Formalization and Comparison of MCDC and Object Branch Coverage Criteria

Cyrille Comar, Jérome Guitton, Olivier Hainque, Thomas Quinot

Formalization and Comparison of MCDC and Object Branch Coverage Criteria

Cyrille Comar, Jerome Guitton, Olivier Hainque, Thomas Quinot
AdaCore, 46 rue d'Amsterdam, F-75009 PARIS (France)
{comar, guitton, hainque, quinot}@adacore.com

Abstract

This paper presents formal results derived from the COUVERTURE project, whose goal was to develop tools to support structural coverage analysis of uninstrumented safety-critical software. After briefly introducing the project context and explaining the need for formal foundations, we focus on the relationships between machine branch coverage and the DO-178B *Modified Condition/Decision Coverage* (MCDC) criterion. A thorough understanding of those relationships is important, since it provides the foundation for knowing where efficient execution trace techniques can be used to demonstrate compliance with the MCDC criterion. We first present several conjectures that were tested using Alloy models, then provide a formally verified characterization of the situations when coverage of object control-flow edges implies MCDC compliance.

Keywords: Structural coverage, MCDC, formalization, coverage criteria comparison, BDD, DO-178, certification

1 Introduction

Structural coverage analysis can be described as a metric assessing how thoroughly pieces of a software program were exercised by a testing campaign. Several criteria exist both at the object and at the source level. The *statement coverage* criterion, for example, focuses on source statements with metrics such as "proportion of executed statements" and information on which ones were not executed. Coverage analysis is typically mandated by safety critical certification processes, such as the DO-178B standard in the civil avionics domain. It usually involves some sort of instrumentation, most often of the program itself to output information about paths taken at run time, or using hardware probes to fetch traces out of the target execution environment. COUVERTURE's main idea was to leverage on the QEMU [1] emulator to produce machine-level execution traces and to serve as the core of a modern Free Software analysis framework. Relying on object-level traces permits operating on uninstrumented target code, very close to what goes on board. It also reduces dependencies on programming languages, compared to techniques based on source instrumentation, none of which supported Ada 2005 when we started in 2008 for example. Furthermore, unlike hardware probes, the simulator runs entirely on development hosts and is replicable at will, facilitating usage of Agile techniques such as *continuous integration*.

One challenge remains, however: the need to determine valid mappings between the machine-level execution traces and the user-oriented criteria defined by certification standards. Of particular concern in our context are the relationships between the coverage status of machine conditional branch instructions and the *Modified Condition/Decision Coverage* (MCDC) criterion of DO-178B.

For a long time it was commonly accepted that achieving full Object Branch Coverage (OBC), so that all the branch instructions were taken both ways, was sufficient to claim MCDC when Boolean operators are restricted to short-circuit forms. This was exploited in [12] for example. Unfortunately, this admitted informal equivalence was shown not to hold in the general case [10] and it is now clear that a rigorous in-depth analysis is called for.

The following section summarizes the related work and contributions of this paper in this area. Section 3 presents precise definitions of MCDC and branch/edge coverage criteria, together with additional background for the remainder of the paper. Section 4 then introduces conjectures that we came to elaborate using Alloy model checking to explore subsets of the general problem space. Section 5 generalizes them into formal characterizations with associated proofs.

2 Related Work and Contributions

Our concern is the thorough characterization of the relationships between object-level and source-level coverage criteria, and specifically between machine branch coverage and the DO-178B MCDC criterion. The purpose is a clear determination of the situations when efficient execution trace collection techniques, operating in bounded data space, may be used to assess MCDC.

We know of only a few formalization efforts related to MCDC coverage: [15], [17] and [16] offer a formalization of the DO-178B MCDC criterion in Z, explore aspects of its fault detection capabilities and suggest a

stronger criterion to address identified shortcomings. [9] presents a formal framework to model and reason about Ada programs and a wide variety of coverage criteria, all at the source level. Finally, [7] introduces formal definitions of MCDC variants, to compare their main fault detection characteristics and offer important proofs regarding the minimum size of test sets required to fulfill the criteria.

None of these address the aforementioned area of interest and the main contribution of this paper is to formally establish how OBC techniques can be used to claim MCDC coverage. In particular, we provide a formal characterization of the conditions in which trace collection mechanisms, that are typically used to establish OBC, can be leveraged on to assess MCDC when Boolean operators are restricted to short-circuit forms. Our characterization is expressed in multiple forms: firstly over Binary Decision Diagram (BDD) properties, suitable for automated processing by tools, then in source properties understandable by human users. For both we provide a formal proof of validity, based on an intermediate notion of BDD edge coverage.

To our knowledge, this is the first published characterization of this kind, providing solid grounds for efficient trace-based assessment technologies. This allowed the productization of the COUVERTURE project into the GNATCOVERAGE toolset, a professional coverage analysis framework that scales up to industrial needs (by avoiding the need for gigantic trace data) and can be trusted to be correct.

