COUVERTURE
Technical Report on OBC/MCDC properties

Abstract

This document gathers results established or formalized by the COUVERTURE project team about relationships between specific coverage criteria. We focus in particular on how *Object Branch Coverage* (OBC) relates to the *Modified Condition/Decision Coverage* (MC/DC) criterion.

We provide two broad categories of results: formal proofs of important properties over a model of the two criteria, and a machine-automated verification of some of these properties for concrete subsets of the model expressed in Alloy. These results constitute the grounds on which our project coverage analysis framework operates to infer source coverage results from object coverage information out of an instrumented execution environment.

1 Common definitions

1.1 Decisions, conditions

We are considering decisions that are short circuit boolean expressions, i.e. expressions consisting in elementary boolean conditions combined together using only the `and then`, `or else` and `not` operators.

For each decision we construct the associated ROBDD (Reduced Ordered Binary Decision Diagram), whose nodes are conditions. (RO)BDDs have an entry point, and two or more exit edges labeled True and False.

In the remainder of this document, unless otherwise indicated, all references to BDDs denote reduced ordered BDDs. For a set S, #S will refer to this set's cardinal. For a decision D, cond(D) will be the set of its conditions.

1.2 Coverage metrics

1.2.1 Definitions

This document deals with properties of tests exercising object code under the assumption that the code generation chain accurately preserves the decision process captured in the ROBDD. We specifically assume when discussing object branch coverage of the object code, that we can instead reason, unless indicated explicitly, on (RO)BDD branch coverage of the corresponding BDD.

1.3 Minimal test sets

Coverage assessment is accomplished by exercising some piece of object code in a variety of test cases, recording data along the way, and then determining whether the successive executions of the object code satisfy a given criterion of exhaustiveness. The minimum number of distinct executions required to achieve a specific criterion is an important aspect in the evaluation of any coverage assessment methodology. Here we establish some properties that give a hard limit on test set size for various coverage metries.

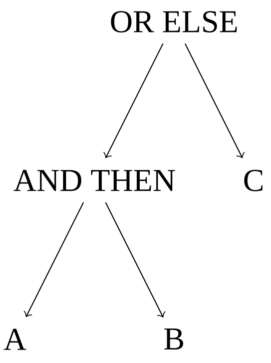
1.3.1 OBC

1.3.2 MC/DC

Formal definition: The formal definition of MC/DC that it used here is the one given in [Chi01], using a graph-coloring algorithm to determine whether two given evaluations show the independent influence of a condition. Amongst the various formal definitions that you may find in the litterature, this one has the advantage to apply also to short-circuit operators, which is mandatory in the context of COUVERTURE.

The following will not re-define what has already detailed in [Chi01]; we will just give an example on how it works and will refer the interested reader to the original document.

This formal definition of MC/DC uses a simple graph-coloring algorithm to determine whether two given evaluations show the independent influence of a condition. It first colors each node of the decision syntax tree with both evaluations; for example, for a decision $(A$ `and then` $B)$ `or else` C, the syntax tree can be seen on figure 1(a).



(a)
$(A$ `and then` $B)$ `or else` C

For each evaluation, each node of the tree (atomic condition or root of sub-decision) is colored with the value that it takes in this evaluation: True (T), False (F), Not_Evaluated (X). For the evaluation $e1 = (A = True, B = True, C = Not\_Evaluated)$, we have the coloring given on figure 1(b); and for $e2 = (A = False, B = X, C = Not\_Evaluated)$, we have the coloring given on figure 1(c).

Both graphs are then combined into an influence tree by xor'ing each node. An extension of xor to three-value boolean algebra is used: anything xor Not_Evaluated gives False. In our case, the result is showed on figure 1(d).

The influence set of these two evaluations for the given decision is the set of conditions that have a path of "True nodes" to the root. So here, it is the singleton {A}.

$$\begin{tikzcd}
 & & \text{OR ELSE=T} \arrow[dl] \arrow[dr] & \\
 & \text{AND THEN=T} \arrow[dl] \arrow[dr] & & \text{C=X} \\
\text{A=T} & & \text{B=T} &
\end{tikzcd}$$

(b) e1

$$\begin{tikzcd}
 & & \text{OR ELSE=F} \arrow[dl] \arrow[dr] & \\
 & \text{AND THEN=F} \arrow[dl] \arrow[dr] & & \text{C=F} \\
\text{A=F} & & \text{B=X} &
\end{tikzcd}$$

(c) e2

The two classical variants of MC/DC are then defined as follow:

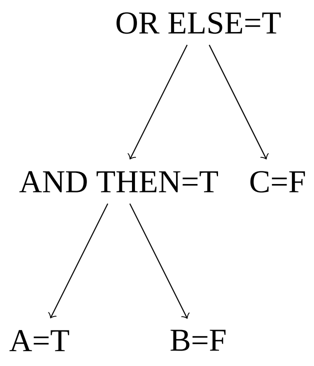
DEFINITION 1 *Given a decision D, a pair of truth vectors satisfies Masking MC/DC for a condition c in cond(D) if and only if its influence set is the singleton c.*

A test set satisfies Masking MC/DC for a decision D if, and only if, for each condition in cond(D), there exists a pair of tests in the test set that satisfies Masking MC/DC.

