Graph-Based Testing
CSE 5324
Graph-Based Testing

• Introduction
• Basic Concepts
• Control Flow Testing
• Data Flow Testing
• Summary
Motivation

• **Graph-based testing** first builds a graph model for the program under test, and then tries to cover certain elements in the graph model.
  – Graph is one of the most widely used structures for **abstraction**.
  – Graph is a well-defined, well-studied structure and is one of the most **fundamental data structures** in computer science.
Major Steps

• **Step 1**: Build a graph model
  – What information to be captured, and how to represent those information?

• **Step 2**: Identify test requirements (TR)
  – A test requirement is a structural entity in the graph model that must be covered during testing

• **Step 3**: Select test paths to cover those requirements

• **Step 4**: Derive test data so that those test paths can be executed
Graph Models

- **Control Flow Graph**: Captures information about how the control is transferred in a program.
- **Data Flow Graph**: Augments a CFG with data flow information
- **Dependency graph**: Captures the data/control dependencies among program statements
- **Cause-effect graph**: Modeling relationships among program input conditions, known as causes, and output conditions, known as effects
Graph-Based Testing

- Introduction
- Basic Concepts
- Control Flow Testing
- Data Flow Testing
- Summary
Graph

• A graph consists of a set of nodes and edges that connect pairs of nodes.
• Formally, a graph $G = \langle N, N_0, N_f, E \rangle$:
  – $N$: a set of nodes
  – $N_0 \subseteq N$: a set of initial nodes
  – $N_f \subseteq N$: a set of final nodes
  – $E \subseteq N \times N$: a set of edges
• In our context, $N$, $N_0$, and $N_f$ contain at least one node.
Example

\[N = \{n_0, n_1, n_2, n_3\}\]
\[N_0 = \{n_0\}\]
\[N_f = \{n_3\}\]
\[E = \{(n_0, n_1), (n_0, n_2), (n_1, n_3), (n_2, n_3)\}\]

\[N = \{n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9\}\]
\[N_0 = \{n_0, n_1, n_2\}\]
\[N_f = \{n_7, n_8, n_9\}\]
\[E = \{(n_0, n_3), (n_0, n_4), (n_1, n_4), (n_1, n_5), \ldots\}\]
Path, Subpath, Test Path

- **Path**
  - A sequence of nodes - \([n_1, n_2, \ldots, n_M]\)
    - Each pair of adjacent nodes \((n_i, n_{i+1})\) is an edge.

- **Length**
  - The number of edges in the path
    - A single node is a path of length 0

- **Subpath**
  - A subsequence of nodes in \(p\), is a subpath of \(p\).

- **Test path**
  - A path that starts at an **initial** node, and ends at a **final** node
    - Represents a path that is executed during a test run
Reachability

- A node $n$ is **syntactically reachable** from node $n'$ if there exists a path from $n'$ to $n$.
- A node $n$ is **semantically reachable** from node $n'$ if it is possible to execute a path from $n'$ to $n$ with some input.
- **reach$(n)$**: the set of nodes and edges that can be **syntactically** reached from node $n$. 
Example

\[ p_1 = [n_0, n_3, n_7] \]
\[ p_2 = [n_1, n_4, n_8, n_5, n_1] \]
\[ p_3 = [n_4, n_8, n_5] \]
\[ \text{reach}(n_0) = ? \]
\[ \text{reach}(n_5) = ? \]
SESE (Single Entry Single Exit) Graph

- All test paths start at a single node and end at another node
  - Single-Entry, Single-exit
  - $N_0$ and $N_f$ have exactly one node
Visiting & Touring

• Visit:
  – A test path $p$ visits node $n$ if $n$ is in $p$
  – A test path $p$ visits edge $e$ if $e$ is in $p$

• Tour:
  – A test path $p$ tours subpath $q$ if $q$ is a subpath of $p$.

