2009 International Conference on Software Testing Verification and Validation

Using Logic Criterion Feasibility to Reduce Test Set Size While Guaranteeing Fault Detection

Garrett Kaminski and Paul Ammann
Software Engineering, George Mason University
Fairfax, VA USA
gkaminsk@gmu.edu, pammann@gmu.edu

Abstract

Some software testing logic coverage criteria demand inputs that guarantee detection of a large set of fault types. One powerful such criterion, MUMCUT, is composed of three criteria, where each constituent criterion ensures the detection of specific fault types. In practice, the criteria may overlap in terms of fault types detected, thereby leading to numerous redundant tests, but due to the unfortunate fact that infeasible test requirements don't result in tests, all the constituent criteria are needed. The key insight of this paper is that analysis of the feasibility of the constituent criteria can be used to reduce test set size without sacrificing fault detection. In other words, expensive criteria can be reserved for use only when they are actually necessary. This paper introduces a new logic criterion, Minimal-MUMCUT, based on this insight. Given a predicate in minimal DNF, a determination is made of which constituent criteria are feasible at the level of individual literals and terms. This in turn determines which criteria are necessary, again at the level of individual literals and terms. This paper presents an empirical study using predicates in avionics software. The study found that Minimal-MUMCUT reduces test set size -- without sacrificing fault detection -- to as little as a few percent of the test set size needed if feasibility is not considered.

KEY WORDS: Software Logic Testing, Software Fault, Criteria, MUMCUT, Disjunctive Normal Form

1. Introduction

Infeasible test requirements are demands for tests that simply do not exist. They are an unfortunate fact of life in software testing. They confound test engineers, who must decide if a given test requirement really is infeasible or if a more diligent search for a suitable input is in order. They also confound attempts by researchers to relate coverage criteria. By definition, an infeasible test requirement for a given criterion does not result in a test. If the corresponding test requirement for a "weaker" criterion happens to be feasible, the infeasibility can cause an apparently "stronger" criterion to fail to subsume the "weaker" one. Many well known cases of this phenomenon pervade the testing literature. In this paper, we address infeasibility in the context of logic testing criteria designed for the fault hierarchy of Lau and Yu [9].

We consider testing predicates over Boolean variables in isolation. In this finite domain, it is straightforward to determine whether a given test requirement is feasible. Of course, when these predicates are buried inside actual programs, there is still a difficult controllability problem of selecting inputs to drive the variables in the predicate to the desired values, but that is not the focus of this research.

A predicate in n variables has at most 2^n tests. For applications where n is large, the exhaustive test set is often prohibitively expensive. Hence, some logic criteria trade fault detection capability for reduced test set size. This paper analyzes feasibility to improve solutions to this tradeoff. We focus on three faults: the Literal Insertion Fault (LIF), the Literal Reference Fault (LRF), and the Literal Omission Fault [8]. A LIF involves inserting a literal, or the negation of a literal, into a term. A LRF involves replacing a literal with a literal, or the negation of a literal, from some other term. A LOF involves omitting a literal. The LIF, LRF, and LOF are important for two reasons. First, they mimic programmer mistakes. The competent programmer hypothesis [1] states that competent programmers often write programs that differ from a correct version by relatively few simple faults, and so faults in the hierarchy are plausible errors for which to test. Second, the fault hierarchy of Lau and Yu [9] assures that detection of these faults guarantees detection of other faults. That is, these three faults sit atop the fault hierarchy.

Chen, Lau and Yu [4] developed the MUMCUT coverage criterion specifically to guarantee detection of all faults in the fault hierarchy. The MUMCUT criterion integrates three *constituent criteria*: the Multiple Unique True Point (MUTP), Multiple Near False Point (MNFP), and Corresponding Unique True Point Near False Point (CUTPNFP) criteria. Details of these constituent criteria are given in Section 2. For this paper, the key issue is the role of feasibility in whether each constituent criterion is necessary.

Specifically, if MUTP is feasible, it is possible to augment it with many fewer tests than those required by CUTPNFP or MNFP, and yet still detect the entire fault hierarchy. The situation is more complex if MUTP is infeasible. Where MUTP is infeasible, but CUTPNFP is feasible, MNFP is not needed at all. If both MUTP and CUTPNFP are infeasible, then MNFP is required. A key aspect of this paper is that the infeasibility arguments apply at the fine-grained level of terms and literals, and hence CUTPNFP

978-0-7695-3601-9/09 $25.00 © 2009 IEEE
DOI 10.1109/ICST.2009.14

and, if needed, MNFP, can be used only where they are required.

MUMCUT takes the direct approach of simply requiring all three constituent criteria. This certainly works, but it is expensive. CUTPNFP and especially MNFP demand large numbers of tests, but, as hinted at above, turn out to be necessary only in relatively few cases.

The contributions of this paper are:
1) Uses an analysis of MUMCUT constituent criterion feasibility at the level of terms and literals. This analysis allows test set sizes to be reduced without sacrificing fault detection.
2) Provides a refinement of fault detection relationships in Lau and Yu's hierarchy [9] based on constituent criterion feasibility from the MUMCUT criterion (Figure 3.1).
3) Presents a new logic coverage criterion, Minimal-MUMCUT, as well as an algorithm to generate Minimal-MUMCUT test sets.
4) Gives a case study that shows the reductions in test set size possible with Minimal-MUMCUT.