DEFINITION 2 *Given a decision D, a pair of truth vectors satisfies Unique Cause MC/DC for a condition c in cond(D) if and only if its influence set is the singleton c and if c is the only condition that is colored with True in the influence tree.*

A test set satisfies Unique Cause MC/DC for a decision D if, and only if, for each condition in cond(D), there exists a pair of tests in the test set that satisfies Unique Cause MC/DC.

In our example, our pair of truth vectors satisfies both Unique Cause and Masking MC/DC for condition A.



(d) Influence tree

Properties: For Unique Cause, the following property holds:

THEOREM 1 *Unique MC/DC on a decision with n independent conditions is achieved with a test set of exactly $n+1$ tests, and cannot be achieved in fewer tests.*

The existence of the test set is proved by induction. For one condition, MC/DC is achieved with two tests, one setting it True and the other False.

Now assume that the property holds for all $n \leqslant N$, and consider a decision with $N+1$ conditions. It is of the form $D = D_l \star D_r$ where $\star$ is either `and then` or `or else`. There is also possibly a negation, which is omitted here since it has no impact on MC/DC. For the remainder of this proof we assume the operator is `and then`; the same reasoning applies similarly for the case of `or else`.

D_l and D_r are decisions with respectively n_l and n_r conditions (both at most N), and $n_l + n_r = N+1$. From the induction hypothesis we have two test vector sets $T_l = \{v_l(0)..v_l(n_l)\}$ and $T_r = \{v_r(0)..v_r(n_r)\}$ that satisfy MC/DC for D_l and D_r respectively, and we can arbitrarily choose the indices so that D_l is True for $v_l(0)$ and D_r is True for $v_r(0)$.

We can now create a combined test set for the complete decision as follows.

$$(\forall j \in [0, n_l])\, v(j) = (v_l(j) \cdot v_r(0))$$

$$(\forall j \in [1, n_r])\, v(n_l + j) = (v_l(0) \cdot v_r(j))$$

We have thus created a set $T = \{v(0)..v(n_l + n_r)\}$ of $N+2$ tests. It is immediate that the elements with indices 0 to n_l give for D the same outcome as the corresponding elements of T_l for D_l, and they all have identical values for the conditions coming from

D_r, so they show independent influence of all conditions coming from D_l. Similarly the vector set $\{v(0), v(n_l+1)..v(n_l+n_r)\}$ shows independent influence of those conditions coming from D_r, so the new test set T satisfies MC/DC for D and thus the induction property holds at $N+1$ as well, since T has $n_l + n_r + 1 = N + 2$ elements.

The proof that this is the minimal test set size is given in [Chi01]. As for Masking MC/DC, we have the following property:

THEOREM 2 *Masking MC/DC on a decision with n independent conditions requires a minimum of* $RUTW(2 * SQRT(n))$ *tests, where RUTW stands for round up to whole.*

The proof of this property is given in [Chi01] as well.

1.4 Coverage assessment through analysis of execution traces

2 Characterization of cases of BDDBC — MC/DC equivalence

This section discusses the distinction between expressions for which BDDBC of the associated ROBDD implies MC/DC, and expressions for which no such implication holds.

2.1 Some cases of non equivalence between OBC and MC/DC

First let us have a look at how MC/DC and object coverage relate to each other on some simple cases. Cases of non-equivalence for decisions with up to 5 conditions have been studied in [FAA07]: non-equivalence cases have been shown to occur in decisions with three or more conditions, and an illustration is provided with (A `and then` B) `or else` C, where A, B and C are three independent conditions. A representation of this decision's BDD is depicted on figure 1(e):

From this representation, we can see that a set of three evaluations can achieve branch coverage of the whole BDD, corresponding to the three vertical paths in figure 1(e). These evaluations are:

