MC/DC for Space

A new Approach to Ensure MC/DC Structural Coverage with Exclusively Open Source Tools

Thomas Wucher, Andoni Arregui

2021-10-07

ESA Software Product Assurance Workshop 2021

GTD GmbH
<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Motivation</td>
</tr>
<tr>
<td>2. Necessary Concepts</td>
</tr>
<tr>
<td>3. Our Method and Tool</td>
</tr>
<tr>
<td>4. Conclusion</td>
</tr>
</tbody>
</table>
Motivation
How many tests do we need for the following code?

- 1 statement
- How many branches?
- How many decisions?
- 1, 2, 3, 4 tests?

```c
#include <stdbool.h>

bool test(bool a, bool b, bool c) {
    return b && c || a;
}
```
Current Approach to MC/DC

- **MC/DC** is a popular **structural coverage** metric required by many standards, e.g., DO-178.
- It is a **demanding metric** as it requires **many more tests** than branch/decision coverage.
- Its assessment is usually done with **proprietary tools** and **late in the project**.

### MC/DC as an ECSS Requirement

- **Definition** in E-ST-40 §3.2.18
- MC/DC is one of the structural coverage requirements of ECSS E-ST-40 (§5.8.3.5.b), Q-ST-80 (§6.3.5.2), and Q-HB-80-04 (Table 5-3):
  - the coverage percentage must be agreed for all categories except CAT A
    →Often the agreement means 0% for Cat B and lower
  - the assessment is to be done on the source code
  - the coverage must be achieved by unit, integration, and validation testing
## E-ST-40 §5.8.3.5.b

<table>
<thead>
<tr>
<th>Code coverage versus criticality category</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code statement coverage</td>
<td>100%</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
</tr>
<tr>
<td>Source code decision coverage</td>
<td>100%</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
</tr>
<tr>
<td>Source code modified condition and decision coverage</td>
<td>100%</td>
<td>AM</td>
<td>AM</td>
<td>AM</td>
</tr>
</tbody>
</table>

NOTE: “AM” means that the value is agreed with the customer and measured as per ECSS-Q-ST-80 clause 6.3.5.2.

## Q-HB-80-04 Table 5-3

<table>
<thead>
<tr>
<th>Statement Coverage (Source Code)</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement Coverage (Object Code)</td>
<td>P.D.</td>
<td>P.D.</td>
<td>P.D.</td>
</tr>
<tr>
<td>Decision Coverage (Source Code)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Modified Condition &amp; Decision Coverage (Source Code)</td>
<td>P.D.</td>
<td>P.D.</td>
<td>P.D.</td>
</tr>
</tbody>
</table>
Figure 1: Minimum probability of logic error detection vs. number of conditions

What’s the Problem

- It is deemed that the testing effort is expensive, as it requires many more tests than simple decision coverage.
- The qualified proprietary tools used for the assessment are expensive and using them requires further effort.
- The tools cannot be freely redistributed within delta-qualification kits (e.g., MLFS, RTEMS SMP)

This leads to

- Trying to avoid MC/DC altogether within a project
  → Agree on 0% MC/DC
- Imposing unnatural coding rules
  → only 1 condition per decision
- Lower the MC/DC goal to make it meaningless
  → achieve 80% MC/DC
We propose

To produce a **method** and a **tool** that:

- Helps developers to assess the achieved MC/DC coverage
- Allows *normal* source code writing without *impractical* coding rule restrictions.
- Allows an open source based approach to ensure the free distribution of the complete unit and validation testing workflow for delta-qualification tool-kits.
Advantages of an Open Source Approach

An open source based system allows us to:

- Deploy a **Continuous Integration** system *without* long-term *licensing costs* capable of:
  - building the software
  - unit-testing the software
  - validation testing the software
  - assessing product assurance metrics including MC/DC

- **Distribute** the complete **Continuous Integration** system together with the test-suite for project specific **delta qualification**.

- Projects can benefit from this approach as commonly available open source tools will suffice for MC/DC assessment.
Necessary Concepts
For structural coverage we need:

- 1 test for statement coverage
- 2 tests for decision coverage
- 4 tests for MC/DC
- We need to show that every condition in the decision correctly contributes to the result.

```c
#include <stdbool.h>

bool test(bool a, bool b, bool c)
{
    return b && c || a;
}
```
There are several kinds of MC/DC, depending on the definition:

- Unique cause MC/DC
- Unique cause + Masking MC/DC
- Masking MC/DC

**Our MC/DC**

We will deal here with masking MC/DC, as it is the most practical one, is the one currently used by DO-178C, and matches well with short-circuiting in the C language.
MC/DC for our Example

Truth table
\[ f(a, b, c) = b \land c \lor a: \]

<table>
<thead>
<tr>
<th>case</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

```c
#include <stdbool.h>

bool test(bool a, bool b, bool c)
{
    return b && c || a;
}
```
MC/DC for our Example

Truth table with masking

\[ f(a, b, c) = b \&\& c \| a \]

<table>
<thead>
<tr>
<th>case</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

```
#include <stdbool.h>

bool test(bool a, bool b, bool c)
{
    return b && c || a;
}
```
## MC/DC for our Example

### Truth table with masking

\[ f(a, b, c) = b \&\& c \| a: \]

