Test Adequacy Assessment using Control Flow and Data Flow

Lecture 15

MC/DC: Example

Consider the following requirements:
- \( R_{1.1} \): Invoke fire-1 when \((x<y) \land (z^2 > y) \land (\text{prev} = \text{"East"})\).
- \( R_{1.2} \): Invoke fire-2 when \((x<y) \land (z^2 \leq y) \land (\text{current} = \text{"South"})\).
- \( R_{1.3} \): Invoke fire-3 when none of the conditions above is true.
- \( R_2 \): The invocation described above must continue until an input Boolean variable becomes true.

MC/DC: Example (cont.)

Program

```c
1 begin
2 float x, y, z;
3 direction do;
4 string prev, current;
5 bool done;
6 input (done);
7 current="North";
8 while (!done) <- Condition C1,
9 input (d);
10 prev=current, current=f(d); // Update
11 if (x<y) and (z^2 > y) and
12 (prev="East") <- Condition C2.
13 fire-1(x, y);
14 else if (x<y and (z^2 \leq y) or
15 (current="South")! <- Condition C3.
16 fire-2(x, y);
17 else
18 input (done);
19 }
20 output("Firing completed.");
21 end
```

Step 1: Tests from requirements

<table>
<thead>
<tr>
<th>Test</th>
<th>Requirement</th>
<th>done</th>
<th>d</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>( R_{1.2} )</td>
<td>false</td>
<td>East</td>
<td>10</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>( R_{1.1} )</td>
<td>false</td>
<td>South</td>
<td>10</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>( f_3 )</td>
<td>( R_{1.3} )</td>
<td>false</td>
<td>North</td>
<td>10</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>( f_4 )</td>
<td>( R_2 )</td>
<td>true</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Adequate with statement/block coverage? ☑
- Adequate with decision coverage? ☑
- Adequate with condition coverage? ❌

- \( R_{1.1} \): Invoke fire-1 when \((x<y) \land (z^2 > y) \land (\text{prev} = \text{"East"})\).
- \( R_{1.2} \): Invoke fire-2 when \((x<y) \land (z^2 \leq y) \land (\text{current} = \text{"South"})\).
- \( R_{1.3} \): Invoke fire-3 when none of the conditions above is true.
- \( R_2 \): The invocation described above must continue until an input Boolean variable becomes true.
MC/DC: Example (cont.)

<table>
<thead>
<tr>
<th>Requirement</th>
<th>done</th>
<th>cond</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>false</td>
<td>East</td>
<td>10</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>$R_2$</td>
<td>false</td>
<td>South</td>
<td>10</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>$R_3$</td>
<td>false</td>
<td>North</td>
<td>10</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>$R_4$</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Need to modify the test set to make it adequate with respect to the condition coverage.

- Needed tests for $C_2$
  - Independent effect of $x < y$
    - Reuse $t_1$
    - Make $x < y$ false, $z^2 > y$ true, $prev = "East"$ true, and $C_2$ false.
  - Independent effect of $z^2 > y$
    - Reuse $t_2$
    - Make $x < y$ true, $z^2 > y$ false, $prev = "East"$ true, and $C_2$ false.
  - Repeat for $C_3$

- Analysis of decisions and conditions

- Need to modify the test set to make it adequate with respect to the condition coverage.

- Needed tests for $C_2$
  - Independent effect of $x < y$
    - Reuse $t_1$
    - Make $x < y$ false, $z^2 > y$ true, $prev = "East"$ true, and $C_2$ false.
  - Independent effect of $z^2 > y$
    - Reuse $t_2$
    - Make $x < y$ true, $z^2 > y$ false, $prev = "East"$ true, and $C_2$ false.
  - Repeat for $C_3$
MC/DC: Example (cont.)

- Test set T3 is adequate with respect to MC/DC coverage criterion.
  - Note again that sequencing of tests is important in this case
    - t1 and t7 have same input values but lead to different effects in subsequent test execution.

<table>
<thead>
<tr>
<th>Test set T3 for Program P9.14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>e</td>
</tr>
<tr>
<td>f</td>
</tr>
<tr>
<td>g</td>
</tr>
<tr>
<td>h</td>
</tr>
</tbody>
</table>

Error Detection and MC/DC Adequacy

Consider three error types in a compound condition
- Missing condition
- Incorrect Boolean operator
- Mixed type
  - 1* conditions missing and 1* Boolean operators are incorrect

Assumption
- Given a test set T for program P with respect to requirements R
- T is adequate with condition coverage criterion

Question
- If we enhance the test set to be adequate with the MC/DC coverage, will T reveal errors?