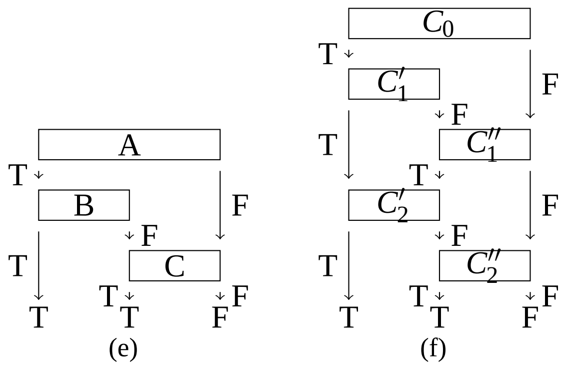


Figure 1: Example decision BDDs

A	B	C	(A `and then` B) `or else` C
T	T	x	T
T	F	T	T
F	x	F	F

where “x” means not evaluated and thus can be indifferently True or False. Now, indeed, even though all the BDD edges are covered, MC/DC is not met. In particular, the independent effect of conditions B on the decision is not shown in the case of Masking MC/DC, and the independent effect of both B and C are not shown in the case of Unique Cause. Since $n+1$ tests are needed to cover a decision with n condition with respect to Unique Cause MC/DC, 3 evaluations cannot cover a three-condition decision. Masking MC/DC requires a minimum of 4 evaluations as well in this particular case.

It turns out that this particular case can be generalized in a quite spectacular counterexample: there exists classes of decisions with an arbitrary high number of conditions that can be branch covered by just three evaluations; the previous example was one element of this class with 3 conditions.

Consider the following set $\{D_n\}_{n\in\mathbb{N}}$ of decisions:

- let D_0 be a simple condition decision; by convention, we will call C_0 its condition;
- let us define D_n, for any $n > 0$, as follows:
  $D_n = (D_{n-1}$ `and then` $C'_n)$ `or else` C''_n
  C'_n and C''_n being independent from each other and from any condition in D_{n-1}.

In other words:

- $D_0 = C_0$
- $D_1 = (C_0$ `and then` $C_1')$ `or else` C_1''
- $D_2 = (((C_0$ `and then` $C_1')$ `or else` $C_1'')$
  `and then` $C_2')$ `or else` C_2''
- $D_3 = (((((C_0$ `and then` $C_1')$ `or else` $C_1'')$
  `and then` $C_2')$ `or else` $C_2'')$
  `and then` $C_3')$ `or else` C_3''
- ...

Figure 1(f) shows the BDD for D_2, where it is visible that all the edges can be covered by three evaluation paths which only demonstrate the independent effect of C_0:

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

We can thus build a decision D_n with an arbitrary number of conditions, that can be BDD branch covered by just three evaluation paths. As Unique Cause MC/DC can only be achieved with a minimal number of $n+1$ evaluations, and Masking MC/DC with a minimal number of $RUTW(2*SQRT(N))$ tests, this is a striking case where BDD branch coverage (and consequently OBC) is far from being equivalent to MC/DC.

We can now adopt a more general perspective and characterize more precisely the difference between these two criteria.

2.2 Construction of the ROBDD

The BDD we associate with a decision is constructed using the following recursive procedure:

Build_BDD.Condition The BDD for a decision consisting in a single condition C has the node "test C" as its entry point, the label True is assigned to the branch corresponding to "C is True", and the label False is assigned to the branch corresponding to "C is False".

Build_BDD.NOT The BDD for `not`(D) is the BDD for D where the labels of the exit edges have been swapped.

Build_BDD.Short_Circuit_Operator This rule defines how the BDD for $(DL) \star (DR)$ is constructed for any short-circuit operator $\star$.

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

The following invariants of ROBDDs follow from the construction process:

- There is exactly one BDD node for each condition.
- All condition nodes are reachable (i.e. there is a path from the entry point to any node in the BDD).
- There are no cycles in the BDD.
- Both outcomes are reachable (i.e. there is a path from the entry point to an exit edge labeled True and to an exit edge labeled False).

Given a decision D, we will now call BDD(D) its (RO)BDD as built by this recursive procedure.

2.3 Node ordering

For a decision D and a condition C in D, let us call index(D,C) the positive number built by the following recursive procedure:

Build_BDD_Order.Condition The sub-decision is a single condition decision, it is of the form:
$D ::= C$
then index(D,C) = 1

Build_BDD_Order.NOT The sub-decision is of the form:
$D ::= \texttt{not}\ D1$
then for each condition C in D, index(D,C) = index(D1,C)

Build_BDD_Order.Short_Circuit_Operator The sub-decision is of the form:
$D ::= DL \star DR$
then, for each condition C in D:

- if C is in DL, $index(D,C) = index(DL,C)$
- if C is in DR, $index(D,C) = index(DR,C) + \#cond(DL)$

This builds a total order over the conditions of a decision; it is a general result for reduced order BDD that the reflexive transitive closure defines a total order over its nodes; this order is the same as the one we just defined. It can also be demonstrated easily, by structural induction, that this order is the order of conditions in the decision's expression.