<table>
<thead>
<tr>
<th>case</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>?</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>?</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>FALSE</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>TRUE</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>?</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Test pairs:
- to test \(b\) with independence: \(\{1, 7\}\)
- to test \(c\) with independence: \(\{5, 7\}\)
- to test \(a\) with independence: \(\{5, 6\}\)

Thus, we can take cases: \(\{1, 5, 6, 7\}\) to achieve full MC/DC.
The free tool gcov shows us:

- Full statement and branch coverage with cases: \{1, 6, 7\}
- 3 test cases are enough:
  - 1 more than decision coverage
  - 1 less than MC/DC
- The 6 covered branches are called OBC: Object Branch Coverage

\[\text{gcov by itself is not enough}\]

gcov alone does not show if full MC/DC is achieved even if it requires more than just decision coverage.
**Assembler vs. AST vs. BDD**

**Basic Block to check 'b':**

```
ld [%fp+72], %g1
cmp %g1, 0
be .LL2
```

**Basic Block to check 'c':**

```
ld [%fp+76], %g1
cmp %g1, 0
bne .LL3
```

**Basic Block to check 'a':**

```
ld [%fp+68], %g1
cmp %g1, 0
be .LL4
```

**LL3:**

```
move 1, %g1
b .LL5
```

Return 1

**LL4:**

```
move 0, %g1
```

Return 0

**LL5:**

```
and %g1, 0xFF, %g1
```

**MC/DC for Space 15**

```
-CompoundStmt 0x55e01c847df8 <line:5:1, line:7:1>
  -ReturnStmt 0x55e01c847de8 <line:6:2, col:19>
    | -BinaryOperator 0x55e01c847dc8 <col:9, col:19> 'int' '||'
    | -BinaryOperator 0x55e01c840480 <col:9, col:14> 'int' '&&'
    | -ImplicitCastExpr 0x55e01c840450 <col:9> 'int' <LValueToRValue>
    | -DeclRefExpr 0x55e01c840410 <col:9> 'int' lvalue ParmVar 0x55e01c8401a0 'b' 'int'
    | -ImplicitCastExpr 0x55e01c840468 <col:14> 'int' <LValueToRValue>
    | -DeclRefExpr 0x55e01c840430 <col:14> 'int' lvalue ParmVar 0x55e01c840220 'c' 'int'
    | -ImplicitCastExpr 0x55e01c847db0 <col:19> 'int' <LValueToRValue>
    -DeclRefExpr 0x55e01c847d90 <col:19> 'int' lvalue ParmVar 0x55e01c840120 'a' 'int'
```
Our Method and Tool
For our example function $f(a, b, c) = b \&\& c \lor a$:

- The compiler generates a BDD (Binary Decision Diagram) in object code when invoked without optimization (-O0)
- The BDD has 6 edges as the OBC branches count in gcov
- The BDD has 3 paths/endpoints as the tests needed to cover 100% in gcov
If we transform our example function to \( f(a, b, c) = a \parallel b \&\& c \):

- The BDD has 6 edges
  as the OBC branches count in gcov
- The BDD has 4 paths/endpoints
  as the tests needed to achieve MC/DC

Treelike BDDs show MC/DC

For treelike BDDs it is proven\(^2\) that OBC implies MC/DC and gcov is able to show this.

Assessing Structural Coverage with gcov

The free tool gcov shows us:

- Full statement but incomplete object branch coverage with cases: \( \{1, 6, 7\} \)
- 3 test cases are not enough:
  - 2 more than decision coverage are required now
  - This is proven to be equivalent to MC/DC
- One test case (\( \{5\} \) true, false, false) is missing to achieve 100% MC/DC
Features of our tool:

- Enables us to assess all decisions in C
- Detects decisions with non-treelike BDDs
- Proposes reordering to achieve a treelike BDD
- Enables the MC/DC assessment with gcov
- Python and clang based
- Makes use of the clang AST (Abstract Syntax Tree)

```
$ python3 mcdc_checker.py tests/example.c
Processing file tests/example.c
None tree-like decision at:
  tests/example.c line 4, column 12
  Found solution: ['a', 'b', 'c']

Using the found solution, we can reorder the condition to a || b && c
```
• We have run our tool on several code bases finding few non-treelike BDDs:
  • On RTEMS SMP
  • On the Mathematical Library for Critical Systems
• This indicates that for many source code pieces gcov is already showing MC/DC.
• The source code changes required to enable gcov to show MC/DC are minimal.
• Other evaluations on industrial code show non-tree like BDDs to be less than 1% of all decisions.
Conclusion
Conclusions on the Method and the Tool

Advantages

- The method has been mathematically proven.
- Our tool enables MC/DC assessment with the open source tool gcov.
- The required source code changes have very little impact and ensure a good maintainability.
- The structural coverage analysis including MC/DC can be done on target.
- The tool can be freely and easily integrated into existing CI/CD pipelines.

Cautions

- The tool may in some cases not find a solution
  → then manual assessment is still needed.
- The gcov assessment is done on a compilation without optimization (i.e., -00).
- The gcov assessment requires code instrumentation.
- The tool is not yet qualified.
These ideas, methods, and tools heavily base on the work of others.

In particular we want to thank Andreas Jung (ESA ESTEC) for stimulating us to find an open source solution for this problem and to Thanassis Tsiodras (ESA ESTEC) whose initial works on using gcov for this purpose and discussions led us to these results.