The paper is organized as follows. The remainder of Section 1 reviews relevant Boolean logic terminology and related work in logic criteria. Section 2 reviews MUMCUT and the three constituent criteria MUTP, CUTPNFP, and MNFP. Section 3 reviews the fault hierarchy. Section 4 develops the results explaining the impact of infeasibility on fault detection, presents algorithms to determine feasibility for each criterion at the level of terms and literals, and finally synthesizes these results into the Minimal-MUMCUT algorithm. Section 5 is a case study to assess the reduction in test set size provided by Minimal-MUMCUT. Section 6 discusses how this work relates to more general issues in testing, and section 7 concludes the paper.

1.1 Boolean Logic Terminology

Table 1.1 lists the definitions for terms used in this paper.

Table 1.1 Basic Definitions

Term or Symbol	Definition
1	The Boolean value TRUE
0	The Boolean value FALSE
Literals	Variables representing clauses in a predicate
+	OR operator
Adjacency between literals	AND operator
Term	A set of literals connected by AND
~	negation
Disjunctive Normal Form (DNF)	Predicate syntax where terms are separated by OR and literals are separated by AND
Implicant	A term that when TRUE, means the predicate is TRUE
Prime implicants	Implicants where removing a literal could potentially change the value of the predicate
Irredundant DNF	Predicate syntax where it is possible to make each term TRUE in turn while all other terms are FALSE
Minimal DNF	Predicate syntax in irredundant DNF where all implicants are prime implicants
Unique True Point (UTP)	An assignment of values such that only a single term is TRUE. In ab + cd, UTPs for ab are 1100, 1101, 1110.
Near False Point (NFP)	An assignment of values such that the predicate is FALSE but negating a single literal makes the predicate TRUE [3]. In ab + cd, NFPs for a are 0100, 0101, 0110.
Corresponding NFP	A NFP that differs from an UTP for the literal's term only in the value of that literal. In ab + cd, 0100 is a corresponding NFP for a as it differs from the UTP 1100 for ab only in the value of a.
Feasible	A logic criterion is feasible if and only if it is possible to construct all required tests.

1.2 Related Work

Logic criteria have been studied syntactically (assuming a predicate is in a particular format) and semantically (making no assumption as to format). Chilenski and Miller [6] discuss the modified condition / decision coverage (MC/DC) criterion which is the best known semantic logic criteria. However, MC/DC tests do not guarantee detecting most faults in Lau and Yu's hierarchy [7]. Weyuker, Goradia, and Singh [13] proposed the MAX-A and MAX-B syntactic criteria, whose tests guarantee detecting all faults in the hierarchy. Chen, Lau, and Yu [4] developed the MUMCUT criterion, whose tests guarantee detecting all faults in the hierarchy with a smaller test set size. Chen and Lau [2] implemented the MUTP Greedy algorithm to satisfy the MUTP criterion as a constituent of the MUMCUT criterion. Kaminski, Williams, and Ammann [7] proposed the MUTP/NFP criterion, whose tests guarantee detection of all faults in the hierarchy while further reducing test set size, but only if the criterion is feasible. Sun et al. [12] analyzed how MUMCUT can be extended to apply to predicates in any format and Okun, Black, and Yesha [11] showed how a logic fault hierarchy can apply to predicates in any format. The seminal work in composing a logic fault hierarchy was performed by Kuhn [8]. Lau and Yu [9] refined Kuhn's work by introducing new faults and detection relationships.

This paper advances prior research by focusing on criterion feasibility for individual terms and literals. The result is a new criterion reducing test set size without sacrificing fault detection, even if infeasibility occurs for the predicate as a whole.

2. Logic Criteria

Exhaustive logic test size grows exponentially, requiring tests of $O(2^n)$, where n is the number of unique literals. Thus, testers have invented less expensive criteria. Four such criteria are described next with an example of *ab + cd*. A summary of each along with a new logic criterion introduced in section 4 is given in Appendix A. Note that for all of these criteria, if an infeasibility occurs the tests chosen should satisfy the requirements as fully as possible.

2.1 MUTP

Multiple Unique True Point (MUTP): Given a minimal DNF predicate, form tests for an UTP for each term such that all literals not in the term attain values 1 and 0. An UTP for the first term must have *a*=1, *b*=1. Needed tests for *c* and *d* to each = 0 and 1 are 1101 and 1110. An UTP for the second term must have *c*=1 and *d*=1. Needed tests for *a* and *b* to each = 0 and 1 are 0111 and 1011. A test set is {1101,1110,0111,1011}.

2.2 MNFP

Multiple Near False Point (MNFP): Given a minimal DNF predicate, form tests for a NFP of each literal such that all literals not in the literal's term attain values 1 and 0. An UTP for the first term must have *a*=1, *b*=1. NFPs for *a* and *b* so that *c* and *d* each = 0 and 1 are 0101,0110,1001,1010. An UTP for the second term must have *c*=1 and *d*=1. Needed NFPs for *c* and *d* so that *a* and *b* each = 0 and 1 are 0101,1001,0110,1010. A test set is {0101,0110,1001,1010}.

2.3 CUTPNFP

Corresponding Unique True Point Near False Point Pair (CUTPNFP): Given a minimal DNF predicate, for every literal find an UTP and NFP such that only the literal changes value. An UTP for the first term must have *a*=1, *b*=1. If *c*=0 and *d*=1, tests for *ab* are 1101,0101,1001. An UTP for the second term must have *c*=1, *d* = 1. If *a*=1 and *b* = 0, tests for *cd* are 1011,1001,1010. A test set is {1101,0101,1001,1011,1010}.

