CS 4311

LECTURE 11
WHITE-BOX TESTING

Outline

- Program Representation
  - Control Flow Graphs (CFG)
- Coverage Criteria
  - Statement
  - Branch
  - Condition
  - Path
  - Def-Use
Thursday’s Riddle

THREE LOGICIANS WALK INTO A BAR...
DOES EVERYONE WANT BEER?
I DON'T KNOW.
I DON'T KNOW.
YES!

White-Box Testing

Function is understood only in terms of its implementation.
Program representation: Control flow graphs

A basic block in program P is a sequence of consecutive statements with a single entry and a single exit point.
- Block has unique entry and exit points.
- Control always enters a basic block at its entry point and exits from its exit point.
- There is no possibility of exit or a halt at any point inside the basic block except at its exit point.
- The entry and exit points of a basic block coincide when the block contains only one statement.
Reverse Engineering: What does this code do?

Example: Computing \( x \) raised to \( y \)

1. \textbf{begin}
2. \hspace{1em} \textbf{int} \ x, \ y, \ power; \hspace{1em}
3. \hspace{1em} \textbf{float} \ z; \hspace{1em}
4. \hspace{1em} \textbf{input} \ (x, \ y); \hspace{1em}
5. \hspace{1em} \textbf{if} \ (y < 0) \hspace{1em}
6. \hspace{2em} \textbf{power} = y; \hspace{1em}
7. \hspace{2em} \textbf{else} \hspace{1em}
8. \hspace{3em} \textbf{power} = y; \hspace{1em}
9. \hspace{1em} \textbf{z} = 1; \hspace{1em}
10. \hspace{1em} \textbf{while} \ (\text{power} != 0) \{ \hspace{1em}
11. \hspace{2em} \textbf{z} = \textbf{z} \times \text{power}; \hspace{1em}
12. \hspace{2em} \textbf{power} = \textbf{power} - 1; \hspace{1em}
13. \hspace{1em} \} \hspace{1em}
14. \hspace{1em} \textbf{if} \ (y < 0) \hspace{1em}
15. \hspace{2em} \textbf{z} = 1 / \textbf{z}; \hspace{1em}
16. \hspace{1em} \textbf{output} (\textbf{z}); \hspace{1em}
17. \hspace{1em} \textbf{end}

Basic blocks: Example (contd.)

<table>
<thead>
<tr>
<th>Block</th>
<th>Lines</th>
<th>Entry point</th>
<th>Exit point</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 3, 4, 5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11, 12</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>
A control flow graph (CFG) $G$ is defined as a finite set $N$ of nodes and a finite set $E$ of edges. An edge $(i, j)$ in $E$ connects two nodes $n_i$ and $n_j$ in $N$. We often write $G = (N, E)$ to denote a flow graph $G$ with nodes given by $N$ and edges by $E$. In a flow graph of a program, each basic block becomes a node and edges are used to indicate the flow of control between blocks. Blocks and nodes are labeled such that block $b_i$ corresponds to node $n_i$. An edge $(i, j)$ connecting basic blocks $b_i$ and $b_j$ implies that control can go from block $b_i$ to block $b_j$. We also assume that there is a node labeled Start in $N$ that has no incoming edge, and another node labeled End, also in $N$, that has no outgoing edge.
Control Flow Graphs

- Programs are made of three kinds of statements:
  - Sequence (i.e., series of statements)
  - Condition (i.e., if statement)
  - Iteration (i.e., while, for, repeat statements)

- Control flow graph: visual representation of flow of control.
  - Node represents a basic block (sequence of statements with single entry and last statement is transfer of control)
  - Edge represents transfer of control from one node to another

Control Flow Graph (CFG)

- Sequence
- If-then-else
- If-then
- Iterative
Example 1: CFG

Example 2

Quiz 1: Develop a CFG (5 minutes)
Quiz 2: Develop a CFG (3 Minutes)

Example 3
1. read (a,b);
2. if (a ≠ 0 && b ≠ 0) then {
   3.   c ← a + b;
   4.   if c > 10 then
   5.      c ← max
   6.   else c ← min }
7. else print 'Done'

Quiz 3: Develop a CFG (2 Minutes)

Example 4
1. read (a,b);
2. if (a ≠ 0 || b ≠ 0) then {
   3.   c ← a + b
   4.   while (c < 100)
   5.      c ← a + b;
6.   c ← a * b
Statement (Line) Coverage:

- Develop test cases so that every statement is executed at least once

Example 4
1. read (a,b);
2. if (a ≠ 0 || b ≠ 0) then {
   3.   c ← a + b
   4.   while (c < 100)
   5.     c ← a + b;
   6.   c ← a * b