Path $[0,1,3,4,6]$
Visits nodes 0,1,3,4,6
Visit Edges $(0,1),(1,3),(3,4),(4,6)$
Tours subpaths $(0,1,3), (1,3,4), (3,4,6), (0,1,3,4), (1,3,4,6)$
Test Case vs. Test Path

\[ a < b \]

\[ a > b \]

\[ a = b \]

\[ n_0 \quad n_1 \]

\[ n_2 \quad n_3 \]

\[ t_1: (a = 0, b = 1) \Rightarrow p_1 = [n_0, n_1, n_3, n_2] \]

\[ t_2: (a = 1, b = 1) \Rightarrow p_2 = [n_0, n_3, n_2] \]

\[ t_3: (a = 2, b = 1) \Rightarrow p_3 = [n_0, n_2] \]
Graph-Based Testing

- Introduction
- Basic Concepts
- Control Flow Testing
- Data Flow Testing
- Summary
Basic Block

• A basic block, or simply a block, is a sequence of consecutive statements with a single entry and a single exit point.

• Control always enters a basic block at its entry point, and exits from its exit point.

• If a basic block contains a single statement, then the entry and exit points coincide.
1. begin
2.   int x, y, power;
3.   float z;
4.   input (x, y);
5.   if (y < 0)
6.      power = -y;
7.   else
8.      power = y;
9.   z = 1;
10.  while (power != 0) {
11.     z = z * x;
12.     power = power – 1;
13.  }
14.  if (y < 0)
15.     z = 1/z;
16.  output (z);
17. end;

How many blocks does this code have?
Which Lines ?, Which one is Entry ? and Exit ?
Example

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

- Should a function call be treated like a regular statement or as a separate block of its own?

Function calls are often treated as blocks of their own because they cause the control to be transferred away from the currently executing function and hence raise the possibility of abnormal termination of the program.
Control Flow Graph

- A control flow graph is a graph with two distinguished nodes, start and end.
  - Node start has no incoming edges, and node end has no outgoing edges.
  - Every node can be reached from start, and can reach end.

- In a CFG, a node is typically a basic block, and an edge indicates the flow of control from one block to another.
Example
Node Coverage (NC)

- A test set $T$ satisfies **Node Coverage** on graph $G$ if and only if for every syntactically reachable node $n$ in $N$, there is some path $p$ in $\text{path}(T)$ such that $p$ visits $n$.
  - $\text{path}(T)$: the set of paths that are exercised by the execution of $T$
- In other words, the set $\text{TR}$ of test requirements for **Node Coverage** contains each reachable node in $G$. 

Edge Coverage (EC)

• The TR for Edge Coverage contains each reachable path of length up to 1, inclusive, in a graph G.

• Note that Edge Coverage subsumes Node Coverage.
Node Vs. Edge Coverage

Node Coverage:
TR = \{n_0,n_1,n_2\}
Test Path: [n_0,n_1,n_2]

Edge Coverage:
TR = \{(n_0,n_1),(n_0,n_2),(n_1,n_2)\}
Test Path = [n_0,n_1,n_2],[n_0,n_2]
Edge-Pair Coverage

• The TR for Edge-Pair Coverage contains each reachable path of length up to 2, inclusive, in a graph $G$.

• This definition can be easily extended to paths of any length, although possibly with diminishing returns.
Complete Path Coverage

• The TR for Complete Path Coverage contain all paths in a $G$.

How many paths do we need to cover in the above graph?
Structural Coverage Example

Node Coverage
TR = \{ 0, 1, 2, 3, 4, 5, 6 \}
Test Paths: [ 0, 1, 2, 3, 6 ] [ 0, 1, 2, 4, 5, 4, 6 ]

Edge Coverage
TR = \{ (0,1), (0,2), (1,2), (2,3), (2,4), (3,6), (4,5), (4,6), (5,4) \}
Test Paths: [ 0, 1, 2, 3, 6 ] [ 0, 2, 4, 5, 4, 6 ]

Edge-Pair Coverage
TR = \{ [0,1,2], [0,2,3], [0,2,4], [1,2,3], [1,2,4], [2,3,6], [2,4,5], [2,4,6], [4,5,4], [5,4,5], [5,4,6] \}
Test Paths: [ 0, 1, 2, 3, 6 ] [ 0, 1, 2, 4, 6 ] [ 0, 2, 3, 6 ] [ 0, 2, 4, 5, 4, 5, 4, 6 ] …

Complete Path Coverage
Test Paths: [ 0, 1, 2, 3, 6 ] [ 0, 1, 2, 4, 6 ] [ 0, 1, 2, 4, 5, 4, 6 ] [ 0, 1, 2, 4, 5, 4, 5, 4, 6 ] …
A path \( p \) is **simple** if it has no **repetitions** of nodes other than (possibly) the first and last node.