2.4 MUMCUT

MUTP/MNFP/CUTPNFP (MUMCUT): Satisfy the CUTPNFP, MUTP, and MNFP criteria. 1101 and 1110 are UTPs for *ab*. 0101 and 0110 are NFPs for *a* that differ from an UTP for *ab* only in the value *a*. 1001 and 1010 are NFPs for *b* that differ from an UTP for *ab* only in the value of *b*. 0111 and 1011 are UTPs for *cd*. 0101 and 1001 are NFPs for *c* that differ from an UTP of *cd* only in the value of *c*. 0110 and 1010 are NFPs for *d* that differ from an UTP for *cd* only in the value of *d*. In the NFPs above each literal not in the term of interest attains 1 and 0. A test set is {1101,1110,0101,0110,1001,1010,0111,1011}.

3. Logic Tests and Fault Hierarchy

One method for evaluating tests is to determine how many of the nine minimal DNF faults in Table 3.1 a test set is guaranteed to detect [8, 9].

Table 3.1 Typical Minimal DNF Logic Faults

Fault	Description
Expression Negation Fault (ENF)	Predicate implemented as its negation: *ab* + *c* implemented as *~(ab + c)*.
Term Negation Fault (TNF)	A term is negated: *ab* + *c* implemented as *~(ab)* + *c*.
Operator Reference Fault + (ORF+)	Replacing OR with AND: *a* + *b* implemented as *ab*.
Operator Reference Fault . (ORF.)	Replacing AND with OR: *ab* implemented as *a* + *b*.
Literal Negation Fault (LNF)	A literal is negated: *ab* implemented as *a~b*.
Literal Reference Fault (LRF)	A literal is replaced by a literal or the negation of a literal not in the term: *ab* + *cd* implemented as *cb* + *cd* or as *~cb* + *cd*.
Term Omission Fault (TOF)	A term is omitted: *ab* + *cd* implemented as *ab*.
Literal Omission Fault (LOF)	A literal is omitted: *ab* implemented as *a*.
Literal Insertion Fault (LIF)	A literal not in a term is inserted as itself or as its negation: *ab* + *cd* implemented as *abc* + *cd* or as *ab~c* + *cd*.

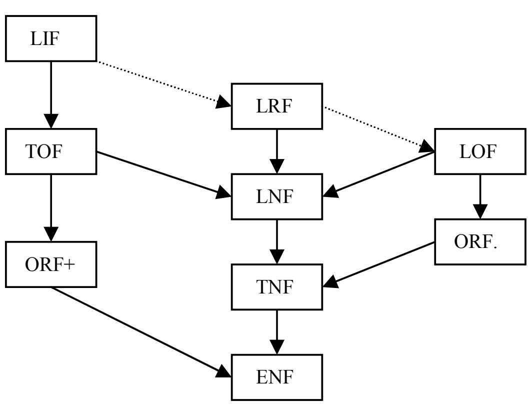
The condition for detecting an LIF is as follows [4]. If some literal not intended to be in term X is inserted into X as itself or as its negation, then a set of UTPs for X where all literals not in X attain the values 0 and 1 detects the fault. MUTP tests are guaranteed to detect an LIF [4]. However, when the MUTP criterion is infeasible, a LRF exists that MUTP tests may not detect (see Theorem 4.1). Consider *ab* + *ac* + *bc* and an LIF producing *ab*~***c*** + *ac* + *bc*. The MUTP criterion is infeasible for *ab* as the only UTP for *ab* is 110. Therefore, MUTP tests do not detect the corresponding LRFs: **~*c****b* + *ac* + *bc* and *a*~***c*** + *ac* + *bc*.

MUTP tests are guaranteed to detect a LRF for a literal if the MUTP criterion is feasible for that literal's term [7]. In this case, it is only necessary to satisfy the MUTP criterion and the NFP criterion (a NFP for each literal in the term) to guarantee detecting an LIF, LRF, and LOF in that term [7]. The NFP for a literal in a MUTP feasible term can overlap with NFPs for other literals in other MUTP feasible terms and with NFPs in CUTPNFP or MNFP tests since any NFP for a literal detects a LOF for that literal [4]. If a term is MUTP infeasible but all external literals that cannot be 0 or 1 in an UTP for the term exist in single-literal terms, LRF detection is still guaranteed by MUTP tests. A LRF involving replacing a literal with a literal (or its negation) that exists in a single-literal term will result in a TOF, LOF, or a TRUE predicate. Since an UTP guarantees detecting a TOF [4] and a NFP guarantees detecting a LOF or a fault where the predicate is stuck at 1, a MUTP test set supplemented with overlapping NFPs guarantees LRF detection. For example, in *a* + *b*, replacing *a* with *b* results in a TOF for *a* and replacing *a* with ~*b* makes the predicate = 1. In *ab* + *c*, replacing *a* with *c* results in a TOF for *ab* and replacing *a* with ~*c* results in a LOF for *a*.

The condition for detecting a LRF is as follows [4]. If literal *x* in X is wrongly implemented as some other literal or the negation of some other literal not in X, then any of the following detects the fault: a set of UTPs for X where all literals not in X attain the values 0 and 1; a set of NFPs for *x* where all literals not in X attain the values 0 and 1; an UTP-NFP pair where the points differ only in the value of *x*. The CUTPNFP criterion is designed to produce tests that detect a LRF but fails to do so when it is infeasible (see Theorem 4.2). However, when the CUTPNFP criterion is infeasible, MNFP tests can be added to guarantee LRF detection [4]. Consider *abc* + *abd* + *~b~d* + *~de*. The CUTPNFP criterion is infeasible for *b* in *abc*. The only UTP for *abc* is 11100. A corresponding NFP of 10100 is not possible for *b* in *abc* because this is a TRUE point. Now consider the LRF *a~ec* + *abd* + *~b~d* + *~de*. Since the CUTPNFP criterion is infeasible for *b* in *abc*, this LRF goes undetected by CUTPNFP tests. A single NFP for *b* in *abc* is not guaranteed to detect a LRF either. The point 10111 is a NFP for *b* in *abc*, but this point fails to detect the LRF. The MNFP criterion requires that the NFP 10110 be used for *b* in *abc*, detecting the LRF.