Error Detection and MC/DC Adequacy: Example

Suppose that condition C=C1 ∧ C2 ∧ C3 has been coded as C’=C1 ∧ C3.
- Four tests that form an MC/DC adequate set are in the following table.

<table>
<thead>
<tr>
<th>Test</th>
<th>Condition</th>
<th>C’</th>
<th>Error detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>true, true, true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>b</td>
<td>false, false, false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>c</td>
<td>true, true, false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>d</td>
<td>false, false, true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

C2 values can be either true or false for both test cases
- Note that the adequate test set is developed for the condition as coded in the program, and not the correct decision.

Error Detection and MC/DC Adequacy: Example

The following test set can detect error.

Satisfying the MC/DC adequacy criteria does not necessarily imply revealing errors
- However, an MC/DC adequate test will be more likely to reveal errors than a decision or condition-coverage adequate test.
Short-Circuit Evaluation and Infeasibility

- Short-circuit evaluation
  - Partially evaluate a compound condition if possible. (also known as lazy evaluation)
  - Consider \( C = C_1 \land C_2 \)
    - If \( C_1 \) is false, \( C_2 \) value may not be considered. (i.e., \( C_1 = \text{false} \) and \( C_2 \) value may be infeasible.)

- Dependencies between decisions might also lead to infeasibility.
  - \( \text{Infeasible condition } A < 5 \)

- What if \( \text{foo()} \) has side effect?

Consider \( C = C_1 \land C_2 \).

Infeasibility and Reachability

- Infeasibility \( \neq \) Reachability

Example

```c
1 int A, B, C;
2 input (A, B, C);
3 if(A>10 and B>30) {
  4 S1 = f1(A, B, C)
  5 if(A<5 and B>10){
  6 S2 = f2(A, B, C);
  7 }
```

- Second decision is not reachable.
- But, what if the \( A+1 \) overflows?

Tracing Test Cases to Requirements

- Test trace-back
  - The task of relating the new test case to the requirements is known as test trace-back
  - \textit{i.e., what portion of the requirements are tested when the program under test is executed against the newly added test case?}

- Advantage
  - Can help determine redundant test cases
  - Could reveal errors or ambiguity in requirements
  - Assists with the process of documenting tests against requirements

Data-Flow Coverage
Control Flow-based Coverage isn’t Enough

- Testing all conditions and statement blocks is often inadequate
  - i.e., the test set does not reveal errors.
- Data flow-based test adequacy criteria can be useful in improving tests adequate with respect to control-flow based criteria.

Test Enhancement using Data Flow

- Neither of the two tests does not force the definition of z at line 7 to be tested at line 9.

### A MD/DC-adequate test set

```
1 begin
2   int x, y, float z;
3   input (x, y);
4   z = 0;
5   if (x! = 0)
6     z = x*y;
7     else a = x - y;
8   if (y! = 0) — This condition should be (y! = 0 and x! = 0)
9     z = x/z;
10    else z = x*z;
11   output(z);
12 end
```

Test Enhancement using Data Flow (cont.)

- The error would be revealed if we test all feasible definition and use pairs for z.
- Example test set
  - Verify that the following test set covers all def-use pairs of z and reveals the error.

<table>
<thead>
<tr>
<th>Test</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>def-use pairs covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>(4, 7), (7, 10)</td>
</tr>
<tr>
<td>t₂</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
<td>(4, 6), (6, 9)</td>
</tr>
<tr>
<td>t₃</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>(4, 7), (7, 9)</td>
</tr>
<tr>
<td>t₄</td>
<td>1</td>
<td>0</td>
<td>1.0</td>
<td>(4, 6), (6, 10)</td>
</tr>
</tbody>
</table>
Definitions and Uses

- **Definitions:** Assigning values to a variable
- **Uses:** Referencing variable values

**Examples**
- Assignment statement \( x = y + z \)
  - Defines variable \( x \) and uses variables \( y \) and \( z \).
- Declaration `int x, y, A[10];`
  - Defines three variables
- Statement `scanf("%d %d", &x, &y)`
  - Defines variables \( x \) and \( y \).
- Statement `printf("Output: %d \n", x+y)`
  - Uses variables \( x \) and \( y \).

Definitions and Uses (cont.)

- Arrays
  - Consider the following example:
    ```c
    int A[10];
    A[i]=x+y;
    ```
    - Defines array \( A \)
    - Defines \( A[i] \) (not entire array), uses \( i, x, y \)

Definitions and Uses (cont.)

