Test Adequacy Assessment using Control Flow and Data Flow

Lecture 14

Dr. Ilchul Yoon (icyoon@sunykorea.ac.kr)

Sources: Foundations of Software Testing (textbook)

Conditions

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

• Examples
  • Given that A and B are Boolean variables, and x and y are integers, A OR B, A AND (x-y), (A AND B), are all 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
  • No use of any Boolean operators except for the not operator.
  • Made up of variables and at most one relational operator from the set {<, <=, >=, ==, !=}.
  • Referred to as atomic or elementary conditions

• A compound condition
  • Made up of two or more simple conditions joined by one or more Boolean operators.

Conditions as Decisions

• Condition can serve as a decision in an appropriate context within a program.
• Example: if, while, and switch statements to serve as contexts for decisions.

if (A)
  task if A is true;
else
  task if A is false;

while(A)
  task while A is true;
else
  task if A is false;

switch(x)
  task for x=a1
default task
  task for x=a2
default task
  w
  w
  w
  w
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.

```c
bool foo(int a_parameter)
{
    while (true) { // An infinite loop.
        a_parameter = 0;
    } // End of function foo().
    ...
    if (x < y and foo(y)) { // foo() does not terminate.
        compute(x, y);
    ...
    ```

Coupled Conditions

- Q: How many simple conditions are there in the compound condition?
- Cond = (A AND B) OR (C AND A)? The first occurrence of A is said to be coupled to its second occurrence.
- Q: Does Cond contain three or four simple conditions?

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.

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. $x = y + z * 8$; if $(x)$...; // Condition true if $x = 1$, false otherwise.
4. $A = x = y; x = A * 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.
  - For if or a while statement, expressions must be evaluated to true and false by (multiple) execution(s).
  - For the switch statement, the flow of control must be diverted to all possible destinations in a set of executions.

- **Covering a decision within a program might reveal an error that is not revealed by covering all statements and all blocks.**

---

**Decision Coverage: Example**

- Requirements
  - Receive input (an integer \(x\)), and if \(x\) is negative, transforms it into a positive value before invoking `foo-1` to compute the output \(z\).
  - If the input \(x \geq 0\), the program is supposed to compute \(z\) using `foo-2`.

- Obviously, the program has errors.

There should have been an `else` clause before this statement.

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

---

**Decision Coverage: Example (cont.)**

- Assume a test set
  - \(T = \{t1:<x=5>\}\).
  - It is adequate with respect to statement and block coverage criteria, but does not reveal the error.

- Another test set \(T' = \{t1:<x=5>, t2:<x=3>\}\) does reveal the error. It covers the decision whereas \(T\) does not.

There should have been an `else` clause before this statement.

---

**Decision Coverage: Definition**

- Coverage domain
  - \(D_c\) is the set of decisions in the program

- Decision coverage of \(T\) with respect to \((P, R)\)
  - \(T = \frac{|D_c|}{|D_i|}\)
  - \(D_c\) is the set of decisions covered
  - \(D_i\) is the set of infeasible decisions

- \(T\) is considered **adequate with respect to the decision coverage criterion** if the decision coverage of \(T\) with respect to \((P, R)\) is 1.
Decision vs. Condition Coverage

- Decision coverage does not care the outcome of each condition.

- Example

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

  - There exist multiple ways in which the decision in line 1 is evaluated to true or false!
  - The condition evaluates to `false` when x≥0, regardless of the value of y.
  - Another condition, such as x<0 OR y<0, evaluates to `true` regardless of the value of y, when x<0.

Condition Coverage

- A decision can be a simple condition or a compound condition.
  - x<0
  - ((x<0 AND y<0) OR (p≥q)).

- In condition coverage, we consider that
  - A simple condition is covered if it evaluates to `true` and `false` in 1* executions of the program.
  - A compound condition is considered covered if each simple condition it is comprised of is covered.

Coverage domain

- $C_c$ is the set of simple conditions in the program

Condition coverage of $T$ with respect to $(P,R)$

- $T = \frac{|C_c|}{|C_c| - |C_i|}$
- $C_c$ is the set of simple conditions covered
- $C_i$ is the set of infeasible simple conditions

- $T$ is considered adequate with respect to the condition coverage criterion if the condition coverage of $T$ with respect to $(P,R)$ is 1.
Condition Coverage: Alternate Formula

- An alternate formula
  
  \[ T = \frac{|C|}{2 \times (|C_c| - |C|)} \]  
  
  each simple condition contributes 2, 1, or 0 to \( C_c \) depending on whether it is covered, partially covered, or not covered, respectively.

- For example, if \( x < y \) evaluates to true but never to false, then it is considered partially covered, and contribute a 1 to \( C_c \).

Condition Coverage: Example

- Consider a test set
  
  \[ T = \{ t_1; < x = -3, y = -2 >; t_2; < x = -4, y = 2 > \} \]

- \( T \) is adequate with respect to the statement, block, and decision coverage criteria and the program behaves correctly against \( t_1 \) and \( t_2 \).
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. However, condition coverage does not require each decision to have taken both outcomes.

Condition/decision coverage is also known as branch condition coverage.

Condition/Decision Coverage: Definition

- The condition/decision coverage of $T$ with respect to $(P, R)$ is computed as
  \[ T = \frac{|C_c| + |D_c|}{(|C_c| - |I_c|) + (|D_c| - |I_d|)} \]
  where
  - $C_c$ is the set of simple conditions covered
  - $D_c$ is the set of decisions covered
  - $C_c$ and $D_c$ are the set of simple conditions and decisions respectively
  - $C_i$ and $D_i$ are the set of infeasible simple conditions and decisions, respectively
  - $T$ is considered adequate with respect to the multiple condition coverage criterion if the condition coverage of $T$ with respect to $(P, R)$ is 1.

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=foo(1,x);
elserun(x):
z=foo(2,x);
output(2);
end
```