- So a simple path \( p \) has no internal loops, but may itself be a loop
- Problem: there are **too many** simple paths, since many are just sub-paths of longer simple paths.
Loops in Graphs

- If a graph contains a loop, it has an infinite number of paths

- Thus Complete Path Coverage (CPC) is not feasible

- Simple Path Coverage (SPC) is not satisfactory because the results are subjective and vary with the tester
Prime Path

- A path is a prime path if it is a simple path, and it does not appear as a proper subpath of any other simple path.
- In other words, a path $p$ is prime iff $p$ is a maximal simple path.

- This cuts down the number of cases to consider.
Prime Path Coverage (PPC)

• The TR for Prime Path Coverage contains each prime path in a $G$. 
Prime paths =
(n0, n1, n2),
(n0, n1, n3, n4),
(n1, n3, n4, n1),
(n3, n4, n1, n3),
(n4, n1, n3, n4),
(n3, n4, n1, n2)
Exercise-1

1. Give the sets N, N₀, Nᶠ, and E for the following graph
2. Give a path that is not a test path
Exercise-2

1. Find test case inputs such that the corresponding test path visits edge \((n1, n3)\).
2. Find test case inputs such that the corresponding test path visits edge \((n0, n3)\).
2. Find test case inputs such that the corresponding test path visits edge \((n3, n2)\).
Graph-Based Testing

- Introduction
- Basic Concepts
- Control Flow Testing
- Data Flow Testing
- Summary
Definition/Use

• A **definition** is a location where a value for a variable is stored into memory.
  – Assignment, input, parameter passing, etc.

• A **use** is a location where a variable’s value is accessed.
  – p-use: a use that occurs in a predicate expression, i.e., an expression used as a condition in a branch statement
  – c-use: a use that occurs in an expression that is used to perform certain computation
**Def., P-use, and C-use - Example**

1. read(x,y)
2. \( z = x+2 \)
3. If(z<y)
4. \( w = x+1 \)
   else
5. \( y = y+1 \)
6. print(x,y,w,z);

<table>
<thead>
<tr>
<th>Line</th>
<th>Def.</th>
<th>C-use</th>
<th>P-use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x,y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>z</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>z,y</td>
</tr>
<tr>
<td>4</td>
<td>w</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>y</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>x,y,z,w</td>
</tr>
</tbody>
</table>
A data flow graph (DFG) captures the flow of data in a program.

To build a DFG, we first build a CFG and then annotate each node $n$ in the CFG with the following two sets:

- $\text{def}(n)$: the set of variables defined in node $n$
- $\text{use}(n)$: the set of variables used in node $n$
Example (1)

1. begin
2. float x, y, z = 0.0;
3. int count;
4. input (x, y, count);
5. do {
6.     if (x <= 0) {
7.         if (y >= 0) {
8.             z = y * z + 1;
9.         }
10.     }
11.     } else {
12.         z = 1/x;
13.     }
14.     y = x * y + z;
15.     count = count – 1;
16.     while (count > 0)
17.     output (z);
18. end

<table>
<thead>
<tr>
<th>Node</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>2</td>
<td>5, 6</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8, 9, 10</td>
</tr>
<tr>
<td>5</td>
<td>11, 12, 13</td>
</tr>
<tr>
<td>6</td>
<td>14, 15, 16</td>
</tr>
<tr>
<td>7</td>
<td>17, 18</td>
</tr>
</tbody>
</table>
Example (2)

1. def={x, y, z, count}
   - use = {x}

2. def={}
   - use = {z}
   - x <= 0
   - x > 0

3. def={}
   - use = {y}
   - y >= 0
   - y < 0

4. def={z}
   - use = {y, z}

5. def={z}
   - use = {x}
   - y < 0

6. def={count, y}
   - use = {count, x, y, z}
   - count == 0

7. def={}
   - use = {z}
Graph-Based Testing

• Introduction
• Basic Concepts
• Control Flow Testing
• Data Flow Testing
• Summary
Recap

- **Graph** provides a good basis for systematic test selection.
- **Control flow testing** focuses on the transfer of control, while **data flow testing** focuses on the definitions of data and their subsequent use.
- Control flow coverage is defined in terms of **nodes, edges, and paths**; data flow coverage is defined in terms of **def, use, and du-path**.