COMP5111 – Fundamentals of Software Testing and Analysis
Symbolic Execution

Shing-Chi Cheung

Computer Science & Engineering

HKUST

Automatic Software Testing

- Random testing
- Symbolic testing
- Concolic testing

Automatic Software Testing

- Random testing
- Symbolic testing
- Concolic testing

Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
  if (x>y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
        assert (false); // bug
  }
}
```

- Key idea: execute programs using symbolic input values instead of concrete execution
- Concrete execution x=0, y=1
- Symbolic execution x=a, y=b

Adapted from http://www.zvonimir.info/teaching/sv-2012-fall/sv_lecture_11.pdf

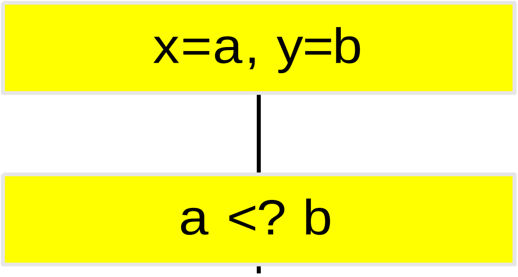
Symbolic Testing (a.k.a. Symbolic Execution)

x=a, y=b

```
☞ foo (int& x, int& y) {
   if (x>y) {
     x = x + y;
     y = x - y;
     x = x - y;
     if (x - y > 0)
          assert (false); // bug
   }
  }
```

Symbolic Testing (a.k.a. Symbolic Execution)

```
  foo (int& x, int& y) {
☞   if (x>y) {
      x = x + y;
      y = x - y;
      x = x - y;
      if (x - y > 0)
          assert (false); // bug
    }
  }
```

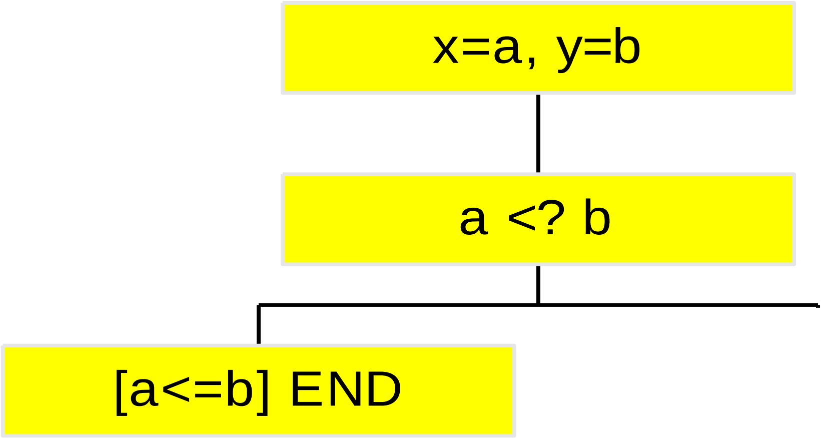


x=a, y=b

a <? b

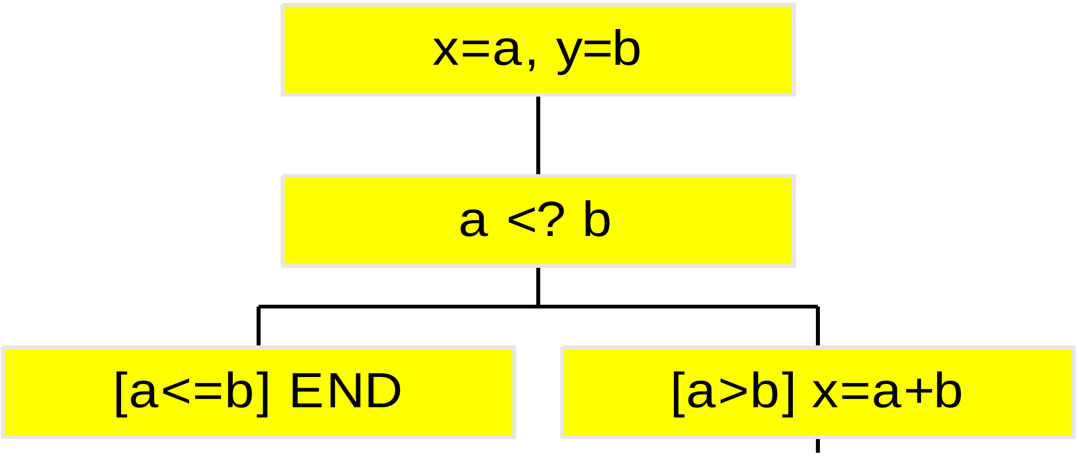
Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
 if (x>y) {
   x = x + y;
   y = x - y;
   x = x - y;
   if (x - y > 0)
       assert (false); // bug
 }
}
```



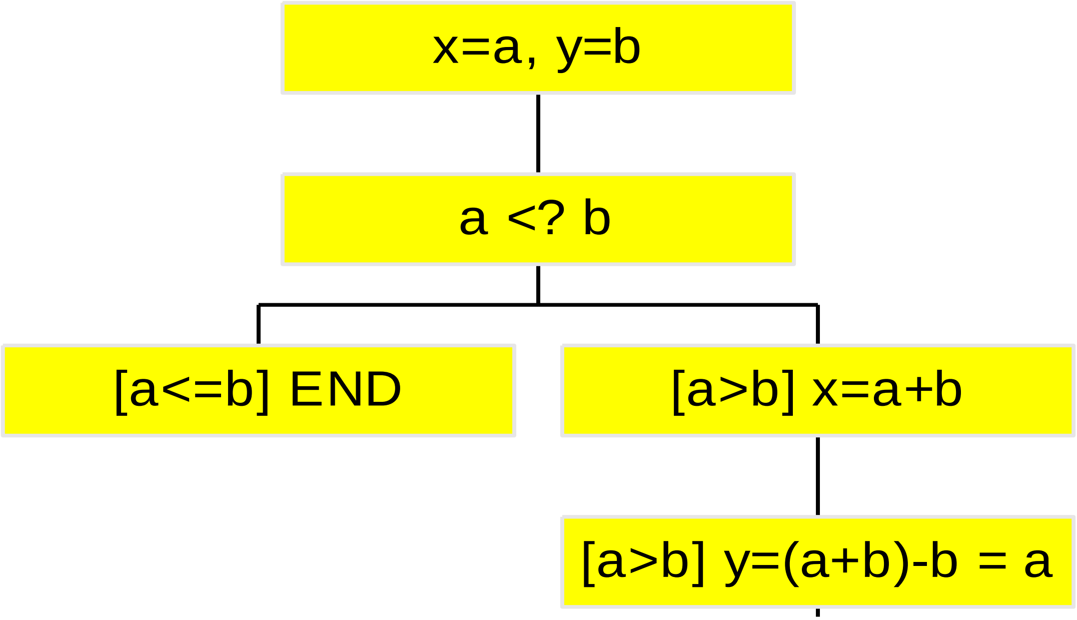
Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
  if (x>y) {
☞   x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
        assert (false); // bug
  }
}
```



Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
 if (x>y) {
   x = x + y;
   y = x - y;
   x = x - y;
   if (x - y > 0)
       assert (false); // bug
 }
}
```