- A parameter \( x \) passed as **call-by-value** to a function
  - A use of, or a reference to, \( x \).
- A parameter \( x \) passed as **call-by-reference**
  - A definition and use of \( x \)

**Example**
- Defines pointer variable \( z \)
  ```c
  z=x;
  y=x+1;
  ```
- Defines \( y \), uses \( z \)
- Defines \( x \) through \( z \)
  ```c
  *z=25;
  y=*z+1;
  ```
  - Uses \( x \) through \( z \)

C-use

- Uses of a variable that occur:
  - within an expression as part of an assignment statement;
  - in an output statement;
  - as a parameter within a function call; and
  - in subscript expressions
- "c" in c-use stands for computational.

**Examples**
- How many c-uses of \( x \)?
  ```c
  z=x+1;
  A[x-1]= B[2];
  foo(x'x)
  output(x);
  ```
p-use

- The occurrence of a variable
  - in an expression used as a condition in a branch statement such as an if and a while
- "p" in p-use stands for predicate.
- Examples
  
  ```
  if(x>0){output(x)};
  while(x>x){...};
  ```

- Possible confusion
  
  ```
  if(A[x+1]>0){output(x)};
  ```
  - The use of A is clearly a p-use.

What about x? c-use? p-use?

Global and Local Definitions and Uses

- Consider a basic block

  - Defines p uses y and z. (Local definition)
  - Defines x uses p. (Killed by the 2nd def)
  - Defines p uses x. (global definition)

  - Uses of y and z are global. Their definitions flow into this block from some other block.
  - Use of p is local.
  - Definition of x is global.

  - We are concerned only with the global definitions and uses.

Data-Flow Graph

- Data-flow graph is to capture the flow of definitions (defs) across basic blocks in a program.
  - Also known as def-use graph
  - Similar to a control flow graph of a program; nodes, edges, and paths in the CFG are preserved in the data flow graph.

- Derivation from CFG
  
  - Find basic blocks, computes defs, c-uses, and p-uses for each basic block.
  - Attach defs, c-use and p-use to each node in the graph.
  - Label each edge with the condition which when true causes the edge to be taken.

Derivation from CFG

<table>
<thead>
<tr>
<th>Node (in Block)</th>
<th>def</th>
<th>c-use</th>
<th>p-use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x, y, z)</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>(x)</td>
<td>(y)</td>
<td>(x)</td>
</tr>
<tr>
<td>3</td>
<td>(x)</td>
<td>(y)</td>
<td>(x)</td>
</tr>
<tr>
<td>4</td>
<td>(x)</td>
<td>(y)</td>
<td>(x)</td>
</tr>
<tr>
<td>5</td>
<td>[]</td>
<td>[]</td>
<td>[]</td>
</tr>
</tbody>
</table>
Def-Clear Path

- Any path starting from a node at which variable x is defined and ending at a node at which x is used, without redefining x anywhere else along the path.
- Path 2-5 is def-clear for variable z.
  - Defined at node 2 and used at node 5.
- Path 1-2-5 is NOT def-clear for variable z.
  - Defined at node 1 and used at node 5.

- Definition of z at node 2 is live at node 5 while z at node 1 is not live at node 5.

Def-Use Pairs

- Def of a variable at line l1 and its use at line l2 constitute a def-use pair. l1 and l2 can be the same.
- Definition and c-use (dcu)
  - dcu(d_i(x)) denotes the set of all nodes where d_i(x) is live and used.
- Definition and p-use (dpu)
  - dpu(d_i(x)) denotes the set of all edges (k, l) s.t. there is a def-clear path from node i to edge (k, l) and x is used at node k.
- A def-use pair (d_i(x), u_j(x)) is considered covered
  - when a def-clear path that includes nodes i to node j is executed.
  - If u_j(x) is a p-use then all edges of the kind (j, k) must also be taken during some executions.
Def-Use Chains

- A sequence of alternating definitions and uses of variables
- Also called k-dr interaction
  - k denotes the length of the chain
  - dr means definition and reference

- Examples
  - Chain (1,4) is obtained from:
    - \( d_1(z), u_4(z) \) -- a k-dr chain for k=2.
  - Chain (1,4,6) is obtained from:
    - \( d_1(z), u_4(z), d_4(z), d_6(z) \) -- k = 3
  - Chain (5,6,4) for variable y and z
    - \( d_5(z), u_6(z), d_6(y), u_4(y) \)

---

Minimal Def-Use Pair Set

- Def-use pairs are items to be covered during testing.
- Coverage of a def-use pair can imply coverage of another def-use pair.
- Analysis of the data flow graph can reveal a minimal set of def-use pairs whose coverage implies coverage of all def-use pairs.

Find minimal set.