Controlflow-based Coverage Criteria

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
http://www.utdallas.edu/~ewong

Speaker Biographical Sketch

- Professor & Director of International Outreach
  Department of Computer Science
  University of Texas at Dallas
- Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)
- Vice President, IEEE Reliability Society
- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)
- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project
  - Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses
- Founder & Steering Committee co-Chair for the SERE conference
  (IEEE International Conference on Software Security and Reliability)
  (http://paris.utdallas.edu/sere13)
Outline

- Block/Statement Coverage
- Decision Coverage
- Condition Coverage
- Multiple Condition Coverage

Statement and Block Coverage
**Declarations and Basic Blocks**

- Any program written in a procedural language consists of a sequence of statements. Some of these statements are *declarative*, such as the `#define` and `int` statements in C, while others are *executable*, such as the `assignment`, `if`, and `while` statements in C and Java.

- Recall that a *basic block* is a sequence of consecutive statements that has exactly one entry point and one exit point.
  - For any procedural language, adequacy with respect to the statement coverage and block coverage criteria are defined next.

- Notation: $(P, R)$ denotes program $P$ subject to requirement $R$.

---

**Statement Coverage**

- The statement coverage of $T$ with respect to $(P, R)$ is computed as $S_c / (S_e - S_u)$, where $S_c$ is the number of statements covered, $S_u$ is the number of *unreachable statements*, and $S_e$ is the total number of *executable statements* in the program, i.e., the size of the coverage domain.

- $T$ is considered *adequate* with respect to the statement coverage criterion if the statement coverage of $T$ with respect to $(P, R)$ is 1.
Block Coverage

- The block coverage of \( T \) with respect to \((P, R)\) is computed as \( B_c / (B_e - B_i) \), where \( B_c \) is the number of blocks covered, \( B_i \) is the number of unreachable blocks, and \( B_e \) is the total number of executable blocks in the program, i.e., the size of the block coverage domain.

- \( T \) is considered adequate with respect to the block coverage criterion if the statement coverage of \( T \) with respect to \((P, R)\) is 1.

---

Example: Statement Coverage

- Coverage domain: \( S_e = \{4, 5, 6, 7, 8, 9, 12, 13\} \) Let \( T_1 = \{t_1; x = -1, y = -1 >, t_2; x = 1, y = 1 >\} \)

- Statements covered:
  - \( t_1 \): 4, 5, 6, 7, 8 and 13
  - \( t_2 \): 4, 5, 6, 12, and 13

- \( S_c = 7, S_i = 1, S_e = 8 \). The statement coverage for \( T_1 \) is \( 7 / (8 - 1) = 1 \).
  
  Hence we conclude that \( T_1 \) is adequate for \((P, R)\) with respect to the statement coverage criterion. Note: 9 is unreachable.
**Example: Block Coverage (1)**

- Coverage domain: \( B_e = \{1, 2, 3, 4, 5\} \)
- Blocks covered:
  - \( t_1 \): Blocks 1, 2, 5
  - \( t_2, t_3 \): same coverage as of \( t_1 \).
- \( B_e = 5 \), \( B_i = 3 \), \( B_b = 1 \).
  - Block coverage for \( T_2 = 3 / (5 - 1) = 0.75 \).
  - Hence \( T_2 \) is **not adequate** for \((P, R)\) with respect to the block coverage criterion.

**Example: Block Coverage (2)**

- \( T_1 \) is adequate w.r.t. block coverage criterion. **Verify this statement!**
- Also, if test \( t_2 \) in \( T_1 \) is added to \( T_2 \), we obtain a test set adequate with respect to the block coverage criterion for the program under consideration.
  - **Verify this statement!**
Coverage Values

- The formulae given for computing various types of code coverage yield a coverage value between 0 and 1. However, while specifying a coverage value, one might instead use percentages. For example, a statement coverage of 0.65 is the same as 65% statement coverage.

Condition and Decision Coverage
**Conditions**

- Any expression that evaluates to *true* or *false* constitutes a *condition*. Such an expression is also known as a *predicate*.

