Software Testing, Quality Assurance and Maintenance Winter 2015

Lecture 9 — January 23, 2015

Patrick Lam *version 1*

Graph Coverage for Source Code

So far, we've seen a number of coverage criteria for graphs, but I've been vague about how to actually construct graphs. For the most part, it's fairly obvious.

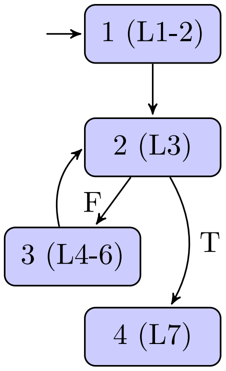
Structural Graph Coverage for Source Code

Fundamental graph for source code: *Control-Flow Graph* (CFG).

- CFG nodes: zero or more statements;
- CFG edges: an edge (s_1, s_2) indicates that s_1 may be followed by s_2 in an execution.

Basic Blocks. We can simplify a CFG by grouping together statements which always execute together (in sequential programs):

```
1       x = 5
2       z = 2
3  q0: if (z < 17) goto q1
4       z = z + 1
5       print (x)
6       goto q0
7  q1: nop
```



We use the following definition:

Definition 1 *A basic block has one entry point and one exit point.*

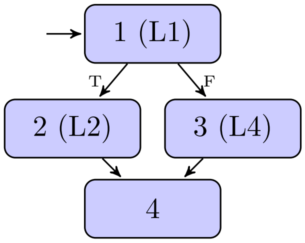
Note that a basic block may have multiple successors. However, there may not be any jumps into the middle of a basic block (which is why statement `l0` has its own basic block.)

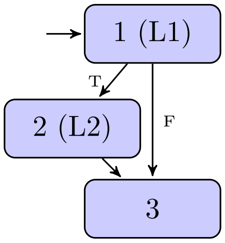
Some Examples

We'll now see how to construct control-flow graph fragments for various program constructs.

if statements: The book puts the conditions (and hence uses) on the control-flow edges, rather than in the `if` node. I prefer putting the condition in the node.

```
if (z < 17)
  print (x);
else
  print (y);
```





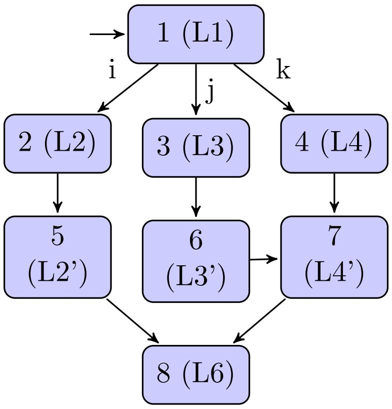
```
if (z < 17)
  print(x);
```

Short-circuit `if` evaluation is more complicated; I recommend working it out yourself.

(Recall that node coverage does not imply edge coverage.)

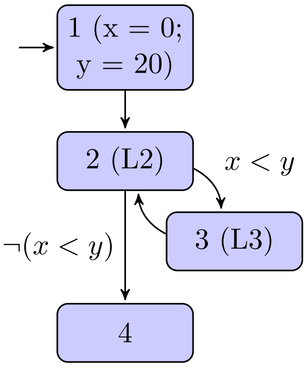
case / switch statements:

```
switch (n) {
  case 'I': ...; break;
  case 'J': ...; /* fall thru */
  case 'K': ...; break;
}
// ...
```



while statements:

```
x = 0; y = 20;
while (x < y) {
  x ++; y --;
}
```



Note that arbitrarily complicated structures may occur inside the loop body.

for statements:

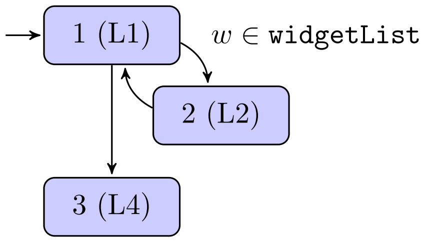
```
for (int i = 0; i < 57; i++) {
  if (i % 3 == 0) {
    print (i);
  }
}
```

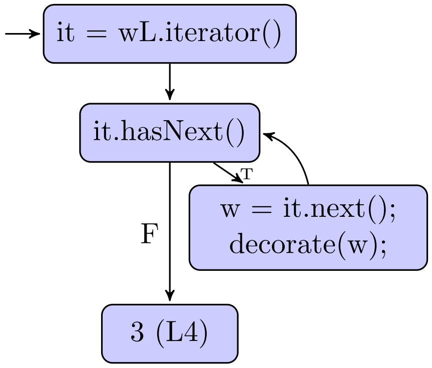
(an exercise for the reader!)