In class exercise: Confirm that $T_1$ is adequate with respect to decision coverage but not condition coverage.
In class exercise: Confirm that $T_2$ is adequate with respect to condition coverage but not decision coverage.

Condition/Decision Coverage: Example

- In class exercise: Check that the following test set is adequate with respect to the condition/decision coverage criterion.

```plaintext
T = \{ t_1 : < x = -3, y = -2 >, t_2 : < x = 4, y = 2 > \}
```

adequate with respect to decision coverage

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

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

adequate with respect to 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 $C$ has been 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 is known as branch condition combination coverage.

Multiple Condition Coverage: Example

- Consider a condition $D=(A<B) \text{ OR } (A>C)$
  - Consists of 2 simple conditions $A < B$ and $A > C$.
  - There are 4 possible combinations of the outcomes of these two simple conditions.

<table>
<thead>
<tr>
<th>$A &lt; B$</th>
<th>$A &gt; C$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Consider a test suite $T$

$T = \{ t_1: \langle A=2 \ B=3 \ C=1 \rangle \}
\{ t_2: \langle A=1 \ B=1 \ C=3 \rangle \}
$

Check: Does $T$ cover all four combinations?

$T'' = \{ t_1: \langle A=2 \ B=3 \ C=1 \rangle \}
\{ t_2: \langle A=2 \ B=1 \ C=3 \rangle \}
\{ t_3: \langle A=2 \ B=3 \ C=5 \rangle \}
\{ t_4: \langle A=2 \ B=1 \ C=1 \rangle \}$

Multiple Condition Coverage: Definition

- Assumption
  - Program under test contains a total of $n$ decisions.
  - Each decision contains $k_1, k_2, ..., k_n$ simple conditions.
  - Each decision has several combinations of values of its constituent simple conditions.

- The total number of combinations to be covered is:

$$\sum_{i=1}^{n} 2^{k_i}$$
Multiple Condition Coverage: Definition

- Coverage domain
  - $C_d$ is the set of combinations in the program $\sum_{i=1}^{n} 2^{i}$

- Condition coverage of $T$ with respect to $(P, R)$
  - $T = \frac{|C_c|}{|C_d| - |C_i|}$
  - $C_c$ is the set of combinations covered
  - $C_i$ is the set of infeasible combinations

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

Multiple Condition Coverage: Example

Consider the following program with specifications in the table.

<table>
<thead>
<tr>
<th>Requirements for computing $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A &lt; B$</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>false</td>
</tr>
<tr>
<td>false</td>
</tr>
</tbody>
</table>

There is an obvious error in the program.