- Given that A, B, and D are Boolean variables, and x and y are integers, A, x > y, A OR B, A AND (x < y), (A AND B) are sample conditions.

- Note that in programming language C, x and x + y are valid conditions, and the *constants 1 and 0* correspond to, respectively, *true and false*.

---

**Simple and Compound Conditions**

- A *simple condition* does not use any Boolean operators except for the *not* operator. It is made up of variables and *at most one* relational operator from the set {<, ≤, >, ≥, ==, ≠}.

- *Simple conditions* are also referred to as *atomic* or *elementary* conditions because they cannot be parsed any further into two or more conditions.

- A *compound condition* is made up of two or more simple conditions joined by one or more Boolean operators.
Conditions as Decisions

- Any condition can serve as a decision in an appropriate context within a program. Most high level languages provide `if`, `while`, and `switch` statements to serve as contexts for decisions.

Outcomes of a Decision

- A decision can have three possible outcomes: true, false, and undefined.

- In some cases the evaluation of a condition might fail in which case the corresponding decision’s outcome is undefined.
**Undefined Condition**

- The condition inside the if statement on line 6 will remain undefined because the loop at lines 2-4 will never end. Thus the decision on line 6 evaluates to undefined.

```c
1    bool foo(int a,a_parameter){
2        while (true) {   // An infinite loop.
3            a,a_parameter=0;
4        }
5    }                       // End of function foo().
```

**Coupled Conditions**

- How many simple conditions are there in the compound condition: \( D = (A \text{ AND } B) \text{ OR } (C \text{ AND } A) \)? The first occurrence of \( A \) is said to be coupled to its second occurrence.
- Does \( D \) contain three or four simple conditions? Both answers are correct depending on one's point of view. Indeed, there are three distinct conditions \( A \), \( B \), and \( C \). The answer is four when one is interested in the number of occurrences of simple conditions in a compound condition.
**Conditions within Assignments**

- Strictly speaking, a condition becomes a decision only when it is used in the appropriate context such as within an if statement.
- At line 4, $x < y$ does not constitute a decision and neither does $A \times B$.
  1. $A = x < y$; // A simple condition assigned to a Boolean variable $A$.
  2. $X = P$ or $Q$; // A compound condition assigned to a Boolean variable $x$
  3. $x = y + z \times s$; if$(x)$…// The condition will be true if $x = 1$ and false otherwise
  4. $A = x < y, x = A \times B$; // $A$ is used in a subsequent expression for $x$ but not as a decision

---

**Decision Coverage**

- A decision is considered covered if the flow of control has been diverted to all possible destinations that correspond to this decision, i.e., all outcomes of the decision have been taken.
- This implies that, for example, the expression in the if or a while statement has evaluated to true in some execution of the program under test and to false in the same or another execution.
Decision Coverage: Switch Statement

• Decision implied by the *switch* statement is considered covered if during one or more executions of the program under test the flow of control has been *diverted to all possible destinations*.

Decision Coverage: Example (1)

• Requirement:
  – The following code inputs an integer \( x \), and if \( x < 0 \), transforms it into a positive value before invoking \( \text{foo-1} \) to compute the output \( z \).
  – It is supposed to compute \( z \) using \( \text{foo-2} \) when \( x \geq 0 \).
  – It has a bug.

```c
1 begin
2 int X, Z;
3 input (x);
4 if(x<0)
5  X = -X;
6  z=foo-1(x);
7  output(z);
8 end
```

There should have been an *else* clause before this statement.
**Decision Coverage: Example (2)**

- Consider the test set \( T = \{ t_1; < x = -5 > \} \).
  - It is adequate with respect to statement and block coverage criteria, but does not reveal the bug.
- Another test set \( T' = \{ t_1; < x = -5 > \ t_2; < x = 3 > \} \) does reveal the bug. It covers the decision whereas \( T \) does not. **Check!**