For a BDD B, we will call root(BDD(D)) the unique node with index 1. This node is indeed the root node of the BDD.

2.4 Evaluation of a decision

Given the transformation of a decision into its ROBDD, evaluating the decision consists in computing its value using the following BDD traversal procedure:

Eval.Condition The value of a decision that consists in a lone condition is the value of the condition.

Eval.Not To evaluate $\texttt{not}(D)$, evaluate D and take the opposite value

Eval.Short_Circuit_Operator To evaluate $(D1) \star (D2)$, evaluate D1. If $D1 = SC$ then the value is SC, else evaluate D2, and the value is that of D2.

The following property holds:

Evals_Are_Paths Evaluating a decision is equivalent to traversing the BDD, evaluating each condition as BDD nodes are traversed, and using the label of the exit edge as the value of the decision.

This property, and most properties that we'll discuss here, is proved by induction on the structure of the decision. For each case of the BDD_Build procedure (Build_BDD.Condition, Build_BDD.Not, Build_BDD.Short_Circuit_Operator), we will assume that the the property holds for the parameters and prove that the build step preserves the property.

Note that the practical implementation of coverage analysis systems based on control flow traces relies on the assumption that the code generator used to produce executable code from expressions actually implements this evaluation strategy.

2.5 Equivalence case

We now consider the case of an expression whose BDD has no diamond path, i.e. for each BDD node there is exactly one path from the entry point to that node. The following property holds:

BDDBC_No_Diamond_Indep_Implies_MCDC For a BDD with no diamond, if conditions are independent, then BDD branch coverage implies Unique Cause MC/DC and Masking MC/DC.

Let's consider a condition C. Since we have BDD branch coverage, all possible paths starting at C have been taken (by recurrence on path length, taking advantage of no cycles and no diamonds).

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 outcomes of the decision. Let's call them PCT and PCF.

These paths are disjoint: any condition appearing in one is not evaluated in the other (because of no-diamond).

These two paths are parts of paths PT and PF from the BDD entry point to either outcome, and they cannot differ on the part of the path from the entry point to C.

So, PT and PF differ in C, in no other condition before C, and in no other non-masked condition after C, so they prove independent influence of C over the decision.

This holds for each condition in the decision, so Unique Cause MC/DC is proved; and since Unique Cause is stronger than Masking MC/DC, Masking MC/DC is proved as well.

2.6 Non-equivalence case

The general idea of this proof is to show that, for any BDD with a diamond, we can build a set of evaluations that covers the BDD branches in such a way that there is at least one condition for which MC/DC is not met.

If we want to have a proof that will work for any definition of MC/DC, we cannot rely on a failure of the independence criteria; *to independently affect* means something different in Unique Cause MC/DC or in Masking MC/DC. So we should rather rely, if it is possible, on the set of properties that these criteria have in common; a sort of *greatest common divisor* of the two criteria. Let us define this Weak MC/DC criteria as follow:

- every possible outcome of the decision have been tried;
- each condition in the decision has taken on every possible outcome;
- each condition in the decision is shown to affect the outcome of the decision.

The only difference with MC/DC here is that Weak MC/DC does not care if the condition *independently* affects the outcome of the decision; whatever *independently* means. Weak MC/DC is weaker than any other MC/DC criteria. So, if a set of evaluations does not satisfy Weak MC/DC, it won't satisfy Unique Cause MC/DC or Masking MC/DC.

The formal definition would be:

DEFINITION 3 *Given a decision D, a pair of truth vectors satisfies Weak MC/DC for a condition c if and only if, in their influence tree, the node c and the root node are both colored in True.*

A test set satisfies Weak MC/DC for a decision D if, and only if, for each condition in cond(D), there exists a pair of tests in the test set that satisfies Weak MC/DC.

It turns out that we can build a set of evaluations such that BBDBC is reached, but not MC/DC, when the BDD has a diamond. Let's take the canonical example:
$(A$ `and then` $B)$`or else` C
and these evaluations:

A	B	C	(A `and then` B) `or else` C
T	T	x	T
T	F	T	T
F	x	F	F

This set covers this decision for BDDBC. However, this does not meet Weak MC/DC; whenever B is evaluated, the decision outcome is True. So this cannot show that B affects the outcome of the decision.

