\wedge e)} \lor \partial F_{c,(c\vee e)} \approx c \wedge \lnot e \lor -c \wedge e \lor \partial F_c \approx (c \oplus e) \wedge \partial F_c \approx \partial F_{c,e}^{\prime}\), \(\partial F_{c,\lnot e}\) is unsatisfiable. Hence \(F_{c,e}^{\prime}\) is an equivalent mutant.

   It follows from Cases 1 and 2 that for any \(VRF\) faulty implementation \(F_{c,e}^{\prime}\), there exists a failure-causing test case \(t \in T\) for the faulty \(F_{c,e}^{\prime}\). Therefore, \(T \in K(F, VRF)\). Hence, \(K(F, CCF) \cap K(F, CDF) \subseteq K(F, VRF)\). Thus, we have \((CCF \cup CDF) \geq_f VRF\).

(2) Given a Boolean expression \(F\), for any \(T \in K(F, SA0) \cap K(F, SA1)\), consider a mutant \(F_{mef}\) with missing condition \(c\). In the evaluation of the original expression \(F\), \(c\) is evaluated as an operand of either \(\land\) and \(\lor\). If \(c\) is evaluated as an operand of \(\land\), then \(F_{mef} \approx F_{c,0}^{\prime}\); if \(c\) is evaluated as an operand of \(\lor\), then \(F_{mef} \approx F_{c,1}^{\prime}\). If \(F_{mef}\) is a faulty implementation, then \(F_{c,0}\) or \(F_{c,1}\) is a faulty implementation. There exists a failure-causing test case \(t \in T\) for \(F_{c,0}\) or \(F_{c,1}\). Therefore \(t\) is also a failure-causing test case for \(F_{mef}\), and hence \(T \in K(F, MVF)\). Thus, we have \((SA0 \cup SA1) \geq_f MVF\).

(3) Given a Boolean expression \(F\), for any \(T \in K(F, SA0) \cap K(F, SA1)\), consider a \(VNF\) mutant \(F_{c,\lnot c}\) and a corresponding \(SA0\) mutant \(F_{c,0}\) and a corresponding \(SA1\) mutant \(F_{c,1}\).

   Case 1: Assume \(F_{c,0}\) or \(F_{c,1}\) is faulty. There exists a failure-causing test case \(t \in T\) for \(F_{c,0}\) or \(F_{c,1}\). It follows immediately from the detection conditions in Table IV that \(\partial F_{c,0} \approx c \land \partial F_c \Rightarrow \partial F_c \approx \partial F_{c,\lnot c}\). Similarly, \(\partial F_{c,1} \Rightarrow \partial F_{c,\lnot c}\).

   Therefore, \(t\) is also a failure-causing test case for \(F_{c,\lnot c}\).

   Case 2: Assume both \(F_{c,0}\) and \(F_{c,1}\) are equivalent mutants. Then \(\partial F_{c,0}\) and \(\partial F_{c,1}\) are unsatisfiable. Since \(\partial F_{c,0} \lor \partial F_{c,1} \approx c \land \partial F_c \lor \partial F_{c,\lnot c} \lor \partial F_c \approx \partial F_c \approx F_{c,\lnot c}\), \(\partial F_{c,\lnot c}\) is unsatisfiable. \(F_{c,\lnot c}\) is an equivalent mutant.

ACM Transactions on Software Engineering and Methodology, Vol. V, No. N, Month 20YY.
It follows from Cases 1 and 2 that for any $VNF$ faulty implementation $F^c, \neg c$, there exists a failure-causing test case $t \in T$ for the faulty $F^c, \neg c$. Therefore, we have $T \in K(F, VNF)$. Thus, we have proved that $(SA_0 \cup SA_1) \geq_f VNF$.

As a result, an enhanced fault class hierarchy is developed in Figure 3. Obviously, our hierarchy is more informative than the hierarchy in Figure 2. It shows that there are five core fault classes, namely, $ASF$, $ORF$, $CCF$, $CDF$ and $ENF$. To reveal all faults in the ten fault classes, it is sufficient to use a test set for these five core fault classes only. In other words, the test sets for the non-core fault classes are redundant. Therefore, this hierarchy helps to identify redundant test cases, and hence to reduce the cost of testing. Furthermore, it helps to prioritize test cases as follows. Among the five core fault classes, $CCF$ and $CDF$ should be targeted before $ASF$, $ORF$ and $ENF$. Intuitively speaking, such an ordering of testing would yield an earlier detection of faults, because test sets revealing faults of $CCF$ and $CDF$ would also reveal faults of the remaining fault classes, while test sets revealing faults of $ASF$, $ORF$ and $ENF$ may not have this additional benefit.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{enhanced_hierarchy.png}
\caption{An Enhanced Hierarchy}
\end{figure}

4. CONCLUSION

Kuhn firstly showed that it is possible to compare fault classes using a theoretical approach [Kuhn 1999]. Following his pioneering work, several hierarchies have been subsequently established to capture the relationships of fault classes for DNF Boolean specifications [Tsuchiya and Kikuno 2002] [Lau and Yu 2005]. However, in practice, DNF are rarely used. Though there exists a standard algorithm to convert a general Boolean expression into its equivalent DNF, it is well known that a single fault in a general form may lead to multiple faults in the corresponding DNF. Hence, research has been extended to the fault class hierarchies for general Boolean expressions [Okun et al. 2004] [Kapoor and Bowen 2007]. In this paper, we have identified the flaws in Kapoor and Bowen’s hierarchy [Kapoor and Bowen 2007] and have presented an enhanced fault class hierarchy with richer information and structure. Our results show that $ASF$, $ORF$, $CCF$, $CDF$ and $ENF$ are the core fault classes. Detection of these five core types of faults guarantees the detection of the remaining five types of faults.
Acknowledgement
The authors would like to thank Prof. Decheng Ding for his valuable discussions.

REFERENCES


ACM Transactions on Software Engineering and Methodology, Vol. V, No. N, Month 20YY.