This example illustrates how and why decision coverage might help in revealing a bug that is not revealed by a test set adequate with respect to statement and block coverage.

```
1 begin
2 int x, z;
3 input (x);
4 if(x<0)
5   x=x;
6   z=foo-1(x);
7 output(z);
8 end
```

Decision Coverage: Computation

- The decision coverage of \( T \) with respect to \((P, R)\) is computed as \( \frac{D_c}{D_e - D_i} \), where \( D_c \) is the number of decisions covered.
- \( D_i \) is the number of infeasible decisions, and \( D_e \) is the total number of decisions in the program, i.e., the size of the decision coverage domain.
- \( T \) is considered adequate with respect to the decision coverage criterion if the decision coverage of \( T \) with respect to \((P, R)\) is 1.
**Decision Coverage: Domain**

- The domain of decision coverage consists of *all decisions in the program under test.*

**Condition Coverage**

- A decision can be composed of a *simple condition* such as \(x < 0\), or of a *more complex condition*, such as \((x < 0 \text{ AND } y < 0) \text{ OR } (p \geq q)\).
- AND, OR, XOR are the *logical operators* that connect two or more simple conditions to form a *compound condition*.
- A *simple condition is considered covered if it evaluates to true and false in one or more executions* of the program in which it occurs.
- A compound condition is considered covered if each simple condition it is comprised of is also covered.
Decision and Condition Coverage (1)

- Decision coverage is concerned with the coverage of decisions regardless of whether or not a decision corresponds to a simple or a compound condition. Thus in the statement

```java
if (x < 0 and y < 0) {
    z = foo(x, y)
}
```

- There is only one decision that leads control to line 2 if the compound condition inside the `if` evaluates to true.
- However, a compound condition might evaluate to true or false in one of several ways.

Decision and Condition Coverage (2)

- Referring to the following code

```java
if (x < 0 and y < 0) {
    z = foo(x, y)
}
```

- The condition at line 1 evaluates to false when \( x \geq 0 \) regardless of the value of \( y \).
- Another condition, such as \((x < 0 \text{ OR } y < 0)\), evaluates to true regardless of the value of \( y \), when \( x < 0 \).
- With this evaluation characteristic in view, compilers often generate code that uses short circuit evaluation of compound conditions.
**Decision and Condition Coverage (3)**

- Here is a possible translation:

```java
1 if (x < 0 and y < 0) {
2   z = foo(x, y);
1 if (x < 0)
2   if (y < 0)
3   z = foo(x, y);
```

- We now see two decisions, one corresponding to each simple condition in the `if` statement.

---

**Condition Coverage**

- The *condition coverage* of T with respect to (P, R) is computed as $C_c / (C_e - C_i)$, where
  - $C_c$ is the number of simple conditions covered,
  - $C_i$ is the number of infeasible simple conditions, and
  - $C_e$ is the total number of simple conditions in the program.

- T is considered adequate with respect to the condition coverage criterion if the condition coverage of T with respect to (P, R) is 1.

- An alternate formula where each simple condition contributes 2, 1, or 0 to $C_c$, depending on whether it is covered, partially covered, or not covered, respectively, is:

$$\frac{C_c}{2 \times (C_e - C_i)}$$
Condition Coverage: Example (1)

- Partial specifications for computing $z$

<table>
<thead>
<tr>
<th>$x &lt; 0$</th>
<th>$y &lt; 0$</th>
<th>Output ($z$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>foo1($x,y$)</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>foo2($x,y$)</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>foo2($x,y$)</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>foo1($x,y$)</td>
</tr>
</tbody>
</table>

This program has a bug based on the specification.

Condition Coverage: Example (2)

- Consider the test set $T = \{ t_1 : x = -3, y = -2 > t_2 : x = -4, y = -2 > \}$
- Check that $T$ is adequate with respect to the statement, block, and decision coverage criteria and the program behaves correctly against $t_1$ and $t_2$.