Let us give a general idea of how we would prove that in the general case. First, remember that we are dealing with reduced ordered BDDs; so each node can be ordered. This property allows us to have a proper concept of “last diamond node” and its “last parent”. In our example, the last diamond node would be C, its last parent B.

Now, we would show that the last parent of the last diamond node has an interesting property: one of its exit edge is always connected directly to an outcome (for B, it is outcome True). Its other exit edge being connected to the last diamond node (obviously), it is possible to cover this edge in such a way that the outcome of the corresponding evaluation is the same as the “direct outcome”; e.g. when evaluating B to False, we would evaluate C to True and exit on True. We know that this is possible using the property that, from each node of the BDD (and, in this case, the diamond C), there exists at least one evaluation that reaches outcome True and at least one that reach outcome False.

This means that we can cover all exit and incoming edges of the last parent of the last diamond in such a way that the evaluations *always* exit on the same outcome; and, from the property of its exit edges, we can see that this set of evaluations can be completed to reach BDDBC *without evaluating this node anymore*. This builds a BDD coverage that does not satisfy Weak MC/DC for this node.

The following sections will detail this proof.

2.6.1 Last diamond

We have previously built a function index(D,C) that defines a full order over the nodes of a (RO)BDD. Based on this function, we can define the following entities:

- in a BDD, let the last node be the node with the greatest index;
- let a diamond node be a node with more than one parent;
- let the last diamond node of a decision be the diamond with the greatest index;
- let the last parent of a node be the parent with the greatest index.

Some examples from our canonical case $D ::= (A$ `and then` $B)$`or else` C:

- index(D,A) = 1, index(D,B) = 2, index(D,C) = 3;
- the last diamond node is C; it is also the last node;
- the last parent of C is B.

We have the following properties:

Lemma 2.6.1 *The two exit edges of the last node are connected to both outcomes.*

Lemma 2.6.2 *If a BDD contains diamonds, then the last parent of the last diamond has an exit edge that is directly connected to an outcome.*

This outcome will be called the direct outcome (of the last of parent of the last diamond). The edge connected to the the direct outcome will be called the direct exit edge.

This can be seen in our example:

- the last node being C, its exit edges are connected directly to True and False;
- the last parent of the last diamond being B; when it is True, the outcome True is reached.

The two properties can be proved by structural induction on the BDD. The first is the easiest one and we will let it as an exercice for the reader. Here is a proof of the second one:

- if $D ::= C$ (simple condition decision):
  the BDD does not contain any diamond, so the property is trivially true in this case.
- if $D ::=$ `not` $D1$:
  if D1 does not contain diamond, D does not either, the propery is trivial as well. If it does contain diamond, well, only outcome labels are different between BDD(D) and BDD(D1), so the property holds for D if it holds for D1.
- if $D ::= DL \star DR$, supposing that the property holds for DL and DR; four cases then:
  - If no diamonds were in BDD(DL) and BDD(DR), and none were introduced by building BDD(D) from them: then BDD(D) contains no diamonds, the property is trivial.

- If DR contains a diamond: then the last diamond node of D is the last diamond node in DR. By construction, BDD(DR) is a sub-tree of BDD(D), the exit edges of the last diamond node are the same in BDD(DR) and in BDD(D). So if the property holds for DR, it holds for D.
- If DR contains no diamonds, and if building BDD(D) from BDD(DR) and BDD(DL) introduces a new diamond: this new diamond node has to be root(DR), as only this node has gained new incoming edges: in Build_BDD, the exit edges of BDD(DR) labeled “not SC” are connected to BDD(DL)’s root. Therefore, its parent nodes are all in cond(DL). And the last node of BDD(DL) is one of these parent nodes; by property 2.6.1, it has an exit edge to “not SC” in BDD(DL). Its other exit edge will remain unchanged in BDD(D) and will be connected to an outcome (“SC”). This last node of BDD(DL) is also the last parent of this diamond node, as no other nodes in cond(DL) have an greater index. And (finally), this introduced diamond node is D’s last diamond node, as DR contains no diamonds (initial hypothesis). So we have identified the last parent of the last diamond and showed that one of exit edge is connected to an outcome in BDD(D); that’s the property that we were trying to prove.
- If DR contains no diamonds, and if no diamonds were introduced when building BDD(D), but if DL contains a diamond: the last diamond node in D is the same node as the last diamond node in DL. The last parent of the last diamond in DL has an exit edge that connects to SC; otherwise, it would be connected to BDD(DR)’s root when building BDD(D), and as the last node would also be connected to this root node (it also has an exit edge to SC in DR, as a consequence of property 2.6.1), BDD(DR)’s root would have at least two fathers in D, which contradicts the hypothesis that no diamonds were introduced. So this exit edges of the last parent of the last diamond in DL (and D) will still be connected to an outcome after building D. So the property is true in this case as well.