This example uses Java's enhanced for loops, which iterates over all of the elements in the `widgetList`:

```
for (Widget w : widgetList) {
  decorate(w);
}
// ...
```

I will accept the simplified CFG or the more useful one on the right:

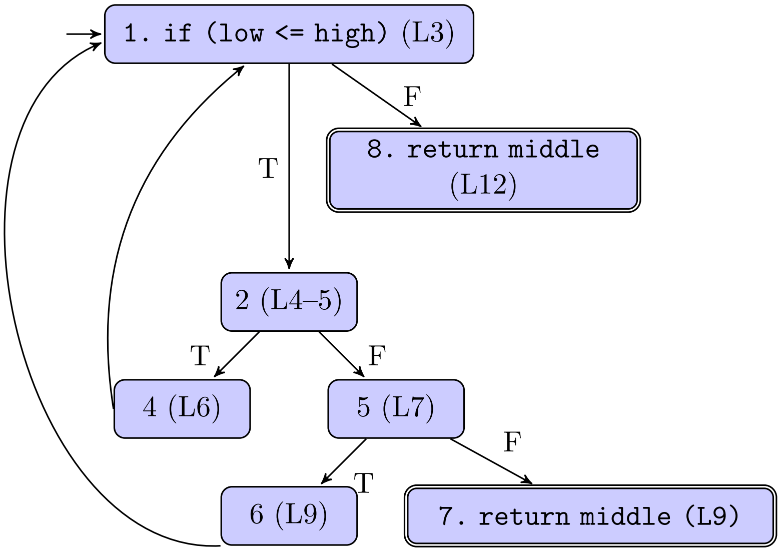




All of these graphs admit the notions of node coverage (statement coverage, basic block coverage) and edge coverage (branch coverage).

Larger example. You can draw a 7-node CFG for this program:

```
  /** Binary search for target in sorted subarray a[low..high] */
  int binary_search(int[] a, int low, int high, int target) {
    while (low <= high) {
      int middle = low + (high-low)/2;
      if (target < a[middle)
        high = middle - 1;
      else if (target > a[middle])
        low = middle + 1;
      else
        return middle;
    }
    return -1; /* not found in a[low..high] */
  }
```



Exercises

Here are more exercise programs that you can draw CFGs for.

```
/* effects: if x==null, throw NullPointerException
            otherwise, return number of elements in x that are odd, positive or both. */
int oddOrPos(int[] x) {
  int count = 0;
  for (int i = 0; i < x.length; i++) {
    if (x[i]%2 == 1 || x[i] > 0) {
      count++;
    }
  }
  return count;
}

// example test case: input: x=[-3, -2, 0, 1, 4]; output: 3
```

Next, we have a really poorly-designed API (I'd give it a D at most, maybe an F) because it's impossible to succinctly describe what it does. **Do not design functions with interfaces like this.** But we can still draw a CFG, no matter how bad the code is.

```
  /** Returns the mean of the first maxSize numbers in the array,
      if they are between min and max. Otherwise, skip the numbers. */
  double computeMean(int[] value, int maxSize, int min, int max) {
    int i, ti, tv, sum;

    i = 0; ti = 0; tv = 0; sum = 0;
    while (ti < maxSize) {
      ti++;
      if (value[i] >= min && value[i] <= max) {
        tv++;
        sum += value[i];
      }
      i++;
    }
    if (tv > 0)
      return (double)sum/tv;
    else
      throw new IllegalArgumentException();
  }
```