$C_c = 1$, $C_e = 2$, $C_i = 0$. Hence, condition coverage for $T = 0.5$.

```c
1 begin
2 int x, y, z;
3 input (x, y);
4 if(x<0 and y<0)
5   z=foo1(x,y);
6 else
7   z=foo2(x,y);
8 output(z);
9 end
```
**Condition Coverage: Example**

- Add the following test case to $T$: $t_j: x=3, y=4$
- Check that the enhanced test set $T$ is adequate with respect to the condition coverage criterion and possibly reveals a bug in the program.
  - The programs shows $z = \text{foo2}(x, y)$
  - But the specifications says $z = \text{foo1}(x, y)$
- Under what conditions will the bug be revealed by $t_j$?

```plaintext
1 begin
2 int x, y, z;
3 input(x, y);
4 if(x<0 and y<0)
5   z=foo1(x,y);
6 else
7   z=foo2(x,y);
8 output(z);
9 end
```

**Condition/Decision Coverage**

- When a decision is composed of a compound condition, decision coverage does not imply that each simple condition within a compound condition has taken both values true and false.
- Condition coverage ensures that each component simple condition within a condition has taken both values true and false.
- Question: Does the condition coverage require each decision to take all its outcomes?
**Condition/Decision Coverage: Example**

- Consider the following program and two test sets.

```plaintext
begin
int x, y, z;
input (x, y);
if(x<0 or y<0)
z=boo-1(x,y);
else
z=boo-2(x,y);
output(z);
end
```

Test Sets:

- $T_1 = \{ t_1: < x = -3 \ y = -2 > \ \ t_2: < x = 4 \ y = -2 > \}$
- $T_2 = \{ t_1: < x = -3 \ y = 2 > \ \ t_2: < x = 4 \ y = -2 > \}$

**In-class exercise:**
- Is $T_1$ is adequate with respect to decision coverage?
- Is $T_1$ is adequate with respect to condition coverage?
- How about $T_2$?

---

**Condition/Decision Coverage: Definition**

- The condition/decision coverage of $T$ with respect to $(P, R)$ is computed as $(C_c + D_c) / ((C_e - C_i) + (D_e - D_i))$, where
  - $C_c$ is the number of simple conditions covered
  - $D_c$ is the number of decisions covered,
  - $C_e$ and $D_e$ are the number of simple conditions and decisions respectively
  - $C_i$ and $D_i$ are the number of infeasible simple conditions and decisions, respectively.
**Condition/Decision Coverage: Example**

- **In-class exercise:** Is $T$ adequate with respect to the condition/decision coverage criterion?

```plaintext
begin  T = \{ t_1 : < z = -3 \quad y = -2 > \} 
  t_2 : < z = 4 \quad y = 2 > \} 
int x, y, z;
input(x, y);
if(x<0 or y<0)
z=foo-1(x,y);
else
z=foo-2(x,y);
output(z);
end
```

---

**Multiple Condition Coverage**
**Multiple Condition Coverage**

- Consider a compound condition with two or more simple conditions. Using condition coverage on some compound condition C implies that each simple condition within C needs to be evaluated to true and false.
- However, does it imply that all combinations of the values of the individual simple conditions in C have been exercised?

---

**Multiple Condition Coverage/Simple Condition Coverage**

- Multiple condition coverage versus simple condition coverage is similar to uni-dimensional equivalence class partitioning versus multi-dimensional equivalence partitioning.
  ➔ considered separately versus considered simultaneously
**Multiple Condition Coverage: Example**

- Consider \( D = (A < B) \) OR \((A > C)\) composed of *two simple conditions* \( A < B \) and \( A > C \). The four possible combinations of the outcomes of these two simple conditions are enumerated in the table.

- Check: Is \( T \) 100% w.r.t. the decision coverage?
- Check: Is \( T \) 100% w.r.t. the condition coverage?
- Check: Does \( T \) cover all four combinations?
- Check: Does \( T' \) cover all four combinations?