Figure 3.1 displays Lau and Yu's Fault Hierarchy [9] modified based on how criterion feasibility affects fault detection. A solid arrow from a source fault to a destination fault indicates that if a test detects a source fault, it also detects a corresponding destination fault. When the MUTP criterion is infeasible, a test set detecting all LIFs is not guaranteed to detect all LRFs. Thus the solid arrow between the LIF and LRF in Lau and Yu's hierarchy is changed to a dashed arrow. In Lau and Yu's hierarchy no arrow exists between the LRF and LOF. A dashed arrow is added to represent that when guaranteeing detection of all LIFs does not guarantee detection of all LRFs, adding tests to detect the undetected LRFs will detect all corresponding LOFs (unless the CUTPNFP criterion is infeasible). The reason is that when the MUTP criterion is infeasible but the CUTPNFP criterion is feasible, an UTP will not detect a LRF but a corresponding NFP will [4].

Figure 3.1 Fault Class Hierarchy



4 Using Criterion Feasibility to Assess Fault Detection

An example of how a MUTP test set detects any LRF for any literal in a MUTP feasible term is given in Tables 4.1 and 4.2. Table 4.2 shows how MUTP tests detect each LRF by using the tests in Table 4.1.

Table 4.1 MUTP tests for *ab + ~ac*

Test Case	Values	Test Name
1	110	MUTP for *ab*
2	111	MUTP for *ab*
3	001	MUTP for *~ac*
4	011	MUTP for *~ac*

Table 4.2 Fault Detection for *ab + ~ac*

Fault Class	Faulty predicate	Test Case detecting fault	Original predicate value	Faulty predicate value
LRF	*cb* + *~ac*	1	True	False
LRF	*~cb* + *~ac*	2	True	False
LRF	*ac* + *~ac*	1	True	False
LRF	*a~c* + *~ac*	2	True	False
LRF	*ab* + *bc*	3	True	False
LRF	*ab* + *~bc*	4	True	False
LRF	*ab* + *~ab*	3	True	False
LRF	*ab* + *~a~b*	4	True	False

This section continues with proofs relating criterion feasibility to fault detection capability.

Theorem 4.1:
If the MUTP criterion is infeasible for a multi-literal term X, a LRF where the negation of some literal y in some multi-literal term Y replaces literal x in X cannot be detected by MUTP tests.

Proof:
Without loss of generality assume *y* must = 0 in an UTP for X. Consider the fault where *~y* is inserted into X. A corresponding LRF is replacing *x* with *~y*. In an UTP for X all literals are such that only X = 1. Substituting *~y* for *x* also makes X = 1 since *y* must = 0 in an UTP of X. Since X and Y are multi-literal terms, the LRF does not result in a TOF or LOF. The MUTP criterion requires only TRUE points for X so MUTP tests fail to detect a LRF.
End of Proof

An example is mutating *ab* + *bc* to *a~c* + *bc*. The MUTP criterion is infeasible for *ab* because 110 is the only UTP for *ab*. 110 does not distinguish between *ab* and *a~c* so MUTP tests fail to detect the LRF.

An algorithm to determine MUTP feasibility for each term in a predicate is given next.

MUTP Feasibility Algorithm

```
Input:  All UTPs for each term
Output: A List of MUTP feasible terms
Declare list MUTPFeasibleTerms = All terms
For each term i
   For each literal x not in term i
      Boolean doesLiteralEqual_0 = false
      Boolean doesLiteralEqual_1 = false
      For each UTP u in the set of UTPs for term i
         If (literal x = 0 in UTP u) doesLiteralEqual_0 = true
         Else doesLiteralEqual_1 = true
         If (doesLiteralEqual_0 && doesLiteralEqual_1)
            Continue for each literal loop
       End For
       Remove term i from MUTPFeasibleTerms
       Continue for each term loop
   End For
End For
Return MUTPFeasibleTerms
```

Theorem 4.2:
If the CUTPNFP criterion is infeasible for literal x in a multi-literal term X, a LRF where the negation of some literal y in some multi-literal term Y replaces x cannot be detected by CUTPNFP tests.

Proof:
Chen, Lau, and Yu [4] show that in general when the MUTP criterion is infeasible, a NFP is needed to detect a LRF. (They do not distinguish between LRFs involving single-literal vs. multi-literal terms as Theorem 4.1 does). When the CUTPNFP criterion is infeasible for *x*, a LRF cannot be detected by a corresponding NFP as none exists. Thus, if it cannot be detected by an UTP for X, CUTPNFP tests do not detect it. MUTP tests do not guarantee LRF detection when the MUTP criterion is infeasible so what remains to be proved is that when the CUTPNFP criterion is infeasible for *x*, the MUTP criterion is infeasible for X. Since no corresponding NFP exists for *x*, flipping the value of *x* in an UTP for X causes some other term Y to = 1. In this case, the MUTP criterion is infeasible for X as the following proof by contradiction shows. Assume the MUTP criterion is feasible for X and the CUTPNFP criterion is infeasible for *x*. Every literal not in X can = 0 or 1 in an UTP for X. So every term other than X can = 0 for an UTP of X no matter if *y* = 0 or 1. Thus, flipping the value of *x* in an UTP for X can form a corresponding NFP, contradicting the original assumption. Since X and Y are multi-literal terms, a LRF does not yield a TOF or LOF.
End of Proof

As an example, consider *abc* + *abd* + *~b~d* + *~de*. The CUTPNFP criterion is infeasible for *b* in *abc*. The only UTP for *abc* is 11100. A corresponding NFP of 10100 is not possible for *b* in *abc* because this is a TRUE point. Now consider the LRF *a~ec* + *abd* + *~b~d* + *~de*. Since the CUTPNFP criterion is infeasible for *b* in *abc*, CUTPNFP tests do not detect the LRF. The MNFP criterion would require that the NFP 10110 be used for *b* in *abc,* which detects the LRF.