The property 2.6.2 has now been established in all possible cases. This proof will actually be quite useful as its structure will be used to build a coverage that satisfies BBDBC but not Weak MC/DC.

2.6.2 Path though BDDs - Conventions

Some conventions first to help us manipulate paths through BDD:

- For two paths pl, pr into two different BDDs, concat(pl, pr) is the concatenation of these two paths obtained by replacing the outcome of pl by the first node in pr.

- For two set of paths PL, PR through two different BDDs, merge(PL, PR) is the set built by doing an arbitrary mapping of each element of PL to each element of PR, and concatenating each pair; as these two sets may not have the same number of elements, all the remaining elements of the greatest set will be mapped to one arbitrary element of the smallest set.

- For any set of paths PS through a BDD (each going from the root node to an outcome), let us call path_to(True, PS) the subset of paths reaching outcome True, and path_to(False, PS) the subset reaching False. We have the obvious property:

  $PS = path\_to(True, PS) + path\_to(False, PS)$

2.6.3 Building a BDD branch coverage that does not verify Weak MC/DC

For a given decision D, our coverage will be ensured by the union of three sets of paths Short_Circuit_LPLD(D), Long_Circuit_LPLD(D) and LPLD_Not_Evaluated(D), which verifies the following set-specific invariants:

Short_Circuit_LPLD_Invariant If BDD(D) is diamondless, Short_Circuit_LPLD(D) is empty; otherwise, it contains a non-empty set of paths, each reaching the last parent of the last diamond node and then exiting on its direct outcome.

Long_Circuit_LPLD_Invariant If BDD(D) is diamondless, Long_Circuit_LPLD(D) is empty; otherwise, it contains a non-empty set of paths, each reaching the last parent of the last diamond node, then going to the last diamond node, and then reaching the same outcome as Short_Circuit_LPLD(D).

LPLD_Not_Evaluated_Invariant No path in LPLD_Not_Evaluated(D) evaluates the last parent of the last diamond node.

...plus one other "global" invariants:

BDDBC_Coverage_Invariant The union of these three sets covers each edges of the considered BDD.

As a consequence of this last invariant, we will now call BDDBC_Coverage the union of these three sets:

$$\begin{aligned}BDDBC\_Coverage(D) = & Short\_Circuit\_LPLD(D)\\ & + Long\_Circuit\_LPLD(D)\\ & + LPLD\_Not\_Evaluated(D)\end{aligned}$$

Note also that an other property falls naturally from LPLD_Not_Evaluated_Invariant and BDDBC_Coverage_Invariant:

Incoming_Edges The union of Long_Circuit_LPLD and Short_Circuit_LPLD covers all incoming edges of the last parent of the last diamond node.

Anyway, here is the definition, case by case, of a recursive build procedure that builds such sets from a decision D. As always when doing a structural induction, it assumes that all its sub-decisions have such sets verifying these invariants (not considering D as a sub-decision of D, obviously; “strict” sub-decisions):

Build_BDD_Cov.Condition The sub-decision is a single condition decision, it is of the form:
$D ::= C$
In this case:

$$\begin{aligned}Short\_Circuit\_LPLD(D) = Long\_Circuit\_LPLD(D) = \{\}\\ LPLD\_Not\_Evaluated(D) = \{C- > True, C- > False\}\end{aligned}$$

Build_BDD_Cov.NOT The sub-decision is of the form:
$D ::= \texttt{not}\ D1$

Then Short_Circuit_LPLD(D) is built by switching the outcome of each path contained in Short_Circuit_LPLD(D1). Same operations for Long_Circuit_LPLD(D) and No_LPLD(D).

