Applicability of modified condition/decision coverage to software testing

by John Joseph Chilenski and Steven P. Miller

Modified condition/decision coverage is a structural coverage criterion requiring that each condition within a decision is shown by execution to independently and correctly affect the outcome of the decision. This criterion was developed to help meet the need for extensive testing of complex Boolean expressions in safety-critical applications. The paper describes the modified condition/decision coverage criterion, its properties and areas for further work.

1 Introduction

The verification and validation of high integrity software within critical applications such as commercial avionics, banking, medical instrumentation and nuclear power control systems is a complex process, consisting of many different activities employed throughout the development life-cycle. One such activity is the low-level testing of individual components, often referred to as unit or module testing, to verify the implementation of the software low-level requirements (specifications). This level of testing may be performed on either stand-alone components or integrated components, depending on the controllability of inputs and observability of outcomes [1].

Module testing can be based on either structural (white box) or functional (black box) criteria, or a combination of the two. In structural testing, test cases are designed based on the internal structure of the program, using the specification to predict the required outcome of each test. In contrast, functional tests are derived from a specification of the program's requirements without regard to the program's internal structure [2–5]. Each approach has its own strengths and weaknesses [6, 7].

An alternative approach that exploits strengths of both techniques is to develop functional tests from the specification and measure coverage against a structural criterion. In this way, the structural criterion is utilised as a *test data adequacy criterion*, rather than a *test data selection criterion* [8].

Different criteria have been proposed for structural coverage, ranging from the requirement that all statements are executed at least once to the requirement that all feasible paths are executed [2, 4]. Recent work has focused on data-flow criteria [9–11] that examine the interactions between definitions and references through the program flow. In many of these criteria, a decision is treated as a single node in the program's structure, regardless of the complexity of the Boolean expression constituting the decision.

For critical real-time applications where over half of the executable statements may involve Boolean expessions, the complexity of the expression is of concern. Modified condition/decision coverage was developed to address the concerns of testing Boolean expressions and can be used to guide the selection of test cases at all levels of specification, from initial requirements to source code. When applied to source code, it can be combined with an analysis of the generated code sequences to ensure that the requirement-based tests execute each statement of object code. Compliance with this criterion, in concert with several other analyses, is required for all flight-critical commercial avionics [12]. This paper describes the modified condition/decision coverage criterion, its properties and relationship to other criteria, and areas for further work.

2 Background

2.1 Notation

The notation used within this paper is based on the laws of Boolean algebra as found in standard texts on discrete mathematics [13] or switching theory [14, 15]. Boolean values will be written as **true** and **false** or **T** and **F**, depending on the context. If $x_1, \ldots, x_n$ are Boolean variables, a function $f(x_1, \ldots, x_n)$ is a Boolean *function* if it is also Boolean valued.

Boolean *expressions* are built from Boolean values, variables, the unary opertor **not**, and the binary operators **and, or, xor,** = and /=. The **and, or,** and **xor** operators have the lowest precedence, and are associated from left to right, = and /= have a higher precedence and are also associated left to right, and **not** has the highest precedence. If dissimilar binary operators (**and, or xor**) are used, they must be separated by parentheses. For expressions coded in Ada [16], the short-circuit operators **and then** and

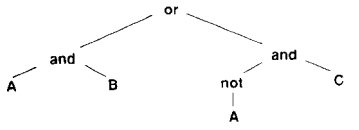

Fig. 1 ((A and B) or (not (A) and (C))

or else and the inequality operators **<, <=, >, >=** are also allowed.

A *condition* is a Boolean valued expression that cannot be broken down into simpler Boolean expressions. Each leaf in the parse tree of an expression is a condition. Thus, although the Boolean expression of Fig. 1 has three variables (**A, B** and **C**), it has four conditions corresponding to the variables **b, c,** and the two instances of **A**. The two instances of **A** are referred to as *coupled* conditions (Section 3.2) as they cannot be varied independently.

For simplicity, Boolean variables will be used to represent the actual conditions found in a source program. For example, the expression (Fuel_Quantity_Valid and Value_To_Display = 0) would be written as **(A and B)**, where **A** corresponds to Fuel_Quantity_Valid and **B** to Value_To_Display = 0.

The distinction between Boolean functions and expressions is significant as two Boolean expressions can represent the same Boolean function (e.g. **not (A and B)** and **(not A) or (not B)**), yet be compiled into dramatically different object code (Section 3.2).

2.2 *Relationship to other work*

Many methods for comparing testing criteria have been proposed [17]; the most popular is the *subsumes* relationship. Criterion A is said to subsume criterion B if, and only if, every test set that satisfies A also satisfies B. Subsumption hierarchies for several structural criteria have been described in the literature [9]. Fig. 2 relates some of the best known control, data flow and path criteria.

Four of the five control-based criteria (statement, decision, condition/decision and multiple-condition) are well known [4]. The modified condition/decision criterion is not as well known but has been used for several years within the commercial avionics industry. Definitions for each coverage criterion are given in Fig. 3.

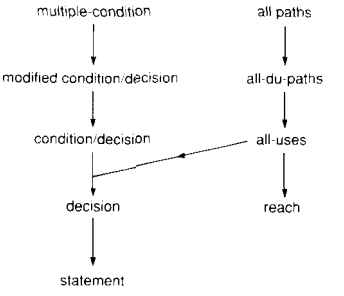

Fig. 2 Subsumption hierarchy

Of the five control-based criteria, only three (condition/decision, modified condition/decision and multiple-condition) are sensitive to the complexity of an individual decision. The condition/decision criterion is the least sensitive of the three and is inadequate for high-integrity applications, both because it does not ensure decision coverage of the object code and because incorrect evaluation of a condition can be masked by the other conditions [4]. Intuitively, this is reinforced by the observation that the number of tests required to achieve condition/decision coverage rather than decision coverage is usually negligible.

Multiple-condition coverage is the most sensitive to the complexity of Boolean expressions and addresses these shortcomings by requiring that each possible combination of conditions is tested at least once [2, 4]. If each condition is viewed as a single input, then multiple-condition coverage is analogous to exhaustive testing of the inputs and demonstrates that a Boolean expression is correctly implemented by the object code and hardware. Unfortunately, the number of tests required grows exponentially with the number of conditions, i.e. N conditions require 2^N tests. Consequently, this form of coverage is practical only if N is small.

For many domains, N is small and multiple-condition coverage is feasible. For example, a check of the decisions in the Software Tools In Pascal [18] and 496 modules in the Booch Components [19] shows that the vast majority consists of a single condition, several consist of two conditions, and a few consist of three conditions (Fig. 4). In the

statement coverage (SC) — *every statement in the program has been executed at least once [12].*

decision coverage (DC) — *every point of entry and exit in the program has been invoked at least once, and every decision in the program has taken all possible outcomes at least once [12].*

condition/decision coverage (C/DC) — *every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, and every decision in the program has taken all possible outcomes at last once [12].*

modified condition/decision coverage (MC/DC) — *every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken on all possible outcomes at least once, and each condition has been shown to independently affect the decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions [12].*

multiple-condition coverage (M-CC) — *every point of entry and exit in the program has been invoked at least once, and all possible combinations of the outcomes of the conditions within each decision have been taken at least once [4].*

reach coverage — *at least one definition clear subpath from each definition to each reference for each object has been executed [9].*

all-uses coverage — *at least one definition clear subpath from each definition to each reference and each successor of the reference for each object has been executed [9].*

all-du-paths coverage — *all definition clear subpaths from each definition to each reference and each successor of the reference for each object have been executed [9].*

Fig. 3 Definitions of coverage criteria

number of decisions with N conditions

N	1	2	3	4	5	6–10	11–15	>15
Software Tools	446	72	9	0	0	0	0	0
Booch Components	9048	402	52	0	0	0	0	0
EFIS avionics display	1343	182	38	16	18	11	3	0

Fig. 4 Complexity of expressions in representative domains

avionics domain, complex Boolean expressions are much more common. Fig. 4 shows the distribution for a typical electronic flight instrumentation system (EFIS) consisting of 34 295 lines of Ada and comments. It is not uncommon to find decisions with 30 or more conditions, requiring 2^{30} tests for a single decision. As tests for certification of flight-critical avionics software must be executed on the actual target, multiple-condition coverage is clearly impractical for such expressions.

The modified/condition decision criterion has been developed to address this issue. Foster [20] presents a test generation algorithm that is close to modified condition/decision coverage. Tai and Su [21] extended Foster's work with two test data generation algorithms; one that produces fewer tests than required by the modified condition/decision criterion, and one that produces more tests than are required.

Both of these studies excluded the **XOR** operator from their analysis. Tai and Su also limited their analyses to singular Boolean expressions (SBEs) in which each operand is a simple Boolean variable that appears only once, i.e. coupled conditions were not addressed (Section 3.4). Fig. 5 shows the number of Boolean functions expressible as SBEs with the operations **and, or** and **not,** with the operations **and, or, not,** and **xor,** and the total number of Boolean functions possible for up to four operands. As can be seen, SBEs account for a very small portion of the function space.

The data-flow criteria [9–11] were developed to detect errors in data usage and concentrate on the interactions between variable definition and reference. In contrast, the control criteria of condition/decision, modified condition/decision, and multiple-condition focus on detecting errors in the coding and translation of Boolean expressions into object code. As indicated in Fig. 2, the data-flow criteria of all-uses and all-du-paths do not subsume these three control-based criteria. Although this has been identified as a potentially misleading comparison [17], it is important to recognise that the data-flow criteria do not ensure condition-sensitive tests, and *vice versa*. The data-flow methods are sensitive to certain classes of faults, whereas the control criteria are sensitive to others. A promising area for future work is the development of criteria that combine both approaches to improve their fault effectiveness.

number of operands

	1	2	3	4
$SBE_{(and, or, not)}$	2	8	64	832
$SBE_{(and, or, xor, not)}$	2	10	114	2,154
Boolean functions	4	16	256	65,536

Fig. 5 Growth of Boolean functions

3 Modified condition/decision coverage

3.1 Definition

The modified condition/decision coverage criterion was developed to achieve many of the benefits of multiple-condition testing while retaining the linear growth in required test cases of condition/decision testing. The essence of the modified condition/decision coverage criterion (Fig. 3) is that each condition must be shown to independently affect the outcome of this decision, i.e. one must demonstrate that the outcome of a decision changes as a result of changing a single condition. Consider the expression **A and B**. From the truth table in Fig. 6, it can be shown that the **(T,T)** test is required as it is the only one that returns **true**. The **(F,T)** test is required as it is the only test that changes the value of only **A** and also changes the decision's outcome, thereby establishing the independence of **A**. In similar fashion, the **(T,T)** and **(T,F)** tests are required to show the independence of **B**. Therefore, the test set **{(T,T), (T,F), (F,T)}** is required to satisfy modified condition/decision coverage for the expression **A and B.**

This information can be shown concisely by extending the truth table of Fig. 6 to the pairs table of Fig. 7. The first column indexes the test cases, and the columns labelled **A** and **B** indicate which test cases can be used to show the

A B	result
T T	T
T F	F
F T	F
F F	F

Fig. 6 Truth table for A and B

number	A B	result	A	B
1	T T	T	3	2
2	T F	F		1
3	F T	F	1	
4	F F	F		

Fig. 7 Pairs table for A and B

number	A B	result	A	B
1	T T	T		
2	T F	T	4	
3	F T	T		4
4	F F	F	2	3

Fig. 8 Pairs table for A or B

number	A B	result	A	B
1	T T	F	3	2
2	T F	T	4	1
3	F T	T	1	4
4	F F	F	2	3

Fig. 9 Pairs table for A xor B

independence of the respective condition. For example, Fig. 7 shows that the test case 1 **(T,T)** can be paired with test case 3 **(F,T)** to show the independence of **A** and that test case 1 **(T,T)** can be paired with test case 2 **(T,F)** to show the independence of **B**. Test cases 2 and 4 are of no value in showing the independence of **A**, and test cases 3 and 4 are of no value in showing the independence of **B**. A similar analysis is shown for the expression **A or B** in Fig. 8 and **A xor B** in Fig. 9. Note that in the case of **A xor B**, there are four minimal tests, {1,2,3}, {1,2,4}, {1,3,4} or {2,3,4}.

To use the pairs table to design a coverage-compliant test set requires that a pair of tests is chosen for each condition. The size of the test set can be minimised by overlapping these pairs. A minimum of $N + 1$ tests (where N is the number of conditions) can usually be achieved. Notice that when a pairs table row has entries in all columns, it defines a minimal test set.

3.2 *Object code coverage*

Unlike condition/decision coverage in which the behaviour of a condition can be masked by the other conditions, modified condition/decision coverage requires that each condition is shown to correctly affect the decision's outcome. If the individual conditions are not coupled (i.e. they can be varied independently of each other) and the compiler performs a straightforward translation of each condition into a single decision (conditional jump) in the object code, this also ensures decision coverage of the underlying object code. For example, Fig. 10 shows four possible compiler implementations, or *predicate graphs*, of the expression **A and (B or C)** under this assumption.

The pairs table for this expression is given in Fig. 11. From Fig. 11, it can be seen that the minimal modified condition/decision coverage test sets for the expression are {2,3,4,6} or {2,3,4,7}. Examination of the four predicate graphs in Fig. 10 shows that both of these test sets provide decision (edge) coverage in all four graphs. The minimal test sets for condition/decision coverage are {1,8}, {2,7} or {3,6}. Examination of the four predicate graphs shows that none of the condition/decision coverage test

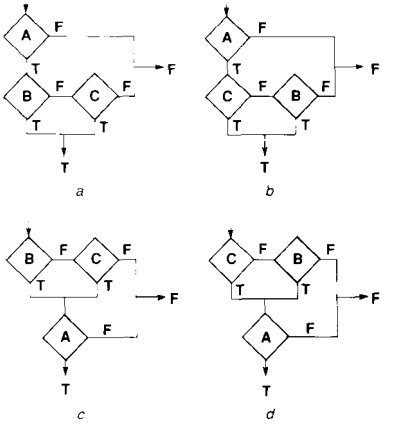

Fig. 10 Possible object code implementations of A and (B or C)

sets provides decision coverage of the generated object code. Further examination also shows that the traversed paths vary for the different test sets and predicate graphs.

It should be noted that decision coverage of the object code does not imply modified condition/decision coverage of the source code. For example, in Fig. 10c, it is possible to obtain decision coverage of the object code with three tests (e.g. tests 1, 7, and 8 of Fig. 11) whereas modified condition/decision coverage requires at least four tests for this expression.

3.3 *Extensions for short-circuit operators*

In addition to the logical operators **and, or, xor, =, /=,** and **not**, the Ada programming language [16] supports the short-circuit operators **and then** and **or else**. When these operators are used, the left operand is always evaluated first. If the left operand in an **and then** construct evaluates to **false**, the right operand is not evaluated and the entire expression evaluates to **false**. In a similar fashion, if the left operand in an **or else** construct evaluates to **true**, the right operand is not evaluated and the entire expression evaluates to **true**. The most common use of short-circuit operators is to preclude evaluation of undefined conditions. For example, give the expression

if x/= 0 and then y/x > 15 then ...

number	ABC	result	A	B	C
1	TTT	T	5		
2	TTF	T	6	4	
3	TFT	T	7		4
4	TFF	F		2	3
5	FTT	F	1		
6	FTF	F	2		
7	FFT	F	3		
8	FFF	F			

Fig. 11 Pairs table for A and (B or C)

the object code generated by the compiler will ensure that the second condition is never evaluated when the first condition is **false**.

Use of short-circuit operators has two effects on the modified condition/decision criterion. First, it reduces the number of possible paths through the object code that need to be considered. Secondly, it forces relaxation of the requirement that all conditions are held fixed while the condition of interest is varied. For example, in the above expression, it is impossible to create a test in which the first condition is **false** and the second takes on either of the values **true** or **false** because **y/0** is undefined.

Conditions that are not evaluated for a specific test due to the presence of short-circuited operators are said to be *insignificant* for that test and have no effect on the path traversed through the object code. As a result, the use of short-circuit operators partitions the set of all possible test cases into classes of equivalent test cases.

For example, in the expression **A and then (B or C)**, conditions B and C are insignificant whenever A is **false**. As a result, three of the possible eight test cases are equivalent and will execute the same path in the object code. In fact, the predicate graphs for this expression are now limited to graphs *a* and *b* of Fig. 10. The pairs table for this expression is shown in Fig. 12.

In Fig. 12, the last four tests from Fig. 11 are folded into the class of equivalent tests 5, where a dash indicates the value of the condition is insignificant. As the values assigned to **B** and **C** do not affect the path executed in the object code, any test in which **A** is **false** can be paired with test case 1, 2, or 3 to show the independence of **A**. It is not necessary that **B** and **C** are held fixed while **A** is varied, both because this may be impossible and because **A** is evaluated to **true** in test cases 1, 2, and 3 and to **false** in any test from class 5. Note that it is not sufficient simply to pair a test from class 5 with a test in which the value of **A** has been varied. For example, test case 4 varies the value of **A** to **true**, but does not change the overall outcome of the decision.

The pairs table for the expession **A and then (B or else C)** is given in Fig. 13. Notice that the predicate graph for this expression is limited to graph *a* of Fig. 10, and that tests 1 and 2 from Fig. 12 are folded into the class of equivalent tests 1.

3.4 *Extensions for coupled conditions*

Conditions that cannot be varied independently are said to be *coupled*. Two or more conditions are *strongly* coupled if varying one always varies the others, and *weakly* coupled if varying one sometimes, but not always, varies

number	test	result	A	B	C
1	TTT	T	5		
2	TTF	T	5	4	
3	TFT	T	5		4
4	TFF	F		2	3
5	F--	F	1,2,3		

Fig. 12 A and then (B orC)

number	test	result	A	B	C
1	TT-	T	4	3	
2	TFT	T	4		3
3	TFF	F		1	2
4	F--	F	1,2		

Fig. 13 A and then (B or else C)

the others. For example, the two conditions in **(0 < I) and (I < 10)** are weakly coupled as changing the value of I from 5 to 10 varies the second condition but not the first, whereas changing the value of I from 0 to 10 varies both conditions. Coupled conditions make some tests infeasible and can be dealt with by removing the infeasible rows from the pairs table. Of course, this may make it more difficult to meet the modified/condition decision criterion because pairs of tests may be more difficult to find or more than $N + 1$ tests are required.

Weakly coupled conditions are not normally a concern because either the infeasible tests play no role in achieving coverage, an alternative test set exists, or the affected tests are still feasible but more difficult to create. For example, in the preceding expression the test **(F,F)** is not possible, but is of no value in meeting the criterion.

However, strongly coupled conditions pose a dilemma because it is not possible to vary one condition while holding all others fixed. One interpretation, referred to as *weak* modified condition/decision coverage, is to treat the strongly conditions as a single condition. For example, the expression **(A and not(B)) or (Not(A) and B)** would be viewed as having only two conditions, **A** and **B**. This is analogous to the black box approach of module testing. Unfortunately, it yields a weak set of tests that fail to guarantee decision coverage of the object code.

For example, under this interpretation, the above expression can be covered from the same test set that covers **(A and not (B))**. In effect, an acceptable test set exists that would be insensitive to the second half of the expression. An advantage of this interpretation of the modified condition/decision criterion is that a compliant test set always appears to exist for non-degenerate expressions, i.e. those that cannot be simplified to an equivalent expression with fewer operands. We have empirically shown that, for two to five operands, there always exists such a modified condition/decision test set.

An alternative interpretation, referred to as *strong* modified condition/decision coverage, is to view each condition as a distinct entity to be varied, but to find tests in which the effect of varying one instance of a variable affects the overall outcome while simultaneously masking the effects of all other instances of the variable. This requires a stronger set of test cases that ensure decision coverage of the object code, given a straightforward translation by the compiler. For example, in the parse tree of Fig. 1, holding the value of **C false** ensures that the second instance of **A** is masked and cannot affect the overall outcome. The independence of the first instance of **A** can then be shown by test cases **TTF** and **FTF**. In a similar fashion, the independence of the second instance of **A** can be shown by test cases **TFT** and **FFT** as holding **B false** masks the first

number	ABC	result	A_1	B	A_2	C
1	TTT	T		3		
2	TTF	T	6	4		
3	TFT	F		1	7	
4	TFF	F		2		
5	FTT	T				6
6	FTF	F	2			5
7	FFT	T			3	8
8	FFF	F				7

Fig. 14 ((A_1 and B) or (not(A_2) and C))

instance of **A**. The pairs table for this expression is given in Fig. 14, where each instance of **A** is shown as a separate column.

As its name suggests, strong modified condition/decision coverage is more difficult to achieve than weak modified condition/decision coverage in the presence of coupled conditions. In some cases, more than $N + 1$ test cases may be required. However, if an expression can be tested, a maximum of $2N$ (two for each condition) tests will always suffice.

Some expressions with coupled conditions cannot be tested at all under the strong interpretation of the modified condition/decision coverage criterion. For example, no modified condition/decision coverage test set exists that can show the independence of either instance of **A** in the expression **(A or B) and (B or (A xor C))**, whose pairs table is given in Fig. 15. Depending on the compiler, such expressions may have branches that can never be taken in the underlying object code. They can usually be rewritten into simpler expressions that can be tested under the strong interpretation. For example, the above expression can be rewritten as the simpler and testable expression **(A and not C) or B**. It is not known if such expressions can always be rewritten into a simpler and testable form.

3.5 *Advantages of the MC/DC criterion*

Some of the advantages of the modified condition/decision criterion (MC/DC) have already been mentioned. Here we discuss its benefits further.

number	ABC	result	A_1	B_1	B_2	A_2	C
1	TTT	T			3		
2	TTF	T					
3	TFT	F			1		4
4	TFF	T					3
5	FTT	T		7			
6	FTF	T					
7	FFT	F		5			
8	FFF	F					

Fig. 15 (A_1 or B_1) and (B_2 or (A_2 or C))

3.5.1 Linear growth in required tests: the modified condition/decision coverage criterion provides a linear and practical rate of growth in the number of tests required as the number of conditions in a Boolean expression increases. For an expression with N uncoupled conditions, the modified condition/decision criterion can be met with a minimum of $N + 1$ tests by varying exactly one condtion of each of the first N tests. If this is not possible because of coupled conditions, the criterion can be met with a maximum of $2N$ tests by selecting unique tests for each condition. Although this is greater than the number of tests required for condition/decision coverage, it is still linear and avoids the exponential growth of multiple-condition testing.

3.5.2 Object code coverage: as described in Section 3.2, the modified condition/decision criterion ensures a much higher level of decision coverage of the object code than either decision or condition/decision coverage. Of course, to ensure complete decision coverage of the underlying object code, we must also taken into account any jumps inserted by the compiler that are not directly traceable to a single condition in the source code.

3.5.3 Operand sensitivity: whereas the modified condition/decision coverage criterion is more sensitive to operand errors than either decision or condition/decision coverage, it is particularly sensitive to missing or extraneous operands. Operands that should affect the outcome, but cannot because of a form of translation error, are found directly during testing, as the criterion requires the tester to find, for each condition, two tests in which only the operand of interest is varied and shown to change the outcome of the overall expression. Extraneous operands that cannot independently affect the outcome are found during construction of the test set (e.g. Fig. 15).

3.5.4 Sensitivity to non-equivalent functions: the modified condition/decision coverage criterion is more sensitive to errors in the encoding or compilation of a single operand than decision or condition/decision coverage. The sensitivity of the five control-based criteria to errors in the implementation of a Boolean function can be quantified in the following manner. For N Boolean operands, there are 2^N possible combinations of the operand values and 2^{2^N} possible Boolean functions [14]. For a given function of N operands and any M distinct tests, there are $2^{(2^N - M)} - 1$ other functions that are indistinguishable, i.e. produce the same outcome, for the M tests. If M is fixed at $N + 1$, the number of indistinguishable functions grows exponentially with N.

Fortunately, the number of distinguishable functions (possible – indistinguishable) also grows rapidly. Given any M distinct tests, the probability $P_{(N,M)}$ of detecting an error in an incorrect implementation of a Boolean expression with N conditions is given by

$$P_{(N,M)} = 1 - \left[\frac{2^{(2^N - M)} - 1}{2^{2^N}}\right]$$

This is plotted as a function of the number of tests for three values of N in Fig. 16. Three points should be noted. First is the relatively low probability of detecting errors with only two test cases, as normally required in decision or

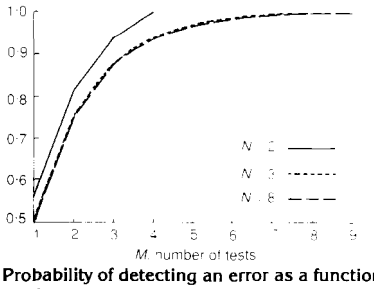

Fig. 16 Probability of detecting an error as a function of tests executed

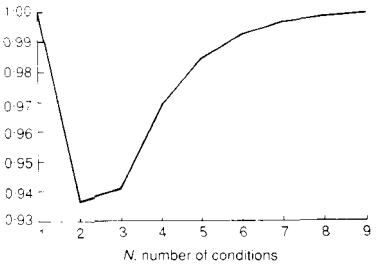

Fig. 17 Probability of detecting an error with modified condition/decision

condition/decision testing. Second is the rapid increase in probability of detecting errors as M increases. This suggests that much of the benefit of the modified/condition decision criterion lies not so much in the specific tests selected as in the requirement that at least $N + 1$ tests are executed. The actual probabilities agree very well with those reported by Tai and Su [21] for their test generation algorithms.

Thirdly, as N grows, $P_{(N,M)}$ rapidly converges to $1 - 1/2^M$ and the sensitivity changes only marginally with N. This is reflected in the very small difference in the sensitivity curves for the cases $N = 3$ and $N = 8$ in Fig. 16. As the modified/condition decision criterion always requires $N + 1$ tests, the probability of detecting an error actually increases as N increases. Fig. 17 shows the increasing likelihood of detecting an error in an expression of N conditions with $N + 1$ tests as N increases. This non-intuitive result occurs because the dominant factor (the number of tests) increases with N while the sensitivity to errors remains relatively stable.

The question is raised as to whether complex Boolean expressions should be broken down into a number of smaller sub-expressions that are then recombined to simplify testing. Often this will reduce the number of tests required for the entire module as tests for one decision can be reused in the testing of another decision. Fig. 17 suggests that the important quality is the total number of tests executed and that it may be more appropriate to *not* decompose complex Boolean expressions unless doing so substantially improves readability and maintainability.

4 Conclusions and future directions

The modified/condition decision criterion fills an important gap in the control-based criteria previously defined, providing better coverage of the underlying object code than condition/decision testing without incurring the exponential growth in test cases required by multiple-condition testing. We have described the modified condition/decision coverage criterion, its properties and relationship to other criteria. We have also identified problems posed by coupled conditions and the Ada short-circuit operators, and proposed approaches for each. Finally, a probabilistic analysis was used to compare the sensitivity of the modified condition/decision criterion with that of decision, condition/decision, and multiple-condition testing in detecting errors in Boolean expressions. The modified/condition decision criterion was shown to be significantly better than either the decision or condition/decision criterion and to compare favourably with the often impractical multiple-condition criterion.

There are many areas for additional work, two of which we discuss below.

4.1 Extensions for specific faults

The sensitivity of modified condition/decision coverage to specific faults should be investigated further. Although this does not seem to be warranted by the analysis in Section 3.5.4, it is possible that simple extensions could be made to the criterion that would improve its sensitivity to the most common classes of faults. Another approach would be to develop coding standards or tools to eliminate the insertion of such faults in the first place.

4.2 Extensions for data-flow testing

The combination of modified condition/decision coverage and the data-flow coverage criteria also needs to be investigated further. Modified condition/decision coverage concentrates on faults that may be introduced in Boolean expressions. Data-flow coverage concentrates on faults that may be introduced into the interactions of variables' definitions and references. Effective testing of programs with complex Boolean expressions requires the examination of both error classes. The simultaneous application of data-flow and modified condition/decision coverage would be a step in that direction.

5 References

[1] HOFFMAN, D.: 'Hardware testing and software ICs'. Proc. Pacific Northwest Software Quality Conf., Portland, Oregan, September 1989, pp. 234–244

[2] BEIZER, B.: 'Software testing techniques' (Van Nostrand Reinhold, New York, 1990) 2nd edn.

[3] HOWDEN, W.: 'Functional program testing', *IEEE Trans.*, 1980, **SE-6,** (2), pp. 162–169

[4] MYERS, G.: 'the art of software testing' (John Wiley and Sons, New York, 1979)

[5] YOUNGBLUT, C., BRYKEZYNSKI, B., SALASIN, J., GORDON, K., and MEESON, R.: 'SDS software testing and evaluation: a review of the state-of-the-art in software testing and evaluation with recommended R&D tasks'. IDA Paper P-2132, US Insitute for Defense Analyses, February 1989

[6] BASILI, V., and SELBY, R.: 'Comparing the effectiveness of software testing strategies', *IEEE Trans.*, 1987, ***SE*-13,** (12), pp. 1278–1296

[7] HOWDEN, W.: 'Theoretical and empirical studies of program testing, *IEEE Trans.*, 1978, **SE-4,** (4), pp. 162–169

[8] WEYUKER, E.J., WEISS, S.N., and HAMLET, D.: 'Comparison of program testing strategies'. Proc. Symp. on Testing, Analysis, and Verification (TAV4), Victoria, British Columbia, October 1991, pp. 1–10
[9] CLARK, L.A., POSGURSKI, A., RICHARDSON, D.J., and ZEIL, S.J.: 'A formal evaluation of data flow path selection criteria', *IEEE Trans.*, 1989, **SE-15,** (11), pp. 1318–1332
[10] NTAFOS, S.: 'A comparison of some structural testing strategies', *IEEE Trans.*, 1988, **SE-14,** (6), pp. 868–874
[11] RAPPS, S., Weyuker, E.: 'Selecting software test data using data flow information', *IEEE Trans.*, 1985, **SE-11,** (4), pp. 367–375
[12] 'Software considerations in airborne systems and equipment certification'. Document RTCA/DO-178B, RTCA, Inc., December 1992
[13] GRIES, D.: 'The science of programming' (Springer-Verlag, New York, 1981)
[14] KOHAVI, Z.: 'Switching theory and finite automata theory' (McGraw-Hill Book Company, New York, 1978)
[15] McCLUSKEY, E.: 'Logic design principles' (Prentice-Hall, New Jersey, 1986)
[16] 'Reference manual for the Ada programming language' ANSI/MIL-STD-1815A-1983, United States Department of Defense & American National Standards Institute, Inc., February 1983
[17] HAMLET, R.: 'Theoretical comparison of testing methods'. Proc. Third Symp. on Testing, Analysis, and Verification, Key West, Florida, December, 1989, pp. 28–37
[18] KERNIGHAN, B.W., and PLAUGER, P.J.: 'Software tools in Pascal' (Addison Wesley, Reading, Massachusetts, 1981)
[19] BOOCH, G.: 'Software components with Ada' (Benjamin/Cummings, California, 1987)
[20] FOSTER, K.A.: 'Sensitive test data for logic expressions', *ACM SIGSOFT Softw. Eng. Notes*, 1984, **9,** (2), pp. 120–125
[21] TAI, K-C., and SU, H-K.: 'Test generation for Boolean expressions'. Proc. 11th Int. Symp. on Computer Software and Applications (COMPSAC '87), Tokyo, Japan, ISSN 0730-3157, October 1987, pp. 278–283

The paper was first received 14 October 1993 and in revised form 29 July 1994.

John Joseph Chilenski is with Boeing Commercial Airplane Group, PO Box 3707, M/S 6H-TW, Seattle WA 98124-2207, USA; Steven P. Miller is with Collins Commercial Avionics, Rockwell International, Cedar Rapids IA 52498, USA.