CS 5154: Software Testing

Applying Graph Based Coverage to Source Code

Instructor: Owolabi Legunsen

Fall 2021
Implementing Graph-based MDTD

• Develop a model of the software as a graph

• Require tests to visit/tour sets of nodes, edges, or sub-paths

• Choose inputs that satisfy the test requirements

• Implement and automate tests based on the inputs chosen
Relating “Abstract Design” to Source Code

• **Graph**: Usually the control flow graph (CFG)

• **Nodes**: Statements or statement sequences (basic blocks)

• **Edges**: Transfers of control

• **Loops**: structures such as for loops, while loops, etc
Relating Graph Coverage Criteria to Source Code

- **Node coverage** : Execute every statement (i.e., statement coverage)
- **Edge coverage** : Execute every branch (i.e., branch coverage)
- **Edge-pair coverage** : ??
- **Prime-path Coverage** : Execute every statement, branch, loop
An essential concept for creating CFGs

• **Basic Block**: A sequence of statements such that if the first statement is executed, all statements in the sequence will be executed (no branches)

• Implication: Put all statements in a basic block in one CFG node
  • We will see one exception to the rule
Rules for creating CFG from Java source code

• We show one rule/template for commonly used Java features

• There are other sets of rules that can be used

• Differences in the sets of rules are usually not so important for testing
Rule 1: if-then-else

```java
if (x < y) {
    y = 0;
    x = x + 1;
} else {
    x = y;
}
```
Rule 2: if-then

```java
if (x < y)
{
    y = 0;
    x = x + 1;
}
```
Rule 3: if-with-return

```c
if (x < y)
{
    return;
}
print (x);
return;
```
Rule 4: while

```c
x = 0;
while (x < y)
{
    y = f(x, y);
    x = x + 1;
}
return (x);
```
Rule 5: do-while

```
x = 0;
do
{  
y = f(x, y);
x = x + 1;
}
while (x < y);
return (y);
```
Rule 6: for

```c
for (x = 0; x < y; x++)
{
    y = f(x, y);
}
return (x);
```

**Diagram:**
- Node 1: `x = 0`
- Node 2: `x < y` → `x >= y`
- Node 3: `y = f(x, y)`
- Node 4: `x = x + 1`
- Node 5: Implicitly initializes loop
- Node: Implicitly increments loop
Rule 7: break and continue

```c
x = 0;
while (x < y) {
    y = f (x, y);
    if (y == 0) {
        break;
    } else if (y < 0)  {
        y = y*2;
        continue;
    }
    x = x + 1;
}
return (y);
```
Rule 8: switch

```
read ( c ) ;
switch ( c )
{
    case ‘N’:
        z = 25;
        break;
    case ‘Y’:
        x = 50;
        break;
    default:
        x = 0;
        break;
}
print (x);
```

Cases without breaks fall through to the next case
try {
    s = br.readLine();
    if (s.length() > 96)
        throw new Exception("too long");
    if (s.length() == 0)
        throw new Exception("too short");
} (catch IOException e) {
    e.printStackTrace();
} (catch Exception e) {
    e.getMessage();
}
return (s);
Rule 10: putting it all together

• Real programs will require using more than one of these rules

• Real programs can get very large

• Real programs may require features that we do not cover
  • Recursion
  • Inter-procedural calls
Implementing Graph-based MDTD

• Develop a model of the software as a graph

• Require tests to visit/tour sets of nodes, edges, or sub-paths

• Choose inputs that satisfy the test requirements

• Implement and automate tests based on the inputs chosen
/** Return first index of Node n in path, or
 * -1 if n is not present in path */

public int indexOf (Node n, List<Node> path){
    for (int i=0; i < path.size(); i++){
        if (path.get(i).equals(n))
            return i;
    }
    return -1;
}
Summary

• Basic definition and terminology

• Graph Coverage Criteria and their relationships

• Obtaining graphs from source code

• You will apply Graph coverage on the next homework
Next

• Quiz

• Logic-based testing