Quiz 4: Develop Test Cases Using Statement Coverage (5 Minutes)

- Develop test cases so that every statement is executed at least once

Example 4
1. read (a,b);
2. if (a ≠ 0 || b ≠ 0) then {
   3.   c ← a + b
   4.   while (c < 100)
   5.     c ← a + b;
   6.   c ← a * b
Quiz 4: Develop Test Cases Using Statement Coverage (5 Minutes)

- Develop test cases so that every statement is executed at least once

**Example 4**
1. `read (a,b);`
2. `if (a \neq 0 || b \neq 0) then {`
3. `c \leftarrow a + b`
4. `while (c < 100)`
5. `c \leftarrow a + b;`
6. `c \leftarrow a * b`

**Test Cases:**
- Test1: `a = 50, b = 49 (1,2,3,4,5,4,5,4,5,...)`
- Test2: `a = 0, b = 0 (1,2,4,6)`
Statement Coverage (Problems)

- Software developers and testers commonly use line coverage because of its simplicity and availability in object code instrumentation technology.
- Of all the structural coverage criteria, line coverage is the weakest, indicating the fewest number of test cases.
- Bugs can easily occur in the cases that line coverage cannot see.
- The most significant shortcoming of line coverage is that it fails to measure whether you test simple if statements with a false decision outcome. Experts generally recommend to only use line coverage if nothing else is available. Any other measure is better.

Branch Coverage

- Each edge of a program’s CFG is traversed at least once in some test.
- Independent paths:
  - 1, 2, 3, 9
  - 1, 2, 3, 4, 5, 6, 7, 8, 3, ..., 9
  - 1, 2, 3, 4, 5, 7, 8, 3, ..., 9

Example 1
Quiz 5: Develop MINIMAL Set of Test Cases Using Branch Coverage (7 Minutes)

1. read (result);
2. read (x,k)
3. while result < 0 {
4. ptr ← false
5. if x > k then
6. ptr ← true
7. x ← x + 1
8. result ← result + 1 }
9. print result

Possible test set: dc stands for don't care
{<result = 5>, x = dc, k = dc}
Quiz 5: Develop MINIMAL Set of Test Cases Using Branch Coverage (7 Minutes)

1. read (result);
2. read (x,k)
3. while result < 0 {
4. ptr ← false
5. if x > k then
6.   ptr ← true
7.   x ← x + 1
8.   result ← result + 1 }
9. print result

Possible test set: dc stands for don't care
{<
<result = -1, x = 5, k = 3>,

<result = -1, x = 10, k = 15>}

Quiz 5: Develop MINIMAL Set of Test Cases Using Branch Coverage (7 Minutes)

1. read (result);
2. read (x,k)
3. while result < 0 {
4. ptr ← false
5. if x > k then
6.   ptr ← true
7.   x ← x + 1
8.   result ← result + 1 }
9. print result

Possible test set: dc stands for don't care
{<
<result = -1, x = 5, k = 3>,

<result = -1, x = 10, k = 15>
Quiz 5: Develop MINIMAL Set of Test Cases Using Branch Coverage (7 Minutes)

Possible test set: dc stands for don't care

Minimal test set
{<result = -2, x = 5, k = 5>}

Condition Coverage

- Each edge is traversed and all possible values of constituents of compound condition are executed at least once
- Test each way a condition can be true or false
- Requires rewriting the CFG to identify new paths
- Basis set yields minimal test set
AND Condition

1. read (a, b);
2. if (a == 0 && b == 0) then {
3.     c ← a + b
4. else c ← a * b}

Basis set:
1, 2A, 2B, 3, J
1, 2A, 2B, 4, J
1, 2A, 4, J

OR Condition

1. read (a, b);
2. if (a == 0 || b == 0) then {
3.     c ← a + b;
4.     while (c < 100)
5.         c ← a + b;
}

Basis set:
1, 2A, 3, 4, 5, 4 ... J
1, 2A, 3, 4, J
1, 2A, 2B, 3, 4, 5, 4, ... J
1, 2A, 2B, J
Quiz 6: Develop Minimal Set of Test Cases Using Condition Coverage (5 Minutes)

1. read (a,b);
2. if (a == 0 || b == 0) then {
3.   c ← a + b;
4.   while (c < 100)
5.   c ← a + b;
}

Test1: a = 0, b = 99
Test2: a = 0, b = 100
Test3: a = 99, b = 0
Test4: a = 1, b = 1