- Is $T$ adequate with respect to
  - Decision coverage?
  - Condition coverage?
  - Multiple condition coverage? Does it reveal the error?

MCC: Example (cont.)

To improve decision coverage, add $t_3$ to $T$

$T' = \{ t_1 : < A = 2, B = 3, C = 1 > \}
\{ t_2 : < A = 2, B = 1, C = 3 > \}
\{ t_3 : < A = 2, B = 3, C = 5 > \}$

Does $T'$ reveal the error? ❌

MCC: Example (cont.)

In class exercise: Construct a table showing the condition combinations covered by $T'$.

- Some combinations of simple conditions remain uncovered.

$T' = \{ t_1 : < A = 2, B = 3, C = 1 > \}
\{ t_2 : < A = 2, B = 1, C = 3 > \}
\{ t_3 : < A = 2, B = 3, C = 5 > \}$

Now add a test to $T'$ to cover the uncovered combinations.

- Does your test reveal the error? If yes, then under what conditions?

$T' = \{ t_1 : < A = 2, B = 3, C = 1 > \}
\{ t_2 : < A = 2, B = 1, C = 3 > \}
\{ t_3 : < A = 2, B = 3, C = 5 > \}$
Linear Code Sequence and Jump (LCSAJ)

- Consider the execution of sequential programs as a chain of (block, jump) pairs, where
  - Block: a sequence of statements, executed one after the other
  - Jump: a pointer to the next such pair.
- A Linear Code Sequence and Jump (LCSAJ)
  - A program unit comprised of a textual code sequence that terminates in a jump to the beginning of another code sequence and jump.
  - Represent as a triple \((X, Y, Z)\) where
    - \(X\) and \(Y\) are, respectively, locations of the first and the last statements
    - \(Z\) is the location to which the statement at \(Y\) jumps.

LCSAJ: Example1

- Consider this program.

<table>
<thead>
<tr>
<th></th>
<th>begin</th>
<th>int x, y, p;</th>
<th>input (x, y);</th>
<th>if(x&lt;0)</th>
<th>p=g(y);</th>
<th>else</th>
<th>p=g(y&quot;y);</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>begin</td>
<td>int x, y, p;</td>
<td>input (x, y);</td>
<td>if(x&lt;0)</td>
<td>p=g(y);</td>
<td>else</td>
<td>p=g(y&quot;y);</td>
<td>end</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The last statement in an LCSAJ \((X, Y, Z)\) is a jump and \(Z\) may be program exit. When control arrives at statement \(X\), follows through to statement \(Y\), and then jumps to statement \(Z\), we say that the LCSAJ \((X, Y, Z)\) is traversed or covered or exercised.
Consider a program and test suite $T$.

$$T = \{ t_1 : < x = -5 \ y = 2 > \} \cup \{ t_2 : < x = 9 \ y = 2 > \}$$

$t_1$ covers $(1, 6, \text{exit})$, $t_2$ covers $(1, 4, 7)$ and $(7, 8, \text{exit})$.

$T$ covers all three LCSAJs.

---

Consider a program.

```java
begin
int x, y, p;
input (x, y);
if (x < 0)
y = g(y); else y = g(y*y);
end
```

Consider a test set. Does this set cover all LCSAJs?

$$T = \{ t_1 : < x = 5 \ y = 0 > \} \cup \{ t_2 : < x = 5 \ y = 2 > \}$$

---

LCSAJ: Definition

- The LCSAJ coverage of a test set $T$ with respect to $(P, R)$ is computed as:

  \[
  \frac{\text{Number of LCSAJs exercised}}{\text{Total number of feasible LCSAJs}}
  \]

- $T$ is considered adequate with respect to the LCSAJ coverage criterion if the LCSAJ coverage of $T$ with respect to $(P, R)$ is.
Problem of MCC

- Multiple condition coverage can be very expensive.
  - Assume that there are many embedded simple conditions.
  - When a compound condition $C$ contains $n$ simple conditions, the maximum number of tests required to cover $C$ is $2^n$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Minimum tests</th>
<th>Time to execute all tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2 hrs</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16 hrs</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>256 hrs</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>65.5 seconds</td>
</tr>
<tr>
<td>32</td>
<td>4294967296</td>
<td>49.5 days</td>
</tr>
</tbody>
</table>

MC/DC: Brute-Force Way

- Consider a condition $C = (C_1$ and $C_2)$ or $C_3$
  - Fix 2 variables and varies the value of the remaining one
  - Redundant test cases

MC/DC: Tests for Compound Condition

- Required tests for a compound condition with 2 simple conditions

MC/DC coverage requires that

- Every compound condition in a program must be tested by demonstrating that each simple condition within the compound condition has an independent effect on its outcome.