x=a, y=b

a <? b

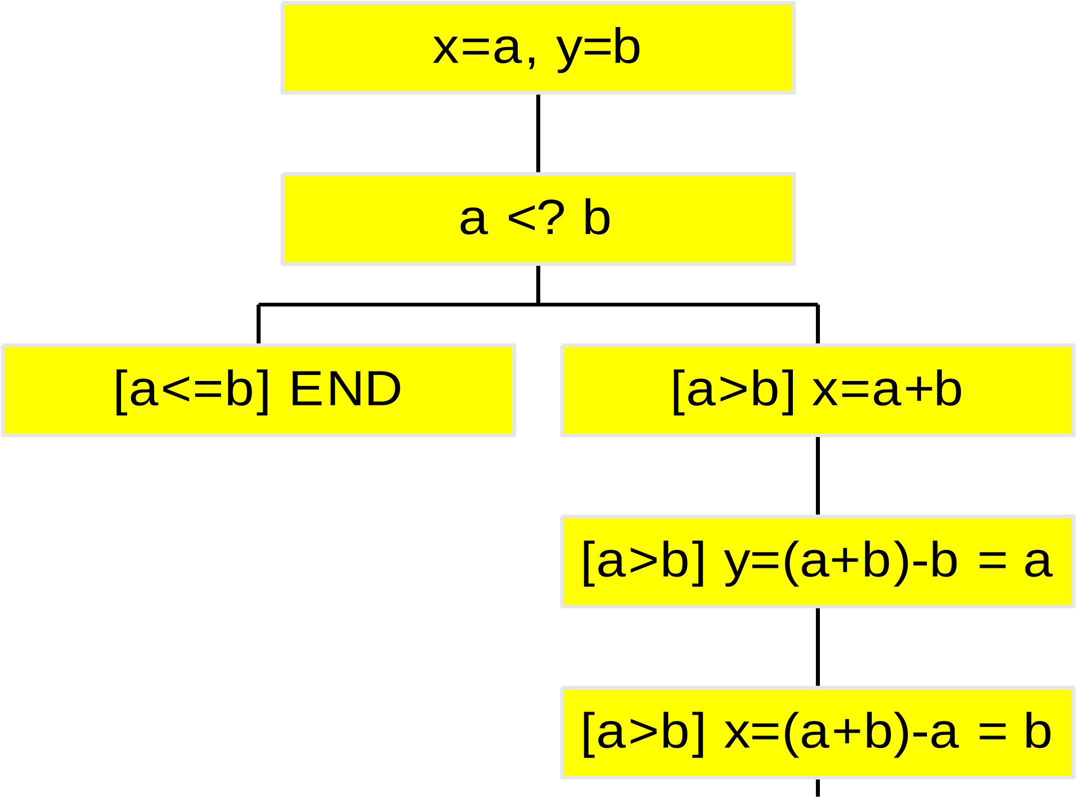
[a<=b] END

[a>b] x=a+b

[a>b] y=(a+b)-b = a

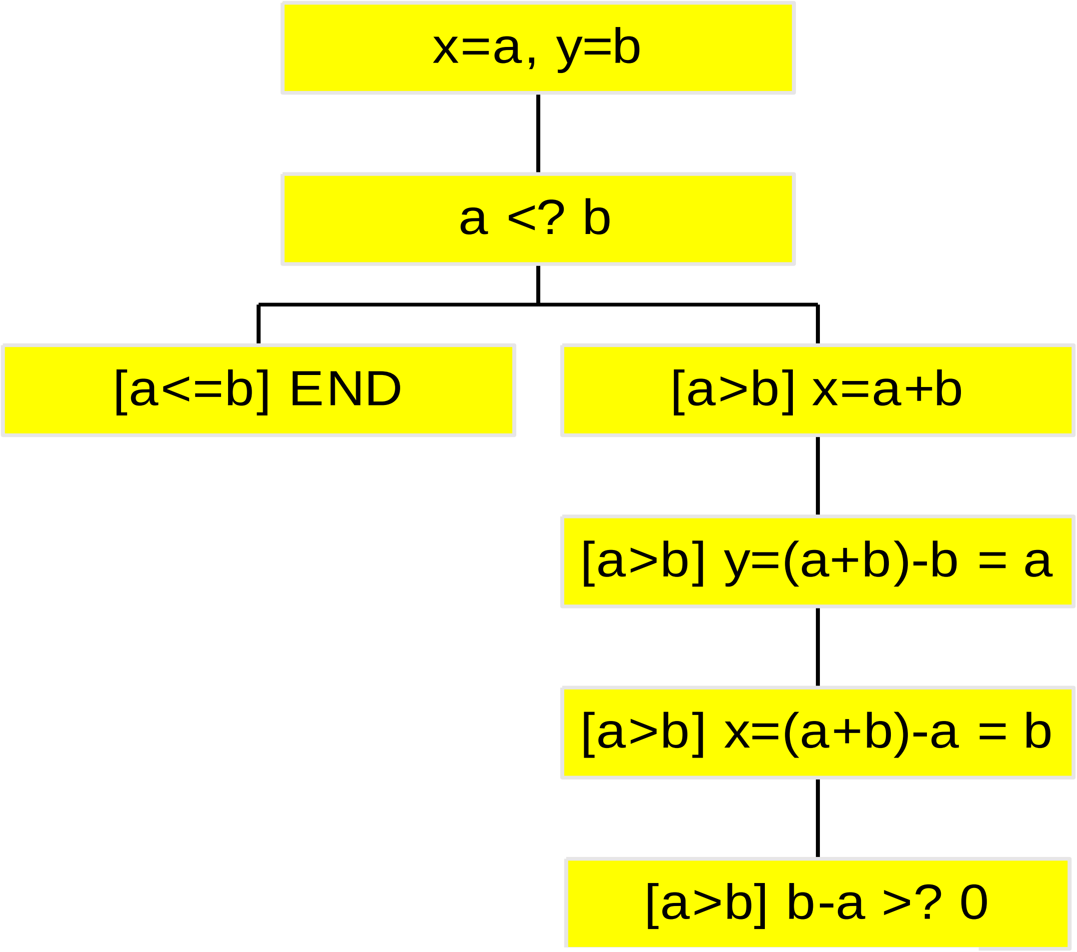
Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
  if (x>y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
         assert (false); // bug
  }
}
```



Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
  if (x>y) {
    x = x + y;
    y = x - y;
    x = x - y;
☞   if (x - y > 0)
        assert (false); // bug
  }
}
```



x=a, y=b

a <? b

[a<=b] END

[a>b] x=a+b

[a>b] y=(a+b)-b = a

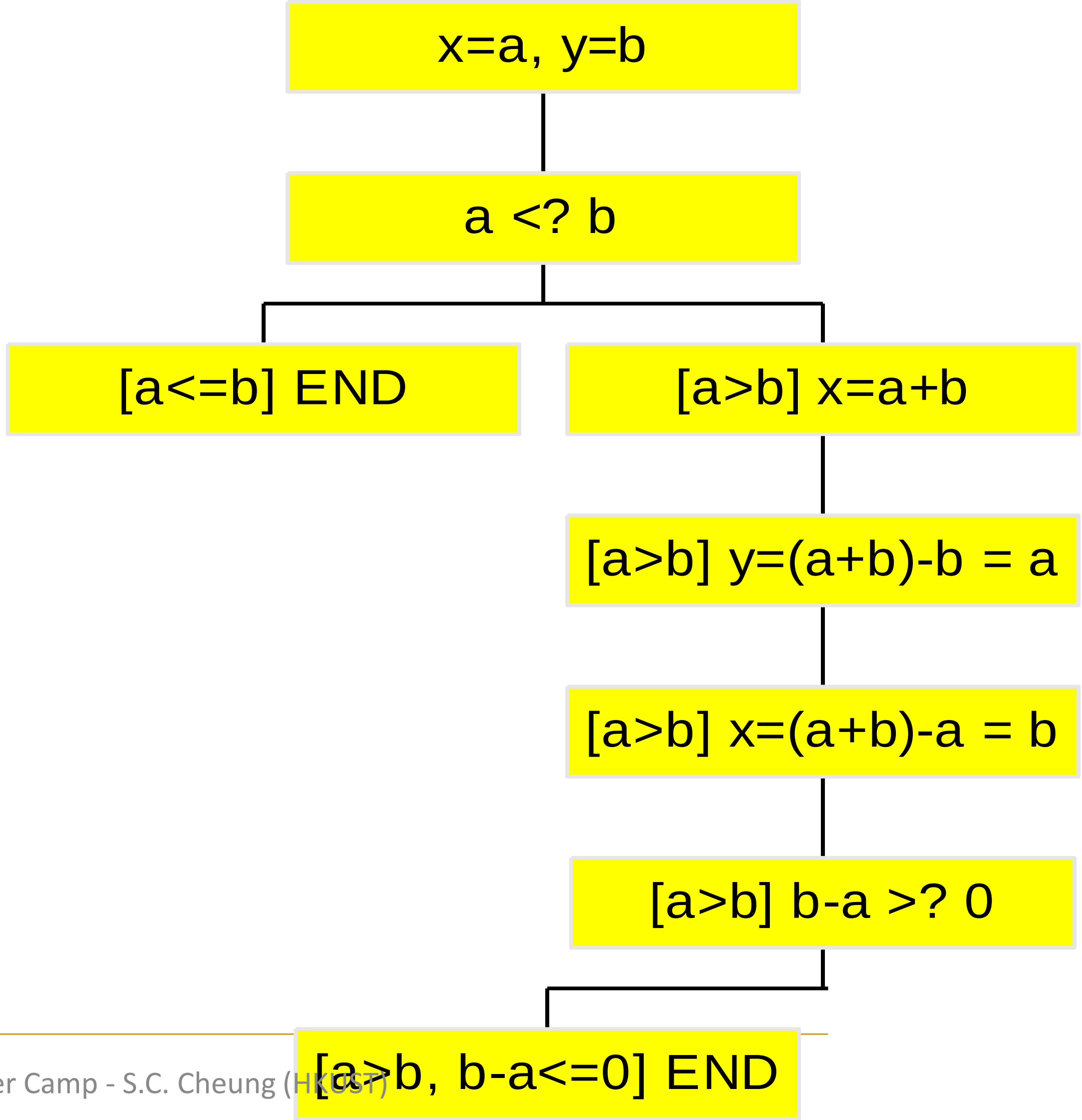
[a>b] x=(a+b)-a = b

[a>b] b-a >? 0

Symbolic Testing (a.k.a. Symbolic Execution)