Cyclomatic Complexity

- Software metric that provides a quantitative measure of the logical complexity of a program.
- Defines the number of independent paths in the basis set of a program
  - Independent path introduces at least one new set of process statements or a new condition.
  - Basis set is set of paths such that if test cases force execution of these set of paths, then every statement in the path and every condition will be executed (both true and false). Question is; which CFG are we talking about?
- Provides the upper bound for the number of tests that must be conducted to ensure that all statements been have executed at least once
  - $V(G) = E - N + 2$
  - $V(G) = B + 1$

What are E, N, and B?
Example: What is the Cyclomatic Complexity?

\[ V(G) = 11 - 9 + 2 = 4 \]

Independent paths:
- 1-11
- 1-2-3-4-5-10-1-11
- 1-2-3-6-8-9-10-1...-11
- 1-2-3-6-7-9-10-1...-11

Path Coverage-1

- Every distinct path through code is executed at least once
- Basis set does not yield minimal test set
- Example
  1. read (x)
  2. read (z)
  3. if x = 0 then begin
  4. \( y \leftarrow x \ast z \); 
  5. \( x \leftarrow z \) end 
  6. else print ‘Invalid’
  7. if y > 1 then 
  8. print y 
  9. else print ‘Invalid’
- Test Paths:
  - 1, 2, 3, 4, 5, J1, 7, 8, J2
  - 1, 2, 3, 4, 5, J1, 7, 9, J2
  - 1, 2, 3, 6, J1, 7, 8, J2
  - 1, 2, 3, 6, J1, 7, 9, J2
Def-Use Coverage

- **Def-use coverage:** every path from every definition of every variable to every use of that definition is exercised in some test.

- **Example**
  1. `read(x)`
  2. `read(z)`
  3. if `x ≠ 0` then begin
     4. `y ← x * z;
     5. `x ← z` end
     6. else print ‘Invalid’
  7. if `y > 1` then
     8. print `y`
    9. else print ‘Invalid’

Test Path: 1, 2, 3, 4, 5, 7, 8, J

Quiz 7: Develop Test Cases using Def-Use Coverage (5 Minutes)

- **Def-use coverage:** every path from every definition of every variable to every use of that definition is exercised in some test.

- **Example**
  1. `read(x)`
  2. `read(z)`
  3. if `x ≠ 0` then begin
     4. `y ← x * z;
     5. `x ← z` end
     6. else print ‘Invalid’
  7. if `y > 1` then
     8. print `y`
    9. else print ‘Invalid’

Test Path: 1, 2, 3, 4, 5, 7, 8, J
Strength of Coverage

Arrows point from weaker to stronger coverage.

Stronger coverage requires more test cases.

Interesting Cases

- Case (Switch)
- For-Loops
- Do-While Loops
Case (Switch Statement)

1. read (c) ;
2. switch ( c )
   {
3.     case 'N':
4.         y = 25;
5.         break;
6.     case 'Y':
7.         y = 50;
8.         break;
9.     default:
10.       y = 0;
11.       break;
12.     }
13. print (y);

Case (Switch Statement)

1. read (c) ;
2. switch ( c )
   {
3.     case 'N':
4.         y = 25;
5.     case 'Y':
6.         y = 50;
7.     default:
8.         y = 0;
9.     }
10. print (y);
For-Loops

1. for (x = 0; x < y; x++)
2. y = f (x, y);
3. print(y);

Do-While Loops

1. read (result);
2. read (x,k)
3. Do while result < 0 {
4. ptr ← false
5. if x > k then
6. ptr ← true
7. x ← x + 1
8. result ← result + 1 }
9. print result
Do-While Loops

1. read (result);
2. read (x,k)
3. Do while result < 0 {
4.   ptr ← false
5.   if x > k then
6.     ptr ← true
7.   x ← x + 1
8.   result ← result + 1 }
9. print result

Possible test set: dc stands for don’t care
{<result = 5>, x = dc, k = dc}

Group Exercise # 1

- Develop the CFG for the Triangle Problem Implementation
Group Exercise # 2

- Develop a Test Suite Using Branch Testing for the Triangle Implementation
- HINT: Start by identifying the upper limit on the number of tests you need and what the distinct paths are

Group Exercise # 3

- Develop Def-Use test cases
- Does it matter which variable(s) you pick?
Group Exercise # 4

- Develop test cases using condition coverage
- Let’s just give it a try
- You should be able to reuse some old test cases

Class Plan Until End of Semester

- Thursday 04/19: Exam 2 review
  - Might also extend review over the weekend
- Tuesday 04/24: Exam 2
- Thursday 04/26: Team Presentations (HL Design)
  - Teams 1, 4, 6, 8
- Tuesday 05/01: Team Presentations (DL Design)
  - Teams 2, 3, 7
- Thursday 05/03: Team Presentations (Testing)
  - Teams 5, 9, 10