An algorithm to determine CUTPNFP criterion feasibility for each literal in a predicate is presented next.

CUTPNFP Feasibility Algorithm

```
Input:  All UTPs for each term and all NFPs for each literal
Output: A List of all CUTPNFP feasible literals
Declare List CUTPNFPFeasibleLiterals
Declare String nfpComplement
Declare Boolean isCUTPNFPFeasible
For each term i
   For each literal j in term i
      isCUTPNFPFeasible = false
      For each nfp k for literal j in term i
         nfpComplement = k where jth bit is complemented
         If the set of utps for term i contains nfpComplement
            isCUTPNFPFeasible = true
            Break
      End For
      If (isCUTPNFPFeasible)
         Add literal j to CUTPNFPFeasibleLiterals
    End For
 End For
Return CUTPNFPFeasibleLiterals
```

A new logic criterion, which we call Minimal-MUMCUT, satisfied by a test set produced by the algorithm below, can reduce MUMCUT test set size without sacrificing fault detection. The algorithm is based on criterion feasibility to 1) overlap NFPs when possible and 2) produce CUTPNFP and MNFP tests only when necessary on a literal-by-literal basis.

Minimal-MUMCUT algorithm

```
For each term X
   Generate MUTP tests for X
   If the MUTP criterion is infeasible* for X
      For each literal x in X
         If the CUTPNFP criterion is feasible for x
            Generate CUTPNFP tests for x to overlap NFPs**
         Else Generate MNFP tests for x to overlap NFPs**
      End For
   Else
      Generate a NFP for x to overlap NFPs**
End For
```

* The MUTP criterion is infeasible in this algorithm if and only if X is a multi-literal term and a literal y in a multi-literal term Y exists where y cannot attain both truth values in an UTP for X.

** Overlapping NFPs is a set covering combinatorial optimization problem known to be NP-complete. An heuristic is used in the algorithm to approximate minimizing the number of NFPs generated.

As an example, consider *ab* + *cd*. 1101 and 1110 are UTPs for *ab* and the MUTP criterion is feasible for *ab*. 0101 and 1010 are NFPs for *a* and *b*, respectively. 0111 and 1011 are UTPs for *cd* and the MUTP criterion is feasible for *cd*. 0101 and 1010 are NFPs for *c* and *d*, respectively. A test set is {1101,1110,0101,1010,0111,1011} which has two less tests than the MUMCUT test set in section 2.4.

5 Empirical Evaluation

Chen, Lau, and Yu [4] evaluated MUMCUT test set size (using the greedy MUTP algorithm [2]) for 19 minimal DNF predicates from an air traffic collision avoidance system (TCAS). There were actually 20 predicates but number 12 was excluded due to a missing a right parenthesis [13]. The predicates have from 5 to 13 unique literals (see Appendix B). In this study, Minimal-MUMCUT tests were created for each predicate and MUTP feasibility for each term and CUTPNFP feasibility for each literal was assessed. The Minimal-MUMCUT algorithm was implemented in Java and used to obtain the results.

The results showed that the CUTPNFP criterion was feasible for all 853 literals, so MNFP tests were not needed for any literal. For 204 literals (23.92%), the MUTP criterion was feasible for the literal's term and thus MUTP tests detect a LRF. For the other 649 literals (76.08%), CUTPNFP tests detect a LRF. For four predicates, the MUTP criterion was feasible for every term, so CUTPNFP tests were not needed. For 16 predicates, the MUTP criterion was feasible for at least one term. Thus, CUTPNFP tests were not needed for literals in at least one term in most predicates. Table 5.1 displays these results. Minimal-MUMCUT and MUMCUT test set size were also compared. On average, Minimal-MUMCUT test set size was 12.66% of MUMCUT test set size and 2.50% of exhaustive test set size. The greatest savings was for predicate 13, where the Minimal-MUMCUT test set size was 1.30% of the MUMCUT test set size and 0.54% of the exhaustive test set size. Table 5.2 displays these results. Minimal-MUMCUT test set size is always less than MUMCUT test set size, except when each literal is in each term, in which case test set size is the same (see predicates 8 and 9). In this case, each term has only one UTP, each literal has only one NFP (which happens to be a corresponding NFP), and no LIFs or LRFs exist.