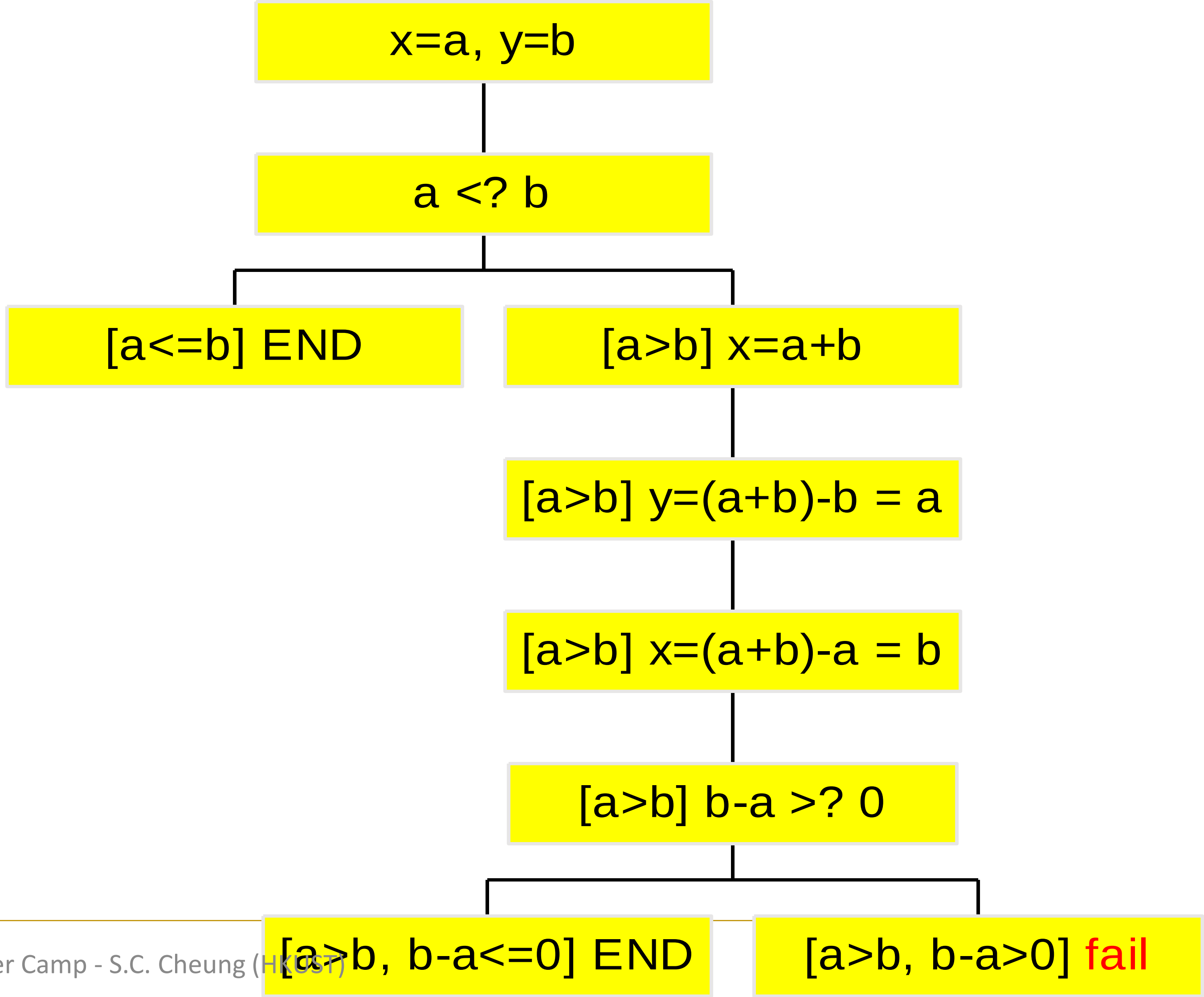
```
foo (int& x, int& y) {
 if (x>y) {
   x = x + y;
   y = x - y;
   x = x - y;
   if (x - y > 0)
      assert (false); // bug
 }
}
```

☞



Symbolic Testing (a.k.a. Symbolic Execution)

```
foo (int& x, int& y) {
  if (x>y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
        assert (false); // bug
  }
}
```



x=a, y=b

a <? b

[a<=b] END

[a>b] x=a+b

[a>b] y=(a+b)-b = a

[a>b] x=(a+b)-a = b

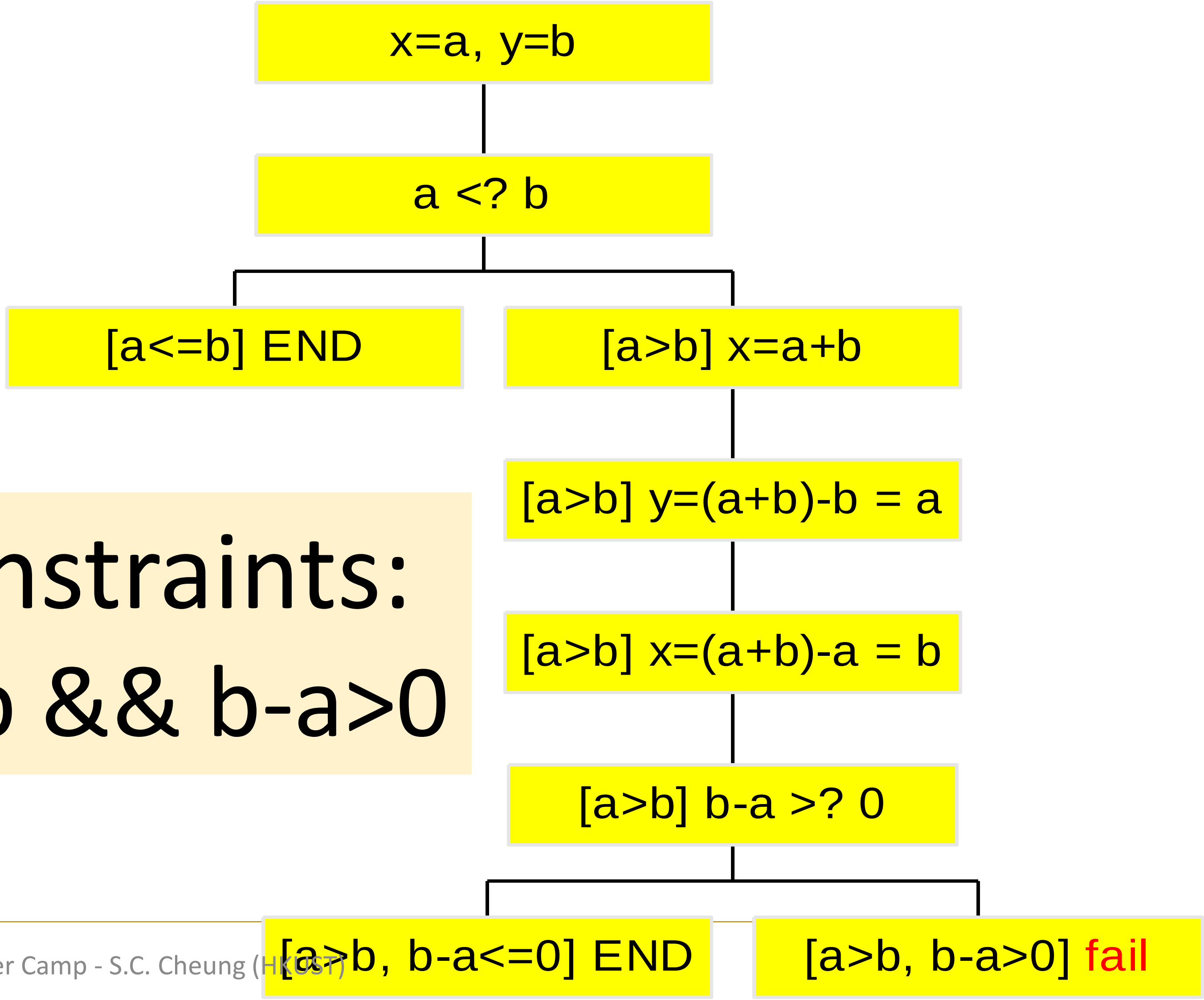
[a>b] b-a >? 0

[a>b, b-a<=0] END

[a>b, b-a>0] fail

Symbolic Testing (Symbolic Execution Tree)

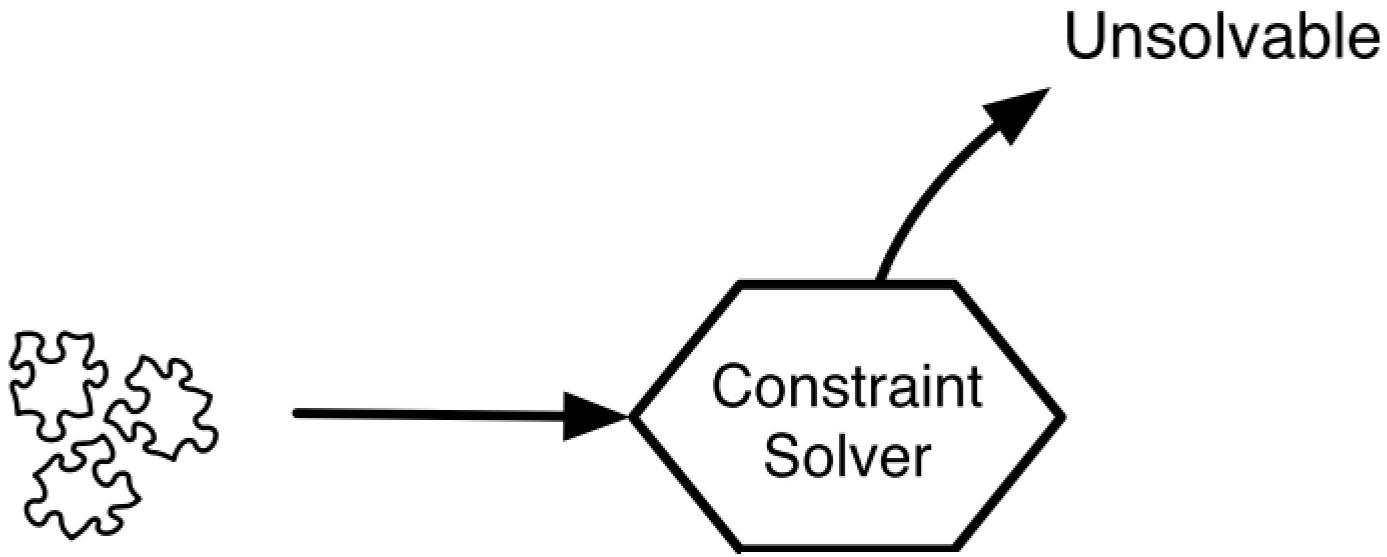
```
foo (int& x, int& y) {
  if (x>y) {
    x = x + y;
    y = x - y;
    x = x - y;
    if (x - y > 0)
☞       assert (false); // bug
  }
}
```



Constraints:

a>b && b-a>0

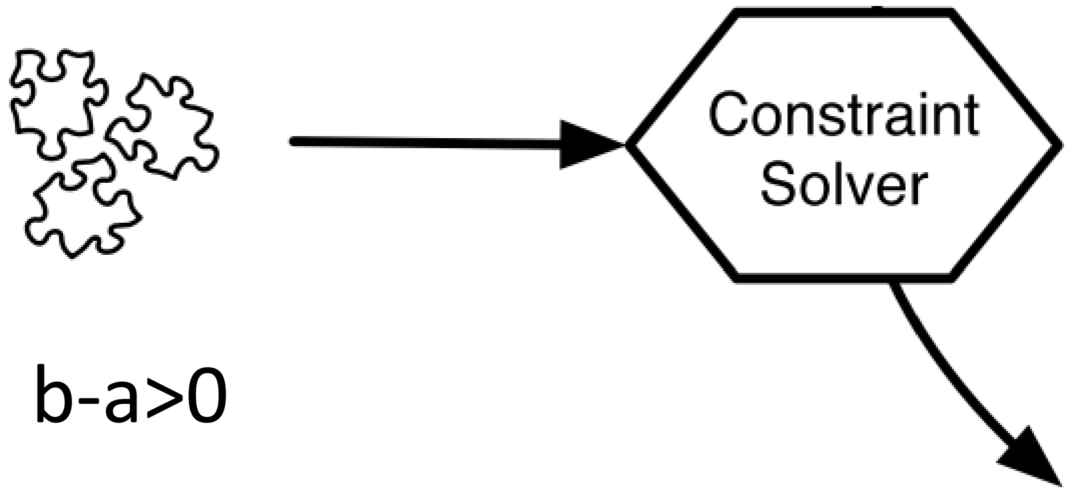
Using a Linear Constraint Solver



a>b && b-a>0

Efficient linear constraint solvers are available

Constraint Solving with What-if Analysis



$b-a>0$

$a = 0, b = 1$

Further reading: Roberto Baldoni et al. A Survey of Symbolic Execution Techniques, ACM Computing Surveys 51 (3), 2018.

Automatic Software Testing

- Random testing
- Symbolic testing
- Concolic testing

Koushik Sen

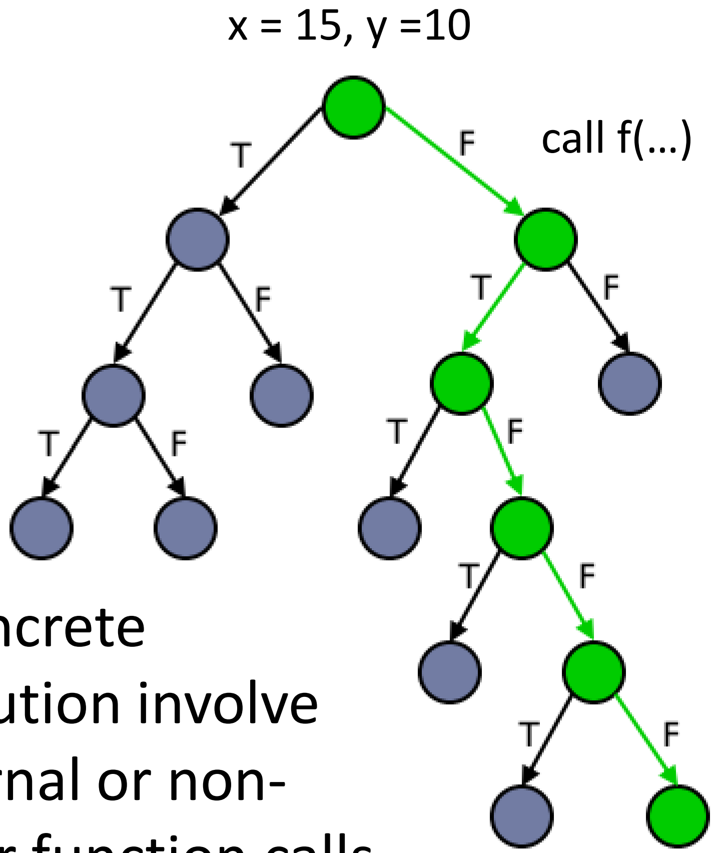
Associate Professor, UC Berkeley

Research Areas

Programming Systems, Software Engineering, Programming Languages, and Formal Methods: Software Testing, Verification, Model Checking, Runtime Monitoring, Performance Evaluation, and Computational Logic

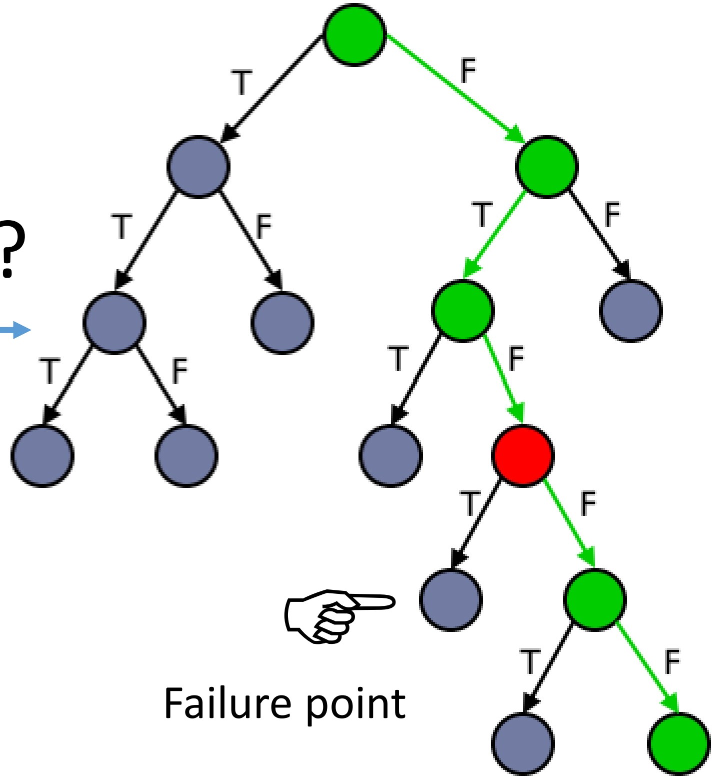
Security

Concolic = Concrete + Symbolic



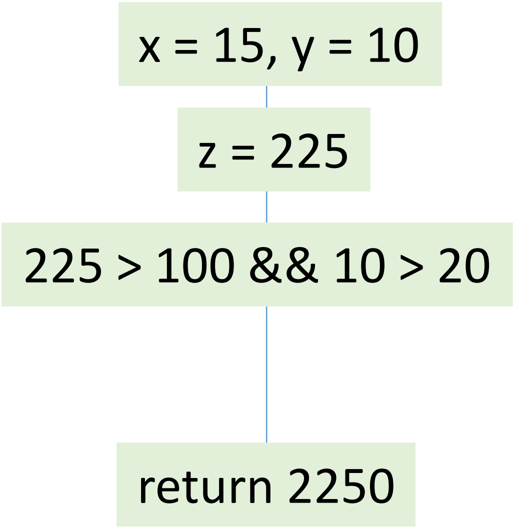
A concrete execution involve external or non-linear function calls

inputs of x and y?



Concolic = Concrete + Symbolic

```
int foo(int x, int y) {
  int z = square(x);
  if (z > 100 && y > 20)
    assert(false);
  return y*z;
}
```



x = 15, y = 10

z = 225

225 > 100 && 10 > 20

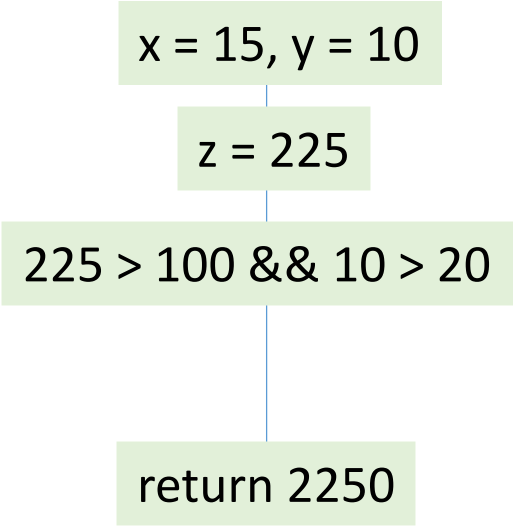
return 2250

Execute program concretely

Test: foo(15, 10)

Concolic = Concrete + Symbolic

```
int foo(int x, int y) {
  int z = square(x);
  if (z > 100 && y > 20)
    assert(false);
  return y*z;
}
```

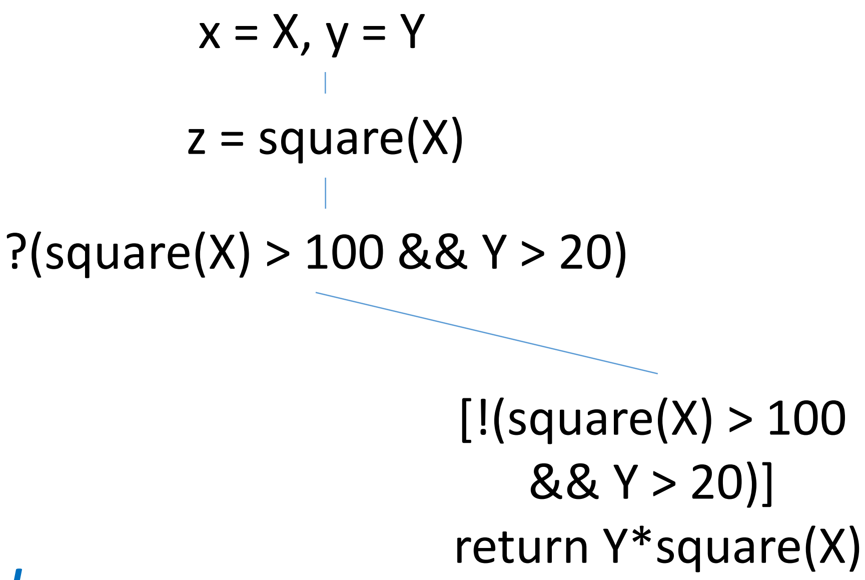


x = 15, y = 10

z = 225

225 > 100 && 10 > 20

return 2250



x = X, y = Y

z = square(X)

?(square(X) > 100 && Y > 20)

[!(square(X) > 100 && Y > 20)]
return Y*square(X)

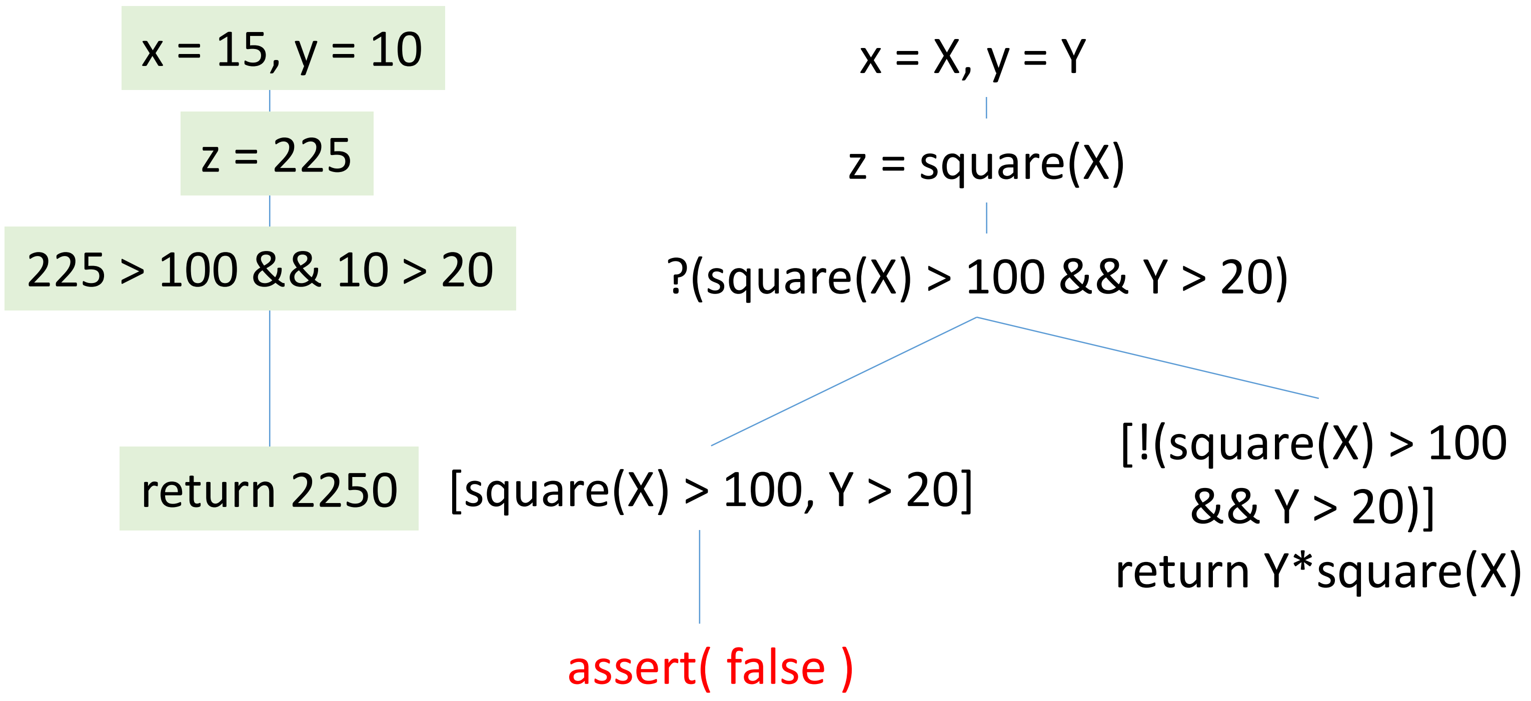
Execute program concretely

Collect symbolic path condition

Test: foo(15, 10)

Concolic Testing

```
int foo(int x, int y) {
  int z = square(x);
  if (z > 100 && y > 20)
    assert(false);
  return y*z;
}
```



x = 15, y = 10

z = 225

225 > 100 && 10 > 20

return 2250

x = X, y = Y

z = square(X)

?(square(X) > 100 && Y > 20)

[square(X) > 100, Y > 20]

assert(false)

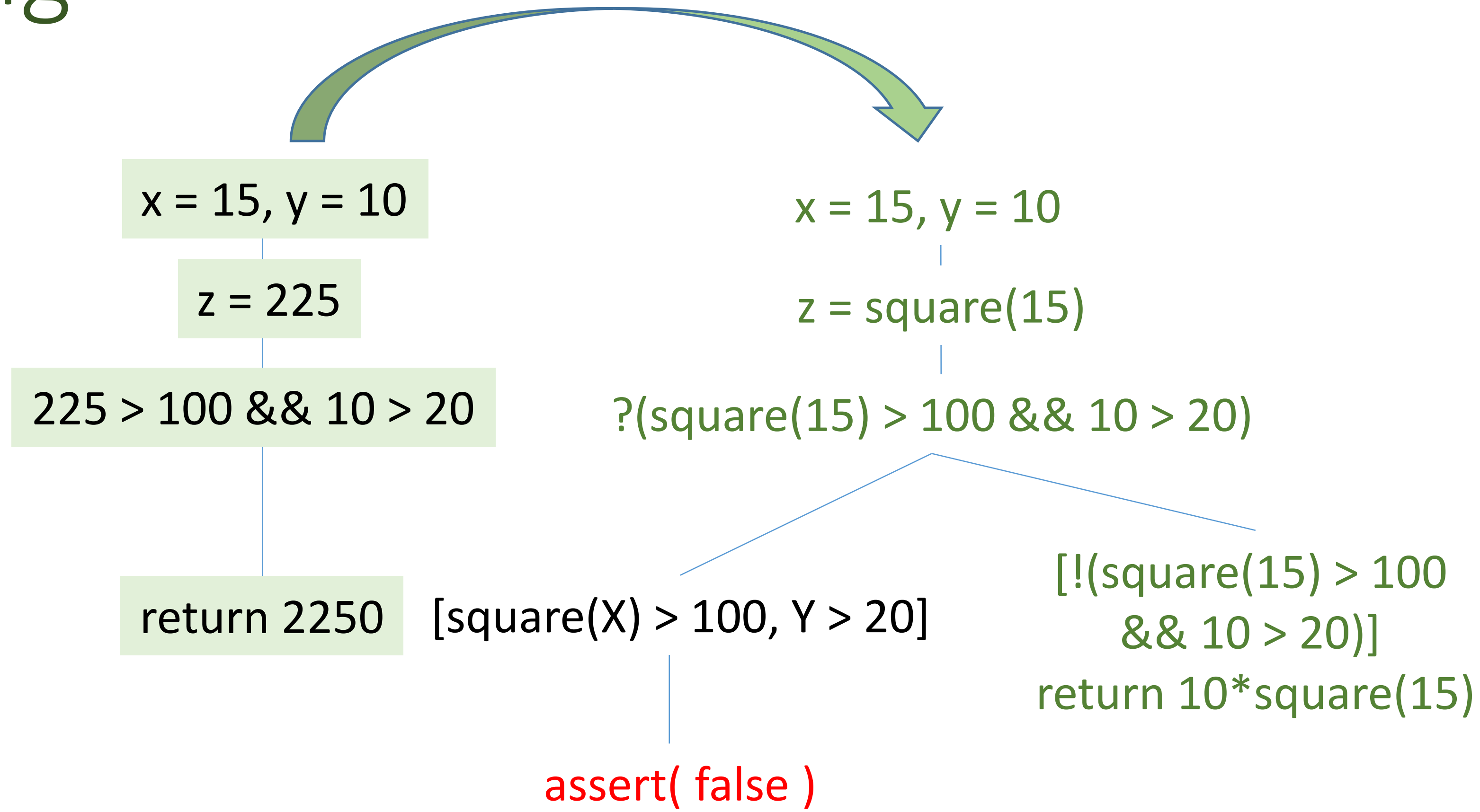
[!(square(X) > 100 && Y > 20)]
return Y*square(X)

Test: foo(15, 10)

Execute program concretely
Collect symbolic path condition
Negate a constraint on the path condition and solve it

Concolic Testing

```
int foo(int x, int y) {
  int z = square(x);
  if (z > 100 && y > 20)
    assert(false);
  return y*z;
}
```



x = 15, y = 10

z = 225

225 > 100 && 10 > 20

return 2250

x = 15, y = 10

z = square(15)

?(square(15) > 100 && 10 > 20)

[square(X) > 100, Y > 20]

assert(false)

[!(square(15) > 100 && 10 > 20)]
return 10*square(15)

Test: foo(15, 10)

The concrete test and our target share a long prefix in execution

→ *The concrete test inputs should partially solve the negated path condition*

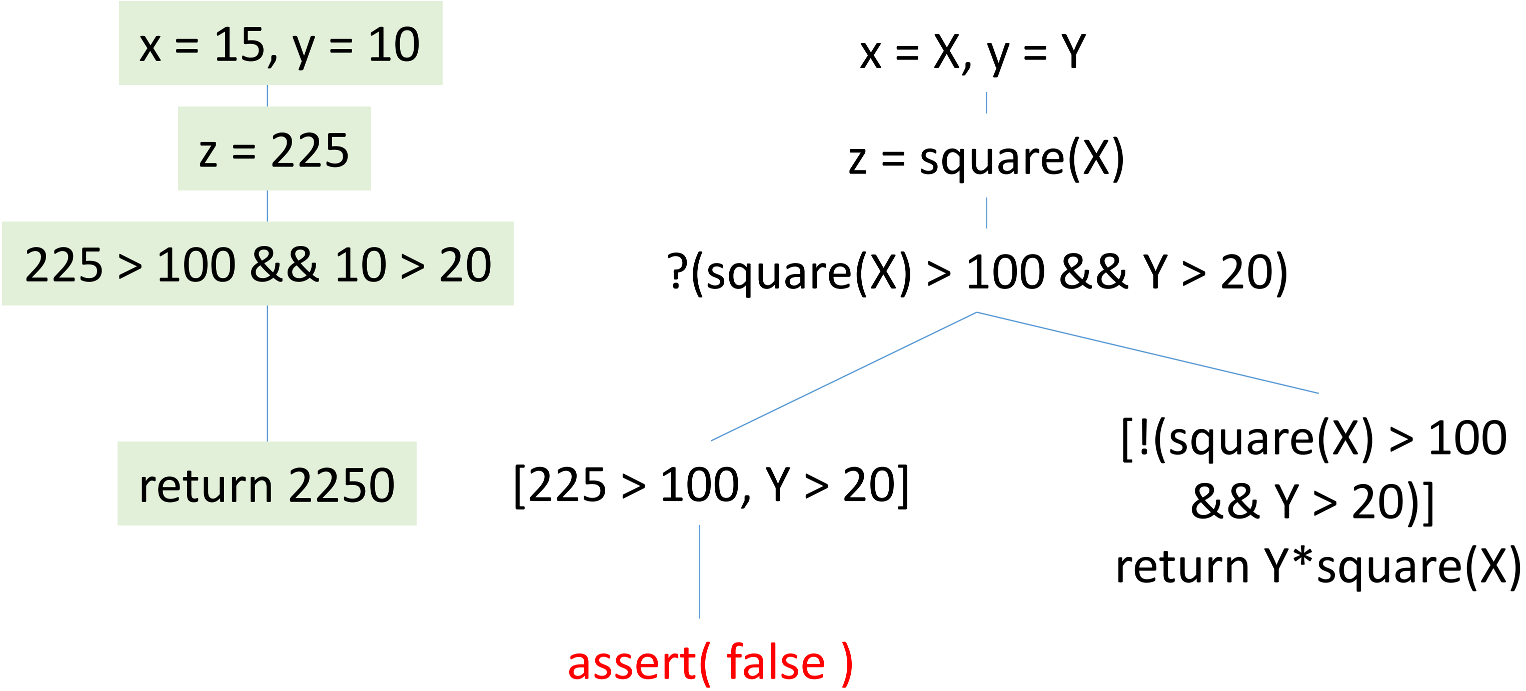
→ *Only need to solve remaining unsolved constraints, which are likely linear*

Concolic Testing

```
int foo(int x, int y) {
  int z = square(x);
  if (z > 100 && y > 20)
    assert(false);
  return y*z;
}
```

Test: foo(15, 10)

Test: foo(15, 21)



x = 15, y = 10

z = 225

225 > 100 && 10 > 20

return 2250

x = X, y = Y

z = square(X)

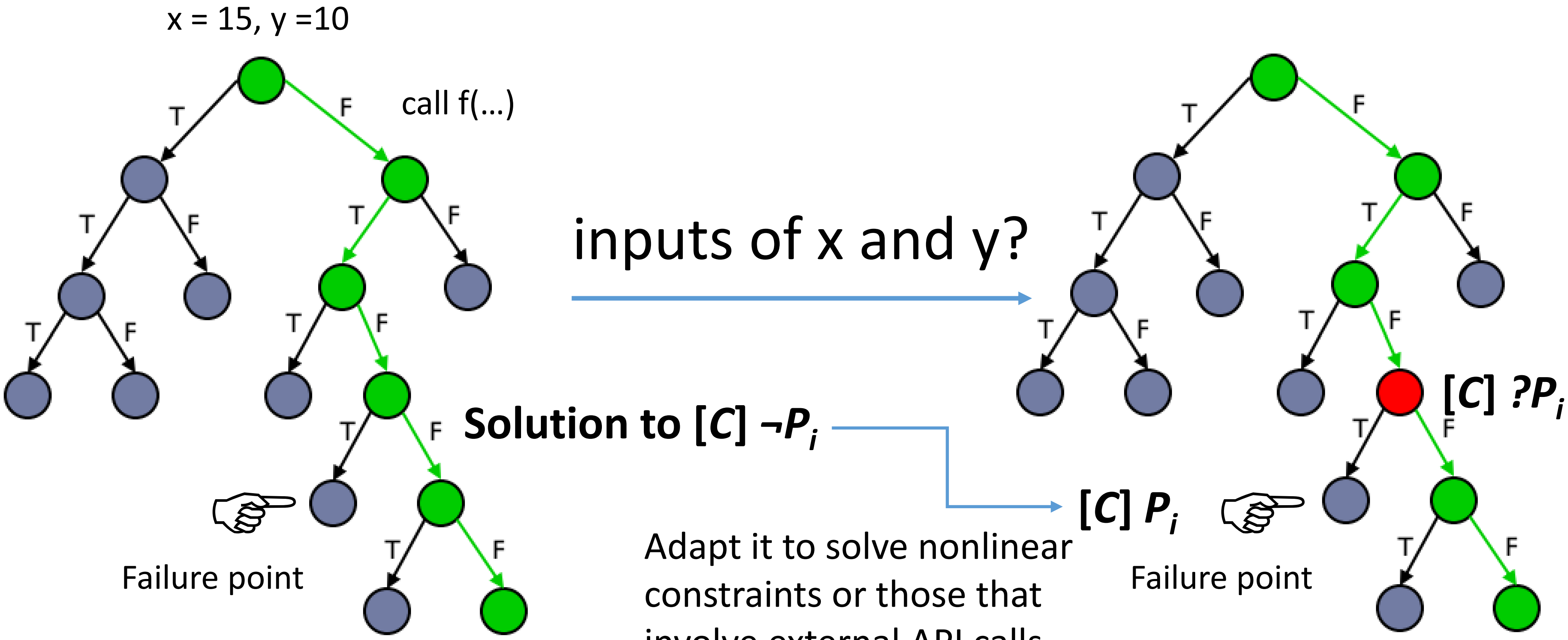
?(square(X) > 100 && Y > 20)

[225 > 100, Y > 20]

assert(false)

[!(square(X) > 100 && Y > 20)]
return Y*square(X)

Concolic = Concrete + Symbolic (Summary)



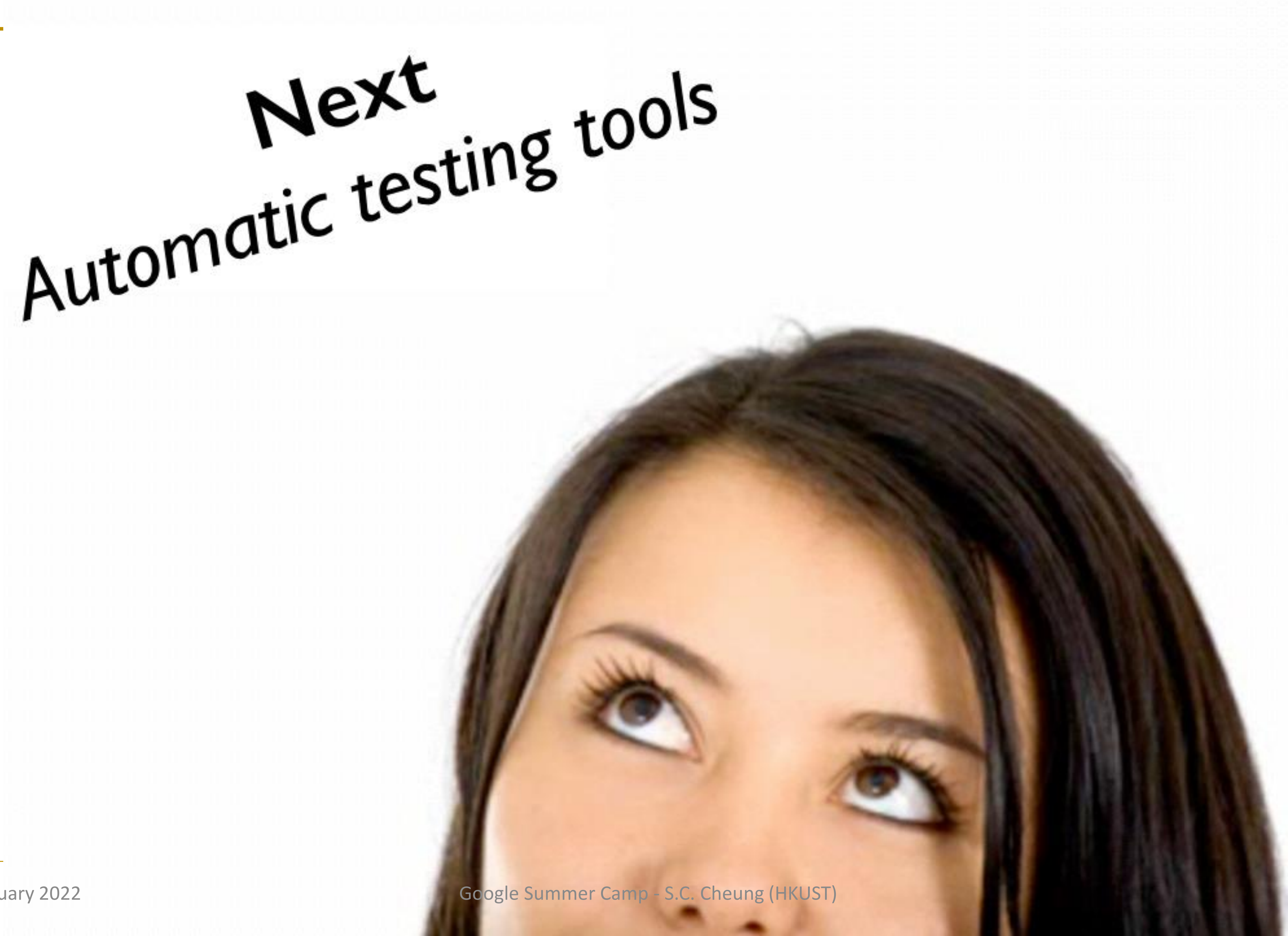
inputs of x and y?

Solution to $[C]$ $\neg P_i$

Adapt it to solve nonlinear constraints or those that involve external API calls

Next

Automatic testing tools



Evosuite

With Dynamic Symbolic Execution Support

Transfer Test Inputs to JUnit Tests

```
public static boolean compare(int a, int b) {
 if (a >= b) {
  return true;
 }
 else {
  return false;
 }
}
```

Transfer Test Inputs to JUnit Tests

```java
public static boolean compare(int a, int b) {
  if (a >= b) {
    return true;
  }
  else {
    return false;
  }
}
```

```java
@Test(timeout = 4000)
public void test0()  throws Throwable  {
    boolean boolean0 = SimpleProgram.compare(1, 0);
    assertTrue(boolean0);
}
@Test(timeout = 4000)
public void test1()  throws Throwable  {
    boolean boolean0 = SimpleProgram.compare(0, 0);
    assertTrue(boolean0);
}
@Test(timeout = 4000)
public void test2()  throws Throwable  {
    boolean boolean0 = SimpleProgram.compare((-1106), 0);
    assertFalse(boolean0);
}
```

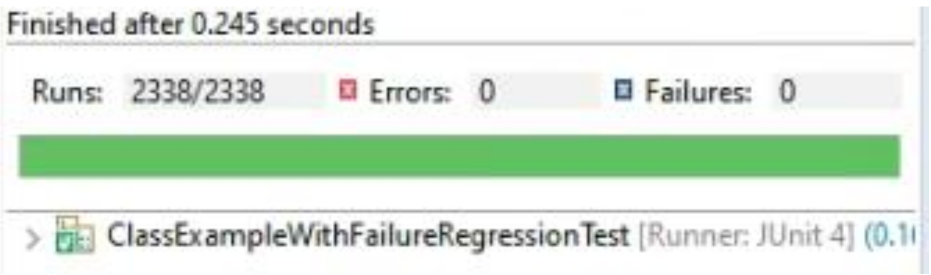
Evosuite

```
public class ClassExampleWithFailure {
    public static int foo(int x, int y) {
        int z = sq(x);
        if (y > 20 && z == 144)
            assert(false);
        return y*z;
    }
    …
}
```

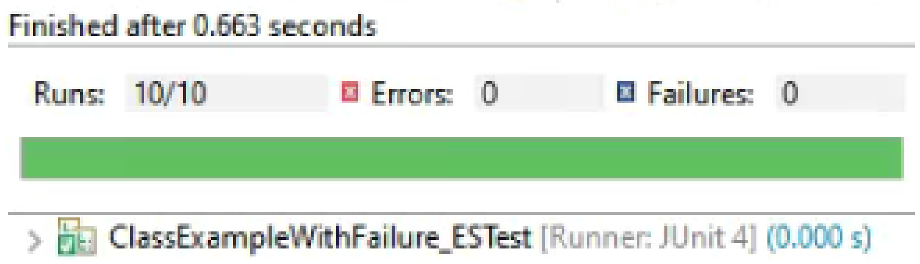
Evosuite

```
public class ClassExampleWithFailure {
        public static int foo(int x, int y) {
                int z = sq(x);
                if (y > 20 && z == 144)
                        assert(false);
                return y*z;
        }
        …
}
```

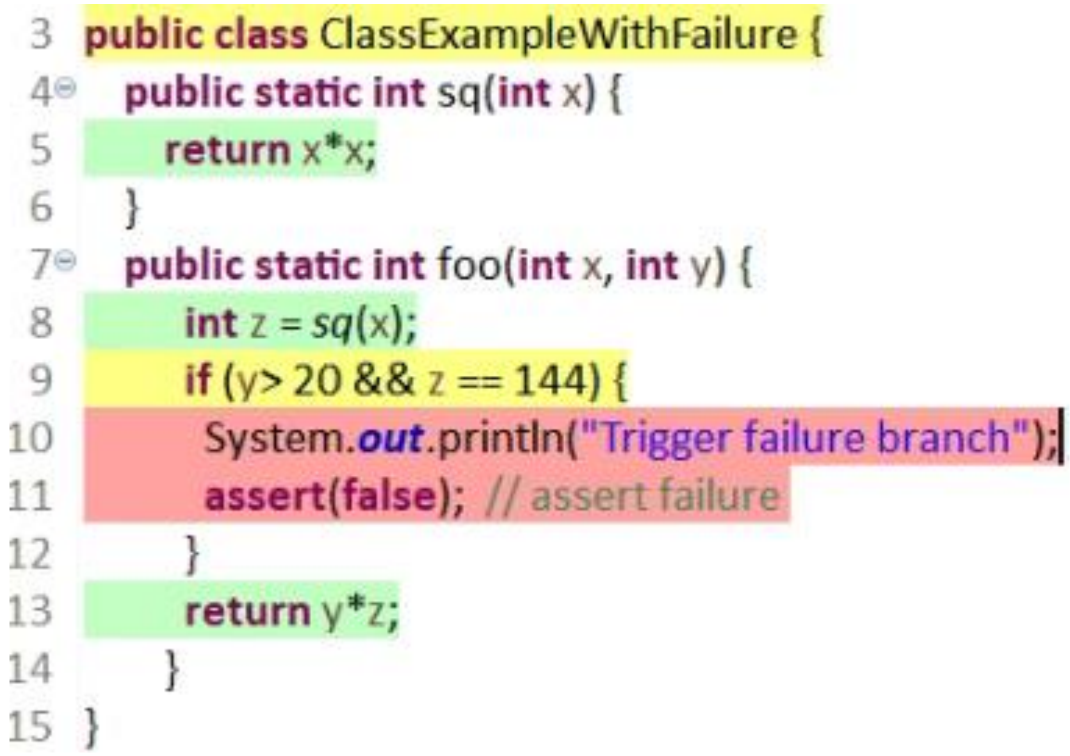
```
@Test(timeout = 4000)
 public void test6()  throws Throwable  {
    try {
     ClassExampleWithFailure.foo(12, 51);
    } catch(AssertionError e) {
      fail("Expecting exception: AssertionError");
    } // …
 }

 @Test(timeout = 4000)
 public void test7()  throws Throwable  {
    int int0 = ClassExampleWithFailure.foo((-1158), 0);
    assertEquals(0, int0);
 }
```



```
Finished after 0.245 seconds
Runs: 2338/2338    Errors: 0    Failures: 0
> ClassExampleWithFailureRegressionTest [Runner: JUnit 4] (0.1
```

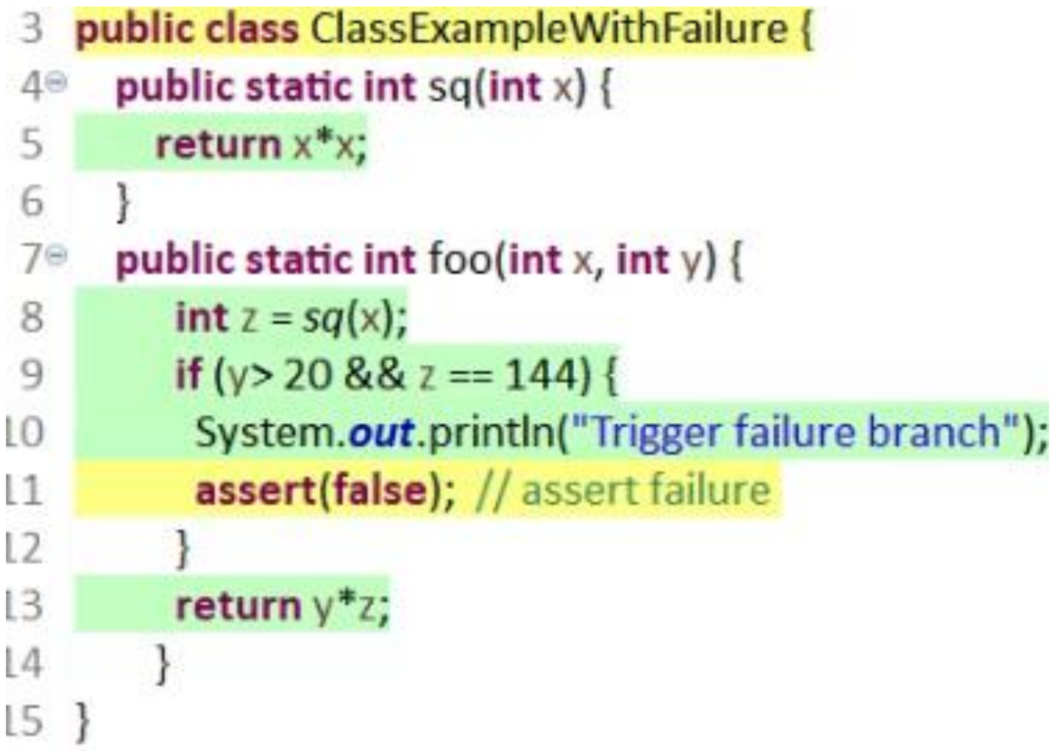


```
Finished after 0.663 seconds
Runs: 10/10    Errors: 0    Failures: 0
> ClassExampleWithFailure_ESTest [Runner: JUnit 4] (0.000 s)
```



```
public class ClassExampleWithFailure {
  public static int sq(int x) {
    return x*x;
  }
  public static int foo(int x, int y) {
      int z = sq(x);
      if (y> 20 && z == 144) {
       System.out.println("Trigger failure branch");
       assert(false);  // assert failure
      }
    return y*z;
    }
}
```

Coverage by Randoop Generated Tests