Build_BDD_Cov.Short_Circuit_Operator The sub-decision is of the form:
$D ::= DL \star DR$

Four sub-cases:

- No diamonds:
  No diamonds in BDD(DL) and BDD(DR), and none were introduced by

building BDD(D) from them. Then:

$$\begin{aligned}
Short\_Circuit\_LPLD(D) &= \{\} \\
Long\_Circuit\_LPLD(D) &= \{\} \\
LPLD\_Not\_Evaluated(D) &= path\_to(SC, BDDBC\_Coverage(DL)) \\
&\quad + merge(path\_to(notSC, BDDBC\_Coverage(DL)), \\
&\quad\ \ BDDBC\_Coverage(DR))
\end{aligned}$$

- Diamond in DR:
  i.e. Short_Circuit_LPLD(DR) and Long_Circuit_LPLD(DR) contain at least one element each. Let p_dl be an arbitrary path to “not SC” in BDD(DL), taken from BDDBC_Coverage(DL). In BDD(D), it corresponds to a path to root(DR). Then:

$$\begin{aligned}
Short\_Circuit\_LPLD(D) &= merge(p\_dl, Short\_Circuit\_LPLD(DR)) \\
Long\_Circuit\_LPLD(D) &= merge(p\_dl, Long\_Circuit\_LPLD(DR)) \\
LPLD\_Not\_Evaluated(D) &= path\_to(SC, BDDBC\_Coverage(DL)) \\
&\quad + merge(path\_to(notSC, BDDBC\_Coverage(DL)), \\
&\quad\ \ LPLD\_Not\_Evaluated(DR))
\end{aligned}$$

- Last diamond introduced:
  i.e. no diamonds in BDD(DR) and a diamond has been created when building BDD(D). In this case:
  - $Short\_Circuit\_LPLD(DR) = Long\_Circuit\_LPLD(DR) = \{\}$
  - the last diamond node in D is root(DR).

  Let SC_DL be the subset of paths in path_to(SC, BDDBC_Coverage(DL)) that evaluates the last node in BDD(DL). Similarly, let LC_DL be the subset of paths in path_to(not SC, BDDBC_Coverage(DL) that evaluates the last node in BDD(DL); in BDD(D), this one corresponds to a path to root(DR). Let $REST\_DL = BDDBC\_Coverage(DL) - (SC\_DL + LC\_DL)$ be the subset of paths in BDDBC_Coverage(DL) that do not evaluates this last node. Let sc_dr be an arbitrary path in DR exiting on SC.
  Then the coverage for BDD(D) is built as follows:

$$\begin{aligned}
Short\_Circuit\_LPLD(D) &= SC\_DL \\
Long\_Circuit\_LPLD(D) &= merge(LC\_DL, sc\_dr) \\
LPLD\_Not\_Evaluated(D) &= path\_to(SC, REST\_DL) \\
&\quad + merge(path\_to(notSC), REST\_DL), \\
&\quad\ \ BDDBC\_Coverage(DR))
\end{aligned}$$

(Note that we are allowed to merge path_to(not SC, REST_DL) because it is non-empty; there is at least one other node that has an exit edge directly connected to “not SC” in BDD(DL), otherwise no diamond would have been created; so it must have at least one path that does not evaluate the last node in BDD(DL); that property gives us the right to use the merge operation. Same thing for LC_DL and SC_DL that are both non-empty thanks to property 2.6.1).

- Diamond in DL only:
  i.e. no diamonds in BDD(DR), no new diamonds introduced when building BDD(D), but at least one diamond in BDD(DL). The last parent of the last diamond in DL has its “direct” exit edge that connects to SC; otherwise, it would have been connected to BDD(DR)’s root when building BDD(D), and as the last node would also be connected to this root node (it also has an exit edge to SC in DR, as a consequence of property 2.6.1), BDD(DR)’s root would have at least two fathers in D, which contradicts the hypothesis that no diamonds were introduced. Let sc_dr be an arbitrary path in DR exiting on SC. Then the coverage for BDD(D) is built as follows:

$$\begin{aligned}
Short\_Circuit\_LPLD(D) &= Short\_Circuit\_LPLD(DL) \\
Long\_Circuit\_LPLD(D) &= merge(Long\_Circuit\_LPLD, sc\_dr) \\
LPLD\_Not\_Evaluated(D) &= path\_to(SC, LPLD\_Not\_Evaluated(DL)) \\
&\quad + merge(path\_to(notSC, LPLD\_Not\_Evaluated(DL), \\
&\quad BDDBC\_Coverage(DR))
\end{aligned}$$

Our sets are now defined in all possible cases. It is quite easy to check that Short_Circuit_LPLD_Invariant, Long_Circuit_LPLD_Invariant, LPLD_Not_Evaluated_Invariant and BDDBC_Coverage_Invariant are enforced by this build procedure; each one falls so obviously from the construction (in every case) that it will be quite painful to elaborate.

So we can build a BDD coverage in such a way that, if there is a diamond in the BDD, then the decision will have the same outcome for any evaluation where the last parent of the last diamond is evaluated. This means that this BDD coverage does not imply Weak MC/DC; that which was to be demonstrated.