Thus MC/DC coverage is a weaker criterion than the multiple condition coverage criterion.
## MC/DC: Tests for Compound Condition

- **Required tests for a compound condition with 3 simple conditions**

<table>
<thead>
<tr>
<th>Test</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

**MC/DC-adequate test set for $C = \{t_1, t_2, t_3, t_4\}$**

Condition: $C_2 = (C_1 \lor C_3)$

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

**MC/DC-adequate test set for $C = \{t_1, t_2, t_3, t_4\}$**

Condition: $C_3 = (C_1 \land C_2)$

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

**MC/DC-adequate test set for $C = \{t_1, t_2, t_3, t_4\}$**

<table>
<thead>
<tr>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

## MC/DC: Procedure to Generate Tests for Compound Conditions

- Let $C = C_1 \land C_2 \land C_3$. Generate test for $C$ by extending simpler compound condition.
- Create a table with five columns and four rows.
- Label the columns as Test, $C_1$, $C_2$, $C_3$, and $C$, from left to right.
- The column labeled Test contains rows labeled by test case numbers $t_1$ through $t_4$. The remaining entries are empty.

<table>
<thead>
<tr>
<th>Test</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

## MC/DC: Procedure to Generate Tests for Compound Conditions (cont.)

- Copy all entries in columns $C_1$, $C_2$, and $C$ from the table for simple conditions into columns $C_2$, $C_3$, and $C$ of the empty table.

<table>
<thead>
<tr>
<th>Test</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

## MC/DC: Procedure to Generate Tests for Compound Conditions (cont.)

- Fill the first three rows in the column marked $C_1$ with true and the last row with false.

<table>
<thead>
<tr>
<th>Test</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
<th>$C$</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
MC/DC: Procedure to Generate Tests for Compound Conditions (cont.)

- Fill the last row under columns labeled C2, C3, and C with true, true, and false, respectively.

<table>
<thead>
<tr>
<th>Test</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>Tests t1 and t2 cover C3.</td>
</tr>
<tr>
<td>t2</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>Tests t1 and t2 cover C2.</td>
</tr>
<tr>
<td>t3</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>Tests t1 and t3 cover C2.</td>
</tr>
<tr>
<td>t4</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>Tests t2 and t3 cover C1.</td>
</tr>
</tbody>
</table>

We now have a table containing MC/DC adequate tests for $C=(C1 \land C2 \land C3)$ derived from tests for $C=(C1 \land C2)$.

MC/DC: Analysis

- The first three requirements above correspond to the block, condition, and decision coverage, respectively.
- The 4th requirement is for the MC coverage.
  - i.e., the MC/DC coverage criterion is a mix of four coverage criteria based on the flow of control.
  - Conditions that are not part of a decision, such as the one in the following statement $A=(p<q) \lor (x>y)$ are also included in the set of conditions to be covered.
  - For a coupled condition such as $(A \land B) \lor (C \land A)$, poses a problem, only demonstrate the independent effect of any one occurrence of the coupled condition

MC/DC: Adequacy

- Coverage domain (for MC coverage)
  - Let $C_1, C_2, ..., C_N$ be the conditions in $P$.
  - $n_i$ denote the number of simple conditions in $C_i$
  - $e_i$ is the number of simple conditions shown to have independent affect on the outcome of $C_i$
  - $f_i$ the number of infeasible simple conditions in $C_i$
  - $M_C = \frac{\sum_{i=1}^{N} e_i}{\sum_{i=1}^{N} (ni-f_i)}$
  - $T$ is considered adequate with respect to the MC coverage if $MC_C$ of $T$ is 1.