Table 5.1 Criterion Feasibility and LRF detection

	Number of terms that are MUTP feasible	Number of terms that are MUTP infeasible	Number of literals for which MUTP detects LRF	Number of literals needing CUTPNFP to detect LRF
1	1	4	5	24
2	4	9	33	72
3	2	23	10	136
4	1	2	1	6
5	1	8	1	27
6	2	4	22	36
7	4	4	28	32
8	4	0	32	0
9	2	0	14	0
10	0	6	0	60
11	1	8	6	57
12*	N/A	N/A	N/A	N/A
13	0	6	0	14
14	0	6	0	16
15	1	10	2	30
16	1	22	2	85
17	2	4	8	24
18	2	6	8	30
19	4	0	20	0
20	2	0	12	0
Sum	34	122	204	649

* number 12 excluded due to a missing a right parenthesis

Table 5.2 Minimal-MUMCUT and MUMCUT Test Set Size

	Minimal-MUMCUT	MUMCUT [4]	Percentage	2^n
1	27	40.50	66.67%	128
2	81	116.00	69.83%	512
3	157	1026.57	15.29%	4096
4	9	14.40	62.50%	32
5	36	232.48	15.49%	512
6	66	89.96	73.37%	2048
7	66	119.80	55.09%	1024
8	36	36.00	100.00%	256
9	16	16.00	100.00%	128
10	62	142.83	43.41%	8192
11	72	888.26	8.11%	8192
12	N/A	N/A	N/A	N/A
13	22	1687.00	1.30%	4096
14	22	73.75	29.83%	128
15	39	187.39	20.81%	512
16	107	1595.32	6.71%	4096
17	40	852.78	4.69%	2048
18	48	182.54	26.30%	1024
19	16	65.64	24.38%	256
20	14	24.00	58.33%	128
Sum	936	7391.22		37,408
Avg	49.26	389.01	12.66%	1968.84

Minimal-MUMCUT test sets for predicates 4, 13, and 19 are given next.

Predicate 4: a~bd + a~cd + e

MUTP test set is:
10110 term a~bd
11010 term a~cd
00001 term e

11111 term e

The MUTP criterion is infeasible for *a~bd* as *c* must = 1 and *e* must = 0 in an UTP for *a~bd*. However, *e* is in a single-literal term so CUTPNFP tests are only needed to detect a LRF where *c* replaces a literal in *a~bd*. The MUTP criterion is infeasible for *a~cd* as *b* must = 1 and *e* must = 0 in an UTP for *a~cd*. However, *e* is in a single-literal term so CUTPNFP tests are only needed to detect a LRF where *b* replaces a literal in *a~cd*. Since the MUTP criterion is feasible for *e* and *e* is in a single-literal term, neither CUTPNFP tests nor a NFP are needed for *e*.

Additional tests needed for a CUTPNFP test set:
00110 term a~bd, literal a
11110 term a~bd, literal b and term a~cd, literal c
10100 term a~bd, literal d
01010 term a~cd, literal a
11000 term a~cd, literal d

Predicate 13: a + b + c + ~def~g~h + ij~l + ik~l

MUTP test set is:
100000000000 term a
100111111111 term a
010000000000 term b
010111111111 term b
001000000000 term c
001111111111 term c
000011000000 term ~def~g~h
000011001111 term ~def~g~h
000000001100 term ij~l
000111111100 term ij~l
000000001010 term ik~l
000111111010 term ik~l

The MUTP criterion is infeasible for all terms. However, CUTPNFP tests for *a*, *b*, and *c* are not needed as these literals are in single-literal terms. Likewise, CUTPNFP tests to detect a LRF where *a*, *b*, or *c* replaces a literal in another term are not needed. No CUTPNFP tests are needed for *~def~g~h* as all external literals in multi-literal terms can attain both the values 0 and 1 in an UTP for *~def~g~h*. Any NFP for each literal in *~def~g~h* can thus be generated to detect a LOF. The only external literal in a multi-literal term that cannot attain both the values 0 and 1 in an UTP for *ij~l* is *k*. Thus, CUTPNFP tests are needed to detect a LRF where *~k* replaces a literal in *ij~l*. The only external literal in a multi-literal term that cannot attain both the values 0 and 1 in an UTP for *ik~l* is *j*. Thus, CUTPNFP tests are needed to detect a LRF where *~j* replaces a literal in *ik~l*.

NFP tests:
000111000000 term ~def~g~h, literal d
000001000000 term ~def~g~h, literal e
000010000000 term ~def~g~h, literal f
000011100000 term ~def~g~h, literal g
000011010000 term ~def~g~h, literal h

Additional tests needed for a CUTPNFP test set:
000000000100 term ij~l, literal i
000000001000 term ij~l, literal j and term ik~l, literal k
000000001101 term ij~l, literal l
000000000010 term ik~l, literal i
000000001011 term ik~l, literal l

Predicate 19: acefg + ace~fh + bdefg + bde~fh

MUTP test set is:
10111110 term acefg
11101111 term acefg
10111001 term ace~fh
11101011 term ace~fh
01111110 term bdefg
11011111 term bdefg
01111001 term bde~fh
11011011 term bde~fh

The MUTP criterion is feasible for all terms so a test set of overlapping NFPs is produced.

Overlapping NFP test set is:
00111111 term acefg, literal a and term bdefg, literal b
11001111 term acefg, literal c and term bdefg, literal d
11110110 term acefg, literal e and term bdefg, literal e
11111010 term acefg, literal f and term ace~fh, literal h
term bdefg, literal f and term bde~fh, literal h
11111101 term acefg, literal g and term ace~fh, literal f and
term bdefg, literal g and term bde~fh, literal f
01101001 term ace~fh, literal a and term bde~fh, literal d
10011011 term ace~fh, literal c and term bde~fh, literal b
11110011 term ace~fh, literal e and term bde~fh, literal e

This overlap of NFPs is not possible if both CUTPNFP and MUTP tests are needed. As an example, 00111111 is used as a NFP for *a* in *acefg* and for *b* in *bdefg*, but it does not satisfy the CUTPNFP criterion for *b* in *bdefg* as it does not differ from either UTP of *bdefg* only in the value of *b*. So although 00111111 is a NFP for *b* in *bdefg*, it is not a corresponding NFP. The smallest test set size satisfying the CUTPNFP and MUTP criteria is 26, which is 10 greater than the Minimal-MUMCUT test set size.