2.7 Conclusion

We have therefore proved the following property:

THEOREM 3 *Given a decision D, branch coverage of the BDD implies MC/DC if, and only if, there is no diamond in the BDD (i.e. no node of the BDD is reachable through more that one path from the root).*

Intuitively, BDDBC is a local property of BDD traversals (i.e. of evaluations of the decision): it is evaluated individually for each BDD node, without respect to how the BDD node was reached. In contrast, MC/DC is a non-local property, since it involves the complete path through the BDD. To establish independent influence of condition C, it is necessary to study what happens for two values of C leading to different outcomes, *with all other conditions fixed or masked*.

3 A second characterization of the equivalence case

In this section we provide an alternative (equivalent) property that characterizes cases where BDD branch coverage implies MC/DC:

THEOREM 4 *Given a decision D, BDD branch coverage implies MC/DC 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.*

The negative normal form is obtained by rewriting the expression using De Morgan's laws so that negations apply only to atomic conditions (and not to more complex subexpressions).

We prove this theorem by showing that this alternative characterization is equivalent to having no diamond in the associated BDD. Theorem 4 follows by application of Theorem 3.

Let's consider a decision D and its BDD.

There is a diamond in a BDD if and only if there is at least one node such that the number of paths to reach it from root is strictly greater than 1; we will use this caracterization to prove our property.

Proof of the simplified case:

Consider first the case where decision D does not contain any negation. Each node in the BDD corresponds to a sub-decision D' in D (i.e. the root of each sub-decision is a node in D's BDD).

$$\begin{tikzcd}
& & DL \arrow[dl, "F"'] \arrow[dr, "T"] & \\
& DR \arrow[dl, "F"'] \arrow[dr, "T"] & & T \\
F & & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
& DL \arrow[dl, "F"'] \arrow[dr, "T"] & & \\
F & & DR \arrow[dl, "F"'] \arrow[dr, "T"] & \\
& F & & T
\end{tikzcd}$$

(a) *DL* `or else` *DR* (b) *DL* `and then` *DR*

Figure 2: case 2 and 3

Proof of reverse implication (simplified case):

Let's call NF(D) 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). We then have three cases:

case 1: D = A (atom)
$NF(D) = NT(D) = 1$

case 2: D = *DL* `or else` *DR*
BDD(D) is constructed from BDD(DL) and BDD(DR). It is shown in Figure 2(a). Then:
$NF(D) = NF(DL) * NF(DR)$
$NT(D) = NT(DL) + NF(DL) * NT(DR)$

case 3: D = *DL* `and then` *DR*
BDD(D) is constructed from BDD(DL) and BDD(DR). It is shown in Figure 2(b). Then:
$NF(D) = NF(DL) + NT(DL) * NF(DR)$
$NT(D) = NT(DL) * NT(DR)$

Lemma 3.0.1 *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 $NF(D) \geqslant 2$.

Lemma 3.0.2 *NF and NT are monotonic functions, i.e. if D' is a sub-decision of D, then we have $NF(D') \leqslant NF(D)$ and $NT(D') \leqslant NT(D)$.*

Proofs of Lemmas 3.0.1 and 3.0.2 are on structural induction on the form of decisions, based on the three cases distinguished above.

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

By Lemma 3.0.1, we know that $NT(D') \geqslant 2$. By Lemma 3.0.2, we know that $NT(DL) \geqslant NT(D')$.

Then, in BDD(D), the paths reaching the DR's root node are exactly those paths in BDD(DL) reaching True, by construction; the number of these paths is NT(DL).

Thus, DR's root node in the BDD(DL) is reachable by more than one path. As a consequence, there is a diamond in this BDD.

Similarly, if a decision D of the form `or else` contains a sub-decision D' of the form `and then` on the left-hand side, we can exhibit a node in the BDD(D) that is reachable by more than one path.

By contraposition, if there are no diamonds in the BDD associated to a decision D, then D has no `or else` sub-decision on the left of its `and then` sub-decisions, and no `and then` sub-decision to the left of its `or else` sub-decision.

Proof of implication (simplified case):

Now suppose 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 3.0.3 *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 3.0.3 is on structural induction on the form of decisions, based on the 3 cases distinguished above.

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

Proof of the general case:

In the general case, decision D may contain `not`sub-decisions. Then, consider D' the negation normal form of D. Since D and D' are represented by the same BDD, Theorem 4 follows.

4 General stateless MC/DC assessment

This section discusses a code transformation after which stateless BDDBC traces carry sufficient information to assess MC/DC coverage.

5 Automated verifications

References

[Chi01] 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.

[FAA07] 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.