<table>
<thead>
<tr>
<th></th>
<th>( A &lt; B )</th>
<th>( A &gt; C )</th>
<th>( D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

\[
T = \begin{cases} 
  t_1 : & A = 2 \quad B = 3 \quad C = 1 > \\
  t_2 : & A = 2 \quad B = 1 \quad C = 3 > 
\end{cases}
\]

\[
T' = \begin{cases} 
  t_1 : & A = 2 \quad B = 3 \quad C = 1 > \\
  t_2 : & A = 2 \quad B = 1 \quad C = 3 > \\
  t_3 : & A = 2 \quad B = 3 \quad C = 5 > \\
  t_4 : & A = 2 \quad B = 1 \quad C = 5 > 
\end{cases}
\]

**Multiple Condition Coverage: Definition** (1)

- Suppose that the program under test contains a total of \( n \) decisions. Assume also that each decision contains \( k_1, k_2, \ldots, k_n \) simple conditions. Each decision has several combinations of values of its constituent simple conditions.

- For example, decision \( i \) will have a total of \( 2^{k_i} \) combinations. Thus the total number of combinations to be covered is

\[
\sum_{i=1}^{n} 2^{k_i}
\]
Multiple Condition Coverage: Definition (2)

The multiple condition coverage of \( T \) with respect to \((P, R)\) is computed as \( C_c / (C_e - C_i) \), where:

- \( C_c \) is the number of combinations covered,
- \( C_i \) is the number of infeasible simple combinations, and
- \( C_e \) is the total number of combinations in the program.

\( T \) is considered adequate with respect to the multiple condition coverage criterion if the condition coverage of \( T \) with respect to \((P, R)\) is 1.

Multiple Condition Coverage: Example (1)

Consider the following program with specifications in the table.

```
begin
  input A, B, C, S=0;
  if(A < B and A > C) S=11(A, B, C);
  if(A < B and A < C) S=12(A, B, C);
  if(A > B and A < C) S=14(A, B, C);
  output(S);
end
```

<table>
<thead>
<tr>
<th>( A &lt; B )</th>
<th>( A &gt; C )</th>
<th>( S )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>( f1(P, Q, R) )</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>( f2(P, Q, R) )</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>( f3(P, Q, R) )</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>( f4(P, Q, R) )</td>
</tr>
</tbody>
</table>

There is an obvious bug in the program: computation of \( S \) for one of the four combinations, line 3 in the table, has been left out.
Multiple Condition Coverage: Example (2)

- Is $T$ adequate w.r.t. decision coverage?
- Multiple condition coverage?
- Does it reveal the bug?

```plaintext
begin
int A, B, C, S=0;
input(A, B, C);
if(A<B and A>C) S=f1(A, B, C);
if(A<B and A<=C) S=f2(A, B, C);
if(A>=B and A<=C) S=f4(A, B, C);
output(S);
end
```

$$
T = \begin{cases} 
  t_1 : & A = 2, B = 3, C = 1 \\
  t_2 : & A = 2, B = 1, C = 3 
\end{cases}
$$

Multiple Condition Coverage: Example (3)

- Is $T'$ 100% with respect to the decision coverage?
- Does $T'$ reveal the bug?

```plaintext
begin
int A, B, C, S=0;
input(A, B, C);
if(A<B and A>C) S=f1(A, B, C);
if(A<B and A<=C) S=f2(A, B, C);
if(A>=B and A<=C) S=f4(A, B, C);
output(S);
end
```

$$
T' = \begin{cases} 
  t_1 : & A = 2, B = 3, C = 1 \\
  t_2 : & A = 2, B = 1, C = 3 \\
  t_3 : & A = 2, B = 3, C = 5 
\end{cases}
$$
Multiple Condition Coverage: Example (4)

- In-class exercise:
  - Is \( T \) 100% w.r.t. simple condition coverage?
  - Is \( T \) 100% w.r.t. multiple condition coverage?

- Now add a test to \( T \) to cover the uncovered combinations.
  - Does your test reveal the bug?
  - If yes, then under what conditions?