6 Context

In order for syntactic logic criteria to be useful in practice, three separate issues need to be addressed: the internal variable problem, minimal DNF, and predicate size.

The internal variable problem is concerned with what inputs give a variable a certain value at some statement in a program. This problem is formally undecidable, but partial solutions using constraints exist [10]. For logic testing, program inputs must be found such that the predicate is reached and literal values make the faulty and original predicate evaluate to different truth values. In other words, program inputs must be such that a criterion is satisfied.

Fault detection that holds for minimal DNF predicates does not hold for non-minimal DNF predicates. This raises two issues. One, how well does the fault hierarchy hold for non-minimal DNF predicates? Two, what types of software have minimal DNF predicates? For the first issue, Yu and Lau [14] found that of a sample of 20 non-minimal DNF predicates, over 99% of the faults in Figure 3.1 were detected by tests that detected the same faults for the corresponding minimal DNF predicates. For the second issue, Chilenski [5] found that 95% of 20,256 predicates in avionics software were in minimal DNF. Only 3% of these predicates contained five or more unique literals, but 80% of these predicates were in minimal DNF.

When a predicate contains less than five unique literals the authors conjecture that exhaustive testing is best. This raises the question of what types of software generally have predicates with at least five unique literals. Chilenski and Miller [6] report that avionics software often has predicates with many literals and Chilenski [5] extracted a predicate with 77 unique literals. Thus, the Minimal-MUMCUT criterion should be useful for testing avionics software.

7 Conclusion

Logic testing needs efficient solutions to the tradeoff problem of reducing test set size without sacrificing fault detection. Several logic criteria have been proposed to address this problem, some of which are composed of other criteria. When a constituent criterion is feasible, a smaller test set satisfying it can often be used instead of a larger test set satisfying the parent criterion without sacrificing fault detection. This paper described an approach where given a minimal DNF predicate, a determination is made of which criteria are feasible for individual literals and terms. This in turn provides determination of which criteria are necessary to detect faults. The approach was examined on a sample of predicates (having from 5 to 13 unique literals) in avionics software. The results showed that a new logic criterion (Minimal-MUMCUT) reduced test set size (without sacrificing fault detection) to as little as 1.30% of the size needed if feasibility is not considered. Future research should focus on determining what inputs cause literals to have the values needed to satisfy the Minimal-MUMCUT criterion.

8 References

[1] A.T. Acree, T.A. Budd, R.A. DeMillo, R.J. Lipton, and F.G. Saywood. "Mutation Analysis," Technical Report GIT-ICS-79/08, School of Information and Computer Science, Georgia Institute Of Technology, Atlanta GA, Sept 1979.

[2] T.Y. Chen and M.F. Lau. An Empirical Study on the Effectiveness of the Greedy MUTP Criterion. Software Engineering: Education and Practice, 1998. Proceedings, 1998 International Conference. January, 1998. Pages 338 – 344.

[3] T.Y. Chen and M.F. Lau. Test Case Selection Strategies Based on Predicates. *Software Testing, Verification, and Reliability*, 11(1):165-180, November 2001.

[4] T.Y. Chen, M.F. Lau, and Y.T. Yu. MUMCUT: A Fault-Based Criterion for Testing Predicates. Software Engineering Conference, December, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific. Takamatsu, Japan. Pages 606-613.

[5] J.J. Chilenski. An Investigation of Three Forms of the Modified Condition Decision Coverage (MCDC) Criterion. Final Technical Report, DOT/FAA/AR-01/18, April 2001.

[6] J.J Chilenski and S.P. Miller. Applicability of Modified condition/decision coverage to Software Testing. *IEE/BCS Software Engineering Journal*, 9(5): 193-200, September 1994.

[7] G. Kaminski, G. Williams, and P. Ammann. Reconciling Perspectives of Logic Testing for Software. *Software Testing, Verification, and Reliability*, 18(3):149-188, September 2008.

[8] D. Richard Kuhn. Fault Classes and Error Detection Capability of Predicate Based Testing. *ACM Transactions on Software Engineering and Methodology*, 8(4): 411-424, October 1999.

[9] M.F. Lau and Y.T. Yu. An Extended Fault Class Hierarchy for Predicate-Based Testing. *ACM Transactions on Software Engineering and Methodology*, 14(3): 247-276, July 2005.

[10] A.J. Offutt and J. Pan. Automatically Detecting Equivalent Mutants and Infeasible Paths. *Software Testing, Verification, and Reliability*, 7(3):165-192, September 1997.

[11] V. Okun, P. Black, and Y. Yesha. Comparison of Fault Classes in Specification-Based Testing. Information & Software Technology, 46(8): 525-533, June 2004.

[12] Chang-ai Sun, Yunwei Dong, R. Lai, K.Y. Sim, and T.Y. Chen. Analyzing and Extending MUMCUT for Fault-based Testing of General Boolean Expressions. Proceedings of the Sixth IEEE International Conference on Computer Information Technology, September, 2006. Pages: 184-189.

[13] E. Weyuker, T. Goradia, and A. Singh. Automatically Generating Test Data from a Predicate. *IEEE Transactions on Software Engineering*, 20(5): 353-363, May 1994.

[14] Y.T Yu and M.F. Lau. Comparing Several Coverage Criteria for Detecting Faults in Predicates. In Proceedings QSIC 2004: 4th International Conference on Quality Software, Pages 14-21.

Appendix A: Logic Criteria Summary

Test Name	Guaranteed Faults Detected	Subsumes	Subsumed by	Minimum Test Set Size	Maximum Test Set Size
Multiple Near False Point (MNFP)	ENF, TNF, LNF, ORF., LOF	-	MUMCUT	When infeasibilities arise: 1. Uncertain otherwise.	$\frac{mn^2}{2}$ where *m* is the number of terms and *n* is the number of literals
Multiple Unique True Point (MUTP)	ENF, TNF, LNF, TOF, ORF+, LIF	-	Minimal-MUMCUT, MUMCUT	*m* to *2m* where *m* is the number of terms	2*m(n-1)* where *m* is the number of terms and *n* is the number of literals
Corresponding Unique True Point Near False Point (CUTPNFP)	ENF, TNF, LNF, TOF, ORF., ORF+, LOF	-	MUMCUT	$\sum_{i=1}^{m} n_i + 1$ where n_i is the number of literals in term *i* and *m* is the number of terms	2*mn* where *m* is the number of terms and *n* is the number of literals
Minimal-MUMCUT	ENF, TNF, LNF, TOF, ORF., ORF+, LOF, LIF, LRF	MUTP	MUMCUT	*m + 1* to *2m + 1* where *m* is the number of terms	Uncertain, but less than $2m(n-1) + \frac{mn^2}{2}$ where *m* is the number of terms and *n* is the number of literals
MUMCUT	ENF, TNF, LNF, TOF, ORF., ORF+, LOF, LIF, LRF	MUTP, MNFP, CUTPNFP, Minimal-MUMCUT	-	When infeasibilities arise: *m* to 2*m + 1* where *m* is the number of terms. Uncertain otherwise.	$2m(n-1) + \frac{mn^2}{2}$ where *m* is the number of terms and *n* is the number of literals

Appendix B: TCAS Boolean Predicates in Minimal DNF

1. a~bd~e~h~f + a~b~de~h~f + a~bcd~e~f + a~bc~de~f + ~ab~de~f

2. a~bc~d~e~gh~i~f + a~b~d~e~g~h~if + a~b~c~e~g~h~if + a~b~c~d~g~h~if + a~bc~d~eg~h~f + a~bc~d~e~hi~f + a~b~cd~eg~h~f + a~b~cd~e~hi~f + a~b~c~deg~h~f + a~b~c~de~hi~f + ~abc~d~e~hi~f + ~ab~cd~e~hi~f + ~ab~c~de~hi~f

3. ~a~bc~g~i~k~m + ~a~bcg~h~l~m + ~a~bc~g~hi~m + ~a~bcgi~l~m + ~a~bcgi~k~m + ~a~bc~h~k~m + ~ab~c~g~i~k + a~b~c~g~i~k + ~a~bc~i~kf + ~ab~c~g~hi + ~ab~cg~h~l + a~b~c~g~hi + a~b~cg~h~l + ~a~bc~hif + ~ab~cgi~k + ~ab~cgi~l + a~b~cgi~k + a~b~cgi~l + a~b~c~h~k + ~ab~c~h~k + a~b~cgf + ~ab~cgf + ~a~bcgf + a~b~c~d + a~b~c~e

4. a~bd + a~cd + e

5. a~g~i~k + ag~h~l + a~g~hi + agi~l + agi~k + a~h~k + a~c + a~b + f

6. ~ab~cdeg~hij~k~f + a~bc~deg~hij~k~f + ~ab~cde~g~h~jf + ~ab~cde~g~h~kf + a~bc~de~g~h~jf + a~bc~de~g~h~kf

7. ~ab~cde~g~i~j + ~ab~cde~h~i~k + a~bc~de~g~i~j + a~bc~de~h~i~k + a~bc~de~g~k + a~bc~de~h~j + ~ab~cde~g~k + ~ab~cde~h~j

8. ~ab~cde~gh~f + a~bc~de~gh~f + ~ab~cdeg~hf + a~bc~deg~hf

9. ~a~b~cd~e~gf + ~abc~d~e~gf

10. a~b~cd~eg~j~l~mf + a~b~cd~eh~j~l~mf + a~b~cd~ei~j~l~mf + a~b~cd~egj~k~mf + a~b~cd~ehj~k~mf + a~b~cd~eij~k~mf

11. a~b~c~g~h~i~j~l + a~b~c~g~h~ij~k + a~b~c~g~h~i~jm + a~b~c~d~e~j~l + a~b~c~d~e~jm + a~b~c~d~ej~k + a~b~c~j~l~f + a~b~cj~k~f + a~b~c~jm~f

12. Not included due to a missing right parenthesis

13. a + b + c + ~def~g~h + ij~l + ik~l

14. ae~h + ad~h + ace + acd + be + bf

15. bei + bdi + bci + aei + aeg + adi + adg + aci + ach + acg + af

16. c~g~i~k~m + cg~h~l~m + c~g~hi~m + cgi~l~m + cgi~k~m + c~h~k~m + b~g~i~k + a~g~i~k + b~g~hi + bg~h~l + a~g~hi + ag~h~l + bgi~k + bgi~l + agi~k + agi~l + a~h~k + b~h~k + ~i~kf + ~hif + gf + a~e + a~d

17. acegij + acehik + bdegij + bdehik + acef + bdef

18. ace~j~k + ace~h~j + ace~g~k + bde~j~k + bde~h~j + bde~g~k + bde~i + ace~i

19. aceh~f + bdeh~f + acegf + bdegf

20. ~a~bd~e~gf + ~abc~e~gf