3 MCDC and OBC/edge coverage

3.1 Modified Condition/Decision Coverage

The definition of MCDC in DO-178B [13] distinguishes between "decisions" and "conditions". A decision is composed of one or more conditions connected by Boolean operators. For example, `(C1 and then C2) or else C3` in Ada is one decision with three conditions `C1`, `C2`, and `C3`. The definition then reads:

> *Modified Condition/Decision Coverage*: 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, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that 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.

Entry and exit points are beyond the scope of this paper. For our purposes, the key point is the concept of a condition that "independently affects" a decision's outcome. To help explain it, we will use a *decision evaluation vector* — a vector of Boolean values where each element corresponds to the value of one condition in the decision. For instance, (T,F,T) for our example decision denotes an evaluation vector where `C1`, `C2` and `C3` evaluate to True, False and True respectively.

A condition C is then said to have an independent influence on a decision D if a pair of evaluation vectors exist which evaluate D to True and False and are identical but for the value of C. A given vector may participate in more than one pair, showing independent influence of different conditions. The latter point is crucial in keeping the required testing complexity linear with the number of conditions, and MCDC is known to require $n + 1$ tests for a decision with n independent conditions [7]. Thus the MCDC criterion calls for much more careful testing than basic decision coverage (2 tests per decision only), while remaining tractable for industrial applications in practice.

Returning to `(C1 and then C2) or else C3`, table 1(a) lists four evaluation vectors that indeed achieve MCDC for the three conditions. The independent influence of `C1` is demonstrated by vectors 1+4, where only `C1` changes and the decision evaluates to True and False. Similarly, the independent influence of `C2` and `C3` is demonstrated by pairs 1+3 and 2+3.

vector	C1	C2	C3	D
1	T	T	F	T
2	T	F	T	T
3	T	F	F	F
4	F	T	F	F

(a) Unique Cause

vector	C1	C2	C3	D
1	T	T	x	T
2	T	F	T	T
3	T	F	F	F
4	F	x	F	F

(b) Short-Circuit

Figure 1: MCDC vectors for (C1 and then C2) or else C3

The strict DO-178B definition we have just introduced has come to be known as *Unique Cause* MCDC. This definition has several limitations:

- Unique Cause MCDC can never be achieved for decisions with coupled conditions — when condition values necessarily change together;
- Even conditions that cannot affect the outcome must remain unchanged in independence pairs, which is unnecessarily restrictive.

Proposals were made in which conditions other than the one of interest may differ in independence pairs. *Unique Cause+Masking* and *Masking* MCDC [7] are two such variants accepted as sound alternatives [5], in which condition variations are allowed when they

cannot influence the decision because *masked* by another condition. The second criterion allows a general use of the masking facility, while the former restricts that option to coupled conditions only.

In COUVERTURE, we restrict Boolean operators to short-circuit forms and resort to what [8] advocates, which we designate as *Unique Cause+Short-Circuit* MCDC. We consider the right-hand side of a short-circuit operator to be masked when the left-hand side alone determines the outcome, and ignore the thus-masked conditions for the determination of independence pairs. This is consistent with the code generation strategy for short-circuit expressions, since the operator right-hand side is not even evaluated in this case. We denote these cases with "x" in vector tables, as for C3 in evaluation vector 1 of table 1(b).

A very useful property of this variant is to allow assessments using execution flow traces, where no information is available about unevaluated conditions. In addition, the set of conditions masked for a given evaluation always is a strict subset of those masked according to the *Masking* criterion, so every set of vectors satisfying MCDC for our variant also satisfies it for the latter. We qualify *Unique Cause+Short-Circuit* as *stronger* than *Masking* to denote this property.

3.2 Object Branch Coverage

Applicants for DO-178B certification have sometimes proposed the use of object code coverage instead of source code coverage as a metric to satisfy the objectives of DO-178B. The proposed approach involves measuring either instruction coverage or branch coverage. Object instruction coverage (OIC) consists in assessing whether all object instructions are executed at least once; object branch coverage (OBC) in addition requires that all conditional branch instructions be exercised in both directions (branch and fall-through). One interesting aspect of these metrics is that they can both be assessed in bounded data space, scaling very well to industrial applications.

The general issue of using object coverage in order to achieve the various source coverage criteria defined in DO-178B [13] is considered in both FAQ 42 of DO-248B [14] (*Can structural coverage be demonstrated by analyzing the object code instead of the source code?*) and CAST paper 17 (*Structural Coverage of Object Code*) issued by the Federal Aviation Administration [6]. Both documents assert that object code coverage can substitute for source code coverage *as long as analysis can be provided which demonstrates that the coverage analysis conducted at the object code will be equivalent to the same coverage analysis at the source code level*. This paper provides elements clarifying the conditions under which such an equivalence can be met in the challenging context of MCDC.

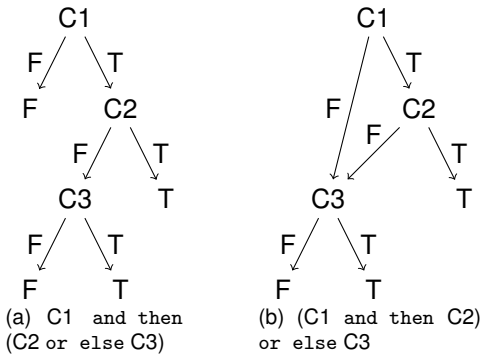


Figure 2: BDD examples

3.3 MCDC assessment strategy

Assessing MCDC from execution traces requires the value of each condition in a source level decision to be represented by a control flow change at execution: there must be a conditional branch testing each condition and altering control flow depending on its value. In this case, evaluating a decision is equivalent to a formal traversal of its reduced ordered BDD (example in figure 2), evaluating conditions from left to right as required. Condition values can be inferred from the direction taken at each conditional branch instruction in the object code.

This assumes that the machine code features the appropriate conditional branch instructions for each condition, i.e. that the *object control flow graph* reflects the source structure of decisions. Even with short-circuit operators only, this has to be carefully controlled. We have introduced a specific compiler mode in the GNAT/GCC suite for this purpose, where optimization passes are tailored and where the compiler generates artifacts that help tracing edges of the control flow graph back to the corresponding edges of each decision's BDD. These include DWARF debugging information (which map object code instructions back to source locations), and Source Coverage Obligations (SCOs), identifying source locations that correspond to constructs that are subject to coverage analysis (statements, decisions, conditions).

As we have described, local branch coverage data (indication for each relevant conditional branch instruction of whether or not it was exercised in both directions) was shown to be insufficient to determine whether MCDC is reached in some situations. In such cases, historical traces are needed to reconstruct the complete set of evaluation vectors, where each vector denotes a path followed in the decision's BDD. A chronological record of the direction taken by each execution of conditional branch instructions needs to be kept, generating a data set of an unwieldy size, as opposed to the bounded size of the data set required to establish OBC.

The following sections provide a precise and formally verified characterization of those situations (decisions) for which local branch coverage data is sufficient to assess MCDC, allowing the collection of complete historical traces to be performed only when necessary.

3.4 Object Branch and BDD Edge Coverage

In order to evaluate MCDC using control flow changes, we will discuss this coverage criterion in terms of BDD coverage: MCDC is assessed by examining the set of distinct paths through the BDD that have been taken. The data we have at our disposal is the trace of conditional branch instructions, which conveys information about exercising edges of the executable control flow graph (CFG). We need to bridge the gap between these edges and those of the source BDD.

The correspondance is straightforward when there is just one conditional branch instruction for a given condition: the two CFG edges for this instruction (branch and fall-through) correspond to the True and False outgoing edges out of the BDD node for the condition. However, the evaluation of some complex conditions may involve more than one conditional branch instruction, as for example in the case of the Ada `mod` operator, which involves an overflow test for modulus -1, and a test on operand sign (figure 3).

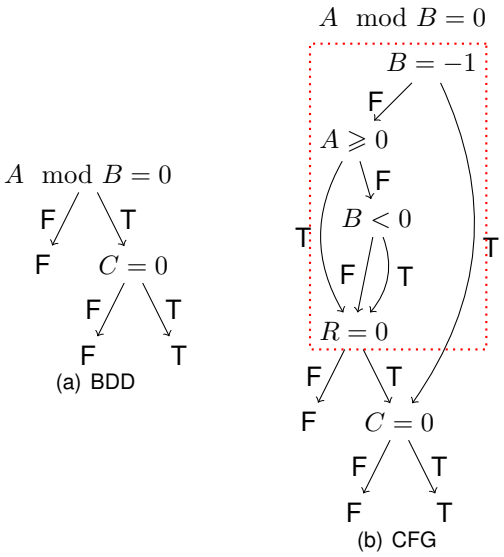


Figure 3: $A \mod B = 0$ `and then` $C = 0$

Edges of the CFG then fall into two categories: either they remain within the region of code for a given condition, or they correspond to a BDD edge, transfering control to code evaluating one of the successor conditions or determining the outcome of the decision. When processing control flow traces to assess MCDC, we concern ourselves only with the latter. We ignore intra-condition branches, as they do not provide any information as to the value of a condition or the outcome of the decision so have no impact on source coverage.

4 Elaborating conjectures with Alloy

In the initial stages of the COUVERTURE project, we tried to establish equivalence conditions between OBC and MCDC, and to determine the relationships between branch coverage data and MCDC evaluation vectors. In order to gain a better understanding of these issues, we formalized the problem in Alloy [11], which proved to be a powerful tool to test and debug conjectures.

Alloy is a formal specification language, just as Z, B, VDM or OCL. It can be used to describe a problem as a set of first-order logic constraints over sets and relations. A binary decision diagram (BDD), for example, can be represented as a set of two relations that record its two types of edges, with the constraint that the corresponding graph should be acyclic and have only one root (sketch on listing 1).

```
pred is_bdd [if_true, if_false :
             BDD_Node -> (BDD_Node + BDD_Term)]

  let graph = if_true + if_false {
    -- Exactly one root. dom[graph] is the
    -- set of nodes in BDD, graph.node is the
    -- set of fathers for node.
    one node : dom[graph] | no graph.node

    -- Acyclicity. ^ is transitive closure;
    -- identity not in the closure means no
    -- node has a path to itself.
    no iden & ^graph
  ...
```

Listing 1: BDDs in Alloy

The full model can be downloaded from the COUVERTURE forge on forge.open-do.org.

Alloy comes with a set of tools to automatically check for consistency and to find counterexamples on small models of the problem: in our case, this typically means that properties could be verified on decisions with fewer than 5 conditions. Although this does not prove these properties in all cases, it allowed us to rule out a significant set of false assumptions. In the context of COUVERTURE, the following abstractions have been formally specified:

- Decisions (by their syntax tree);
- (Reduced Ordered) Binary Decision Diagrams;
- BDD edge coverage;
- Masking and Unique Cause variants of MCDC.

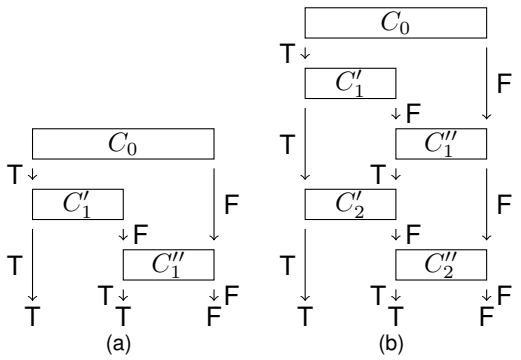
In this model, all conditions in decisions are supposed to be independent; no coupling is considered. The idea was to use this model to help organize the problem space: which coverage criteria are equivalent, in which circumstances, and the different ways to express these circumstances (in terms of constraint on the BDD or on the syntax tree). The following section shows how such a model clarifies the situation by exposing a set of false assumptions and offering high confidence that other assumptions are valid.

4.1 Does BDD edge coverage imply MCDC?

The canonical `(C1 and then C2) or else C3` example is indeed a case where BDD edge coverage does not imply MCDC, so the answer is negative. By modeling BDD edge coverage and the different MCDC criteria, it was possible to explore broadly the space of such counterexamples. In particular, when asked to find decisions that could be covered for BDD edge coverage by 3 evaluations, Alloy was able to find instances with decisions from 3 to 11 conditions (more than 11 conditions made the model too big to be handled by Alloy). The minimum number of evaluations to reach any known MCDC variant is a strictly increasing function of the number N of conditions: [7] proved that Unique Cause needs at least $N + 1$ tests, and Masking MCDC $\lceil(2\sqrt{N})\rceil$ tests. These two functions take values that are strictly greater than 3 for decisions with more than 3 conditions; so these are indeed other cases where the original assumption is invalid.

These counterexamples suggested that a generalization to any number of conditions could be produced. And indeed there exists classes of decisions with an arbitrary high number of conditions that can be edge covered by just 3 evaluations. Consider for instance the following set $\{D_n\}_{n\in\mathbb{N}}$ of decisions:

- let D_0 be a simple condition decision, with its condition denoted C_0; then define:
- $\forall\ n > 0, D_n = (D_{n-1}$ `and then` $C'_n)$ `or else` C''_n, with C'_n, C''_n independent from each other and from any condition in D_{n-1}.



C_0	C'_1	C''_1	C'_2	C''_2	D_2
T	T	x	T	x	T
T	F	T	F	T	T
F	x	F	x	F	F

(c)

Figure 4: BDDs that can be edge covered by 3 evaluations

Figure 4(a) shows the BDD for D_1, and Figure 4(b) for D_2. It can be seen that all the edges can be covered by 3 evaluation paths which only demonstrate the independent effect of C_0; these 3 evaluations are given in figure 4(c).

By induction we can build a decision D_n with an arbitrary number of conditions that can be BDD edge covered by just 3 evaluation paths. As MCDC can only be achieved with a minimal number of $\lceil(2\sqrt{N})\rceil$ evaluations, this is a striking case where BDD edge coverage is far from being equivalent to MCDC.

4.2 When does BDD edge coverage imply MCDC?

After having explored the space of counterexamples, it was still possible to identify the conditions where the implication holds. From the preliminary study, a pair of conjectures emerged for a criterion that would characterize cases where edge coverage and MCDC are equivalent:

Conjecture 1 *If the* BDD *of a decision D is a tree (with only one path from the root to any condition node), then* BDD *edge coverage implies* MCDC*.*

Conjecture 2 *If the* BDD *of a decision D is not a tree, then there exists a set of evaluations that covers the edges of its* BDD *but does not reach* MCDC*.*

When checked against the Alloy model, no counterexample was found for decisions with up to 6 conditions, for all forms of MCDC. Each conjecture was checked separately:

- Assuming BDD is a tree, prove that BDD edge coverage implies the strongest MCDC variant;
- Assuming BDD is not a tree, build cases where BDD edge coverage is reached whereas the weakest MCDC variant is not.

When only short-circuit operators are used in decisions, the strongest MCDC variant in our scope is Unique Cause+Short-Circuit MCDC. The weakest known variant is Masking MCDC, but its formal specification is complicated and degrades the performance of the automatic verification enough to restrict drastically the field of possible explorations. A weaker and simpler criterion was introduced to address this; Weak MCDC, defined as follows:

Definition 1 *Given a decision D, a pair of evaluation vectors satisfies Weak* MCDC *for a condition C if, and only if, the condition and the decision have both been evaluated to True and False.*

A test set satisfies Weak MCDC *for a decision D if, and only if, for each condition in this decision, there exists a pair of tests in the test set that satisfies Weak* MCDC*.*

Weak MCDC does not prove independent effect of a condition on a decision; this makes it much weaker than Masking and Unique Cause. Its formal specification is much simpler though, and allowed validating the two conjectures with up to 6 conditions. It also helped manual proofs, as the next section will show.

Incremental refinements also allowed us to rephrase these two conjectures in a more direct and human-readable manner:

Conjecture 3 *Given a decision D,* BDD *edge coverage implies* MCDC *if, and only if, when considering the negation normal form D' of D, for every sub-decision E of D', all binary operators in the left-hand-side operand of E, if any, are of the same kind as E's operator: that is to say, all of them are* `or else` *if E's operator is* `or else`*, and all of them are* `and then` *if E's operator is* `and then`*.*

Automatic verification finds no counterexamples to this conjecture for decisions with up to 5 conditions.

5 From conjectures to proofs

The previous section discussed how we tested our conjectures on equivalence cases between OBC and MCDC using Alloy models of decisions with up to five conditions. Alloy was an invaluable tool for extracting counterexamples and finding interesting corner cases. Once we had refined our hypotheses, we developed general proofs of these conjectures, which apply to arbitrarily complex decisions.

In the following sections, unless explicitely indicated, we always assume that decisions contain no coupled conditions. The conjectures 1 and 2 can now be rephrased as theorems that we will prove in the general case:

Theorem 1 *If the* BDD *of a decision D is a tree (with only one path from the root to any condition node), then* BDD *edge coverage implies* MCDC*.*

Theorem 2 *If the* BDD *of a decision D is not a tree, then there exists a set of evaluations that covers the edges of its* BDD *but does not reach* MCDC*.*

In other words: a tree BDD is necessary and sufficient for BDD edge coverage to imply MCDC. The proof of these two theorems will rely extensively on structural induction on the reduced ordered BDD of a decision, whose construction is detailed below.

5.1 Construction of the reduced ordered BDD

The BDD for a decision is constructed using the following recursive procedure:

Build_BDD.Condition
As illustrated on figure 5(a), the BDD for a decision consisting in a single condition C has:

- the node"test C" as its entry point;
- the label True assigned to the edge corresponding to "C is True";
- the label False assigned to the edge corresponding to "C is False".

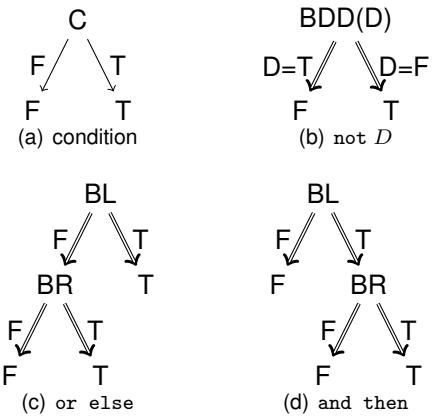


Figure 5: Build_BDD

Build_BDD.NOT
The BDD for $\texttt{not}(D)$ is the BDD for D with the labels of the exit edges swapped. It is illustrated in figure 5(b), where double arrows represent edge sets.

Build_BDD.Short_Circuit_Operator
For any short-circuit operator $\star$, the BDD for $(DL) \star (DR)$ is constructed as follows (figure 5(c) and 5(d)):

If $\star$ is `and then`, let SC be False; if $\star$ is `or else`, let SC be True.

Let BL be the BDD for DL, and BR the BDD for DR.

Then B, the BDD for D, is obtained by combining BL and BR as follows:

- the entry point is that of BL
- the exit edge labeled SC of BL is an exit edge labeled SC of B
- the other exit edge of BL connects to the entry point of BR
- the exit edges of BR are exit edges of B with the same labels

Usual properties of reduced ordered BDDs [4] follow from the construction process. In particular, we have the independent outcome reachability property, which states that, for independent conditions, there is a path from each node to an exit edge labeled True and to an exit edge labeled False, such that the two paths start with distinct edges from the node. Given a decision D, we will now call BDD(D) its BDD as built by this recursive procedure. Each condition of D corresponds to one node in BDD(D), and evaluating a decision is equivalent to traversing its BDD.

Most properties presented in the section are proved by induction on the structure of the decision. For each case of the Build_BDD procedure, we assume that the property holds for the parameters and prove that the build step preserves the property.

5.2 First case: BDD is a tree

We now consider the case of an expression whose BDD is a tree, i.e. for each BDD node there is exactly one path from the entry point to that node. We prove that BDD edge coverage implies Unique Cause+Short-Circuit MCDC, which extends to weaker variants by construction and proves theorem 1.

Let us consider an arbitrary condition C. If we have BDD edge coverage, then all possible paths starting at C have been taken (by recursion on path length, taking advantage of tree structural properties). From the independent outcome reachability property, we have two paths starting at C, beginning each with one edge from C, and ending on the two decision outcomes. Since the BDD is a tree, there is a single path from the entry point to C and any condition appearing on the path from C to one outcome is not evaluated on the path to the other outcome. Thus, we have two paths from the entry point to both outcomes, that differ in C, in no other condition before C, and in no other non-masked condition after C, which proves the independent influence of C over the decision.

This holds for each condition in the decision, so Unique Cause MCDC + Short-Circuit is proved.

We can now comment on the case of coupling. When some conditions of a decision are coupled, the set of possible evaluations is a strict subset of the one that we would have in absence of coupling. If this removes the possibility of covering the BDD edges, then it is still true that BDD edge coverage implies MCDC: the left operand of the implication being false, the implication is true. On the other hand, if coupling does not remove the possibility of covering the BDD edges, then such a coverage also implies MCDC, by the same proof as in the case of no coupling. Therefore theorem 1 also holds if there are coupled conditions in a decision.

For a similar reason, since OBC implies BDD edge coverage, this theorem extends to OBC; i.e. if the BDD of a decision D is a tree, then OBC implies MCDC.

5.3 Second case: BDD is not a tree

The general idea of the proof of theorem 2 is to show that, for any BDD that is not a tree, we can build a set of evaluations that covers the BDD edges in such a way that there is at least one condition for which MCDC is not met. As the notion of "independent influence" differs between MCDC variants, this proof focuses on the weakest one: Weak MCDC. If a set of evaluations proves BDD edge coverage but not Weak MCDC, it necessarily fails to prove any stronger variant as well.

On our `(C1 and then C2) or else C3` example, we observe that the set of evaluations 1+2+4 from table 1(b) achieves BDD edge coverage but not Weak MCDC as the decision outcome remains True whenever C2 is evaluated. For the more general characterization we introduce *multipath nodes* as the BDD nodes reachable by more than one path from the entry point.

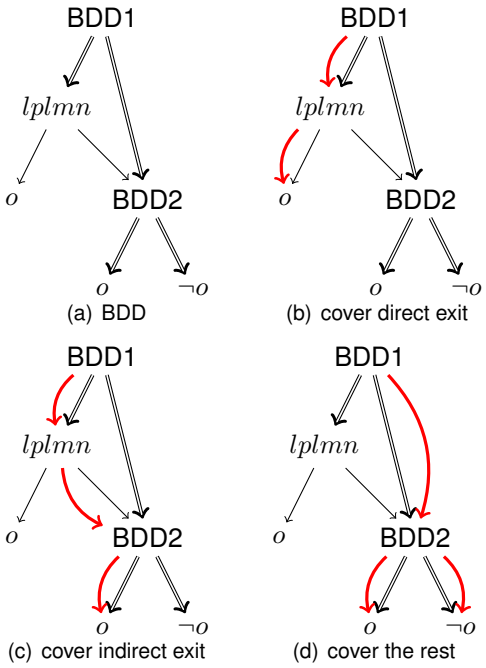


Figure 6: BDD edge coverage but no MCDC

With reduced ordered BDDs each node can be ordered. With short-circuit operators, this order is the order of conditions in the decision expression, and we have well defined notions of *last multipath node* and its *last parent*. In our example, the last multipath node would be C3, and its last parent C2. For these two entities, the following property holds:

Lemma 1 *If a* BDD *is not a tree, then the last parent of the last multipath node has an exit edge that is directly connected to an outcome. We call this outcome the node's direct outcome and the edge connected to the direct outcome will be called the direct exit edge.*

This lemma can easily be proven by structural induction on Build_BDD. Figure 6(a) illustrates its consequences in term of BDD topology: $lplmn$ is the last parent of the last multipath node, one of its edges is directly connected to an outcome o; its second edge is connected to the last multipath node, which is the root of the sub-bdd BDD2; the sub-bdd BDD1 has its exit edges connected to $lplmn$ and BDD2's root, and can have some other exit edges connected to an outcome (not represented on this figure).

From this topology, it is possible to build three particular sets of evaluations:

- a first set such that all incoming edges of $lplmn$ are covered, whose evaluations always evaluate $lplmn$ and always exit on the direct exit edge; this is figure 6(b);
- a second set whose evaluations always evaluate $lplmn$, do not take the direct exit edge, traverse BDD2 and end up exiting on o; this is figure 6(c);

- a third set that never executes $lplmn$ and covers the rest of the BDD (including the exit edges for o that have not been covered by the second set); this is figure 6(d).

The union of these three sets covers the BDD, but Weak MCDC is not satisfied: whenever $lplmn$ is evaluated, the decision has the same value (o). As a consequence, this constitutes a BDD coverage for which no MCDC variants are reached.

The complete formal proof builds these three sets by induction on the structure of the decision. This demonstration is detailed in the COUVERTURE documentation [2].

Note, this time, that theorem 2 does not hold if coupling is permitted: coupling may specifically prevent the BDD edge coverages that fail to achieve MCDC. For the same reason, this theorem cannot be rephrased in terms of OBC: there may be cases where coupling between intra-condition branches and inter-condition branches precludes having BDD edge coverage but not MCDC.

5.4 Formulation in terms of operator combination

In the previous two subsections, we showed that BDD edge coverage is equivalent to MCDC for a decision if, and only if, the decision BDD is a tree. Automatic verification allowed us to reformulate this property in terms of operator combination in the negation normal form of the expression, obtained by rewriting it using De Morgan's laws so that negations apply only to atomic conditions (and not to more complex subexpressions). We can now requalify conjecture 3 as a theorem:

Theorem 3 *Given a decision D,* BDD *edge coverage implies* MCDC *if, and only if, in the negation normal form D' of D, for every sub-decision E of D', all binary operators in the left-hand-side operand of E, if any, are of the same kind as E's operator.*

Let us first consider the case of a decision D containing no negation operator, and its associated BDD.

Let NF(D) be the number of paths from the root of decision D to False (F) and NT(D) the number of paths from the root of decision D to True (T). For a condition, $NF(D) = NT(D) = 1$. For an `and then` form:

$$\begin{cases} NF(D) &= NF(DL) + NT(DL) \times NF(DR) \\ NT(D) &= NT(DL) \times NT(DR) \end{cases}$$

Similarly, for an `or else` form:

$$\begin{cases} NF(D) &= NF(DL) \times NF(DR) \\ NT(D) &= NT(DL) + NF(DL) \times NT(DR) \end{cases}$$

We therefore have the following bounds:

Lemma 2 *For every decision D,* $NF(D) \geqslant 1$ *and* $NT(D) \geqslant 1$.
If D is of the form `or else` *then* $NT(D) \geqslant 2$.
If D is of the form `and then` *then* $NF(D) \geqslant 2$.

and by structural induction on the form of decisions, we can show:

Lemma 3 *NF and NT are monotonic functions, i.e. if D' is a sub-decision of D, then* $NF(D') \leqslant NF(D)$ *and* $NT(D') \leqslant NT(D)$.

With these lemmas, we can now prove theorem 3.

Proof of direct implication, by contraposition:

Consider a decision D of the form `and then` containing a sub-decision D' of the form `or else` on the left-hand side. If DL and DR are the two sub-decisions such that $D = DL$ `and then` DR, then D' is also a sub-decision of DL.

Let's now count the paths from the root of the BDD to the root of DR. Since D is an `and then` form, this is exactly $NT(DL)$. By lemma 3, we know that $NT(DL) \geqslant NT(D')$, and by Lemma 2, we know that $NT(D') \geqslant 2$.

We have thus proved that DR's root node in BDD(D) is reachable by more than one path, so there is a multipath node in the BDD. We can apply similar reasoning for a decision D that is an `or else` form whose left operand contains a subdecision D' that is an `and then` form.

By contraposition, we have therefore shown that if the BDD is a tree, then no operator of one kind has an operator of the other kind in its left operand.

Proof of converse:

We now assume that for every sub-decision in decision D, the left operand of an `and then` sub-decision contains only `and then` sub-decisions and the left operand of an `or else` sub-decision contains only `or else` sub-decisions.

Lemma 4 *If E contains only* `or else` *sub-decisions then* $NF(E) = 1$. *If E contains only* `and then` *sub-decisions then* $NT(E) = 1$.

Proof of Lemma 4 is on structural induction on the form of decisions, based on the 3 cases distinguished above (atomic condition, `and then` form, and `or else`).

By structural induction on the depth of the BDD, we can show that there cannot be any multipath node in the BDD associated with D, which proves that the desired implication holds.

Proof of the general case:

We now extend the above derivation to the case of a decision D containing `not` sub-decisions. Then, consider D' the negation normal form of D. Since D and D' are represented by the same BDD, Theorem 3 follows.

6 Evaluation on industrial code

A first evaluation of the impact of theorem 1 has been given in [3]; two industrial applications were analyzed and it was shown that less than 1% of decisions have a multipath node. In practise, this means that a coverage tool that uses this theorical result does not need to keep full historical traces except for the few decisions requiring it. This optimization has been implemented in GNATCOVERAGE and it drastically reduced the overhead that historical traces introduce.

GNATCOVERAGE has been used to measure the coverage of its own qualification testsuite, and the size of the traces generated by that operation can help us evaluate the impact of the optimization. Of the 1026 decisions in GNATCOVERAGE, only 4 of them contain multipath nodes: this ratio is similar to the one observed in other industrial applications. Coverage traces have been collected for 3 configurations: firstly, for OBC, since these traces do not require any historical information and can serve as a baseline for measuring the overhead of historical traces; secondly, MCDC 1 with historical traces on branches of all decisions (inter-condition branches excluded); thirdly, MCDC 2 taking into account theorem 1 and having historical traces only for BDDs that are not a tree. Every run of the qualification testsuite gathered 10279 traces whose total size are as follows:

configuration	OBC	MCDC 1	MCDC 2
#branches to trace	0	1788	22
size of traces	1.33G	5.06G	1.37G

As this example shows, the optimization allowed to remove 99% of historical traces; the overhead compared to OBC traces can be considered as marginal.

7 Conclusion

Structural coverage analysis plays an important role in the verification of safety-critical software. Historically, such analysis, considered hard, was delegated to the end of the development cycle, and inevitably, software changes required to take into account the results of this analysis were cumbersome and expensive. With the increasing popularity of Agile methods and in particular, one of its key concepts – Continuous Integration – it is important to be able to perform such analysis on a regular basis; ideally every day or even after each software change. Being able to perform such an analysis directly on the final non-instrumented code from moderately-sized and bounded execution traces help considerably in such a context.

Object-level coverage metrics can easily be implemented with such characteristics whereas it is not as usual with source-level ones. The results presented and demonstrated in this paper can provide the foundation for a bridge between those two kinds of coverage techniques. In particular, Theorems 1 and 2 provide the characteristics of complex decisions in which the MCDC criterion can be deduced from bounded execution traces with no history. Theorem 3 provides the same characteristics in source-level terms. Those results have been exploited in the context of the COUVERTURE project in the design and implemention of an efficient and accurate coverage tool producing both source-level and object-level coverage metrics.

References

[1] QEMU, a generic and open source machine emulator and virtualizer. `http://www.qemu.org/`.

[2] AdaCore. Couverture - Technical report on OBC/MCDC properties, 2010.

[3] Matteo Bordin, Cyrille Comar, Tristan Gingold, Jerome Guitton, Olivier Hainque, and Thomas Quinot. Object and Source Coverage for Critical Applications with the COUVERTURE Open Analysis Framework. In *ERTS (Embedded Real Time Sofware and Systems Conference)*, May 2010.

[4] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[5] CAST, Certification Authorities Software Team. Rationale for accepting Masking MCDC in certification projects. Position Paper 6, August 2001.

[6] CAST, Certification Authorities Software Team. Structural Coverage of Object Code. Position Paper 17, June 2003.

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

[8] John J. Chilenski and Steven P. Miller. Applicability of modified condition/decision coverage to software testing. volume 9, issue 5 of *Software Engineering Journal*, pages 193–200, September 2004.

[9] John J. Chilenski and Philip H. Newcomb. Formal Specification Tools for Test Coverage Analysis. Software Engineering Journal, pages 59–68, 1994.

[10] FAA, Federal Aviation Administration. Object Oriented Technology Verification Phase 3 Report - Structural Coverage at the Source Code and Object Code Levels. Technical Report DOT/FAA/AR-07/20, June 2007.

[11] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[12] George Romanski. MCDC coverage analysis using short-circuit conditions. `http://www.verocel.com/`.

[13] RTCA. Software considerations in airborne systems and equipment certification. Document RTCA DO-178B, 1992.

[14] RTCA. Final annual report for clarification of do-178b ”software considerations in airborne systems and equipment certification”. Document RTCA DO-248B, 2001.

[15] Sergiy A. Vilkomir and Jonathan P. Bowen. Formalization of software testing criteria using the z notation. In *Computer Software and Applications Conference (COMPSAC)*, pages 351–356. IEEE Computer Society, 2001.

[16] Sergiy A. Vilkomir and Jonathan P. Bowen. From MC/DC to RC/DC: Formalization and Analysis of Control-Flow Testing Criteria. Technical Report SBU-CISM-02-17, South Bank University, CISM, London, UK, 2002.

[17] Sergiy A. Vilkomir and Jonathan P. Bowen. Reinforced Condition/Decision Coverage (RC/DC): A New Criterion for Software Testing. In Didier Bert, Jonathan P. Bowen, Martin Henson, and Ken Robinson, editors, *ZB2002: Formal Specification and Development in Z and B*, volume 2272 of *Lecture Notes in Computer Science*, pages 295–313. Springer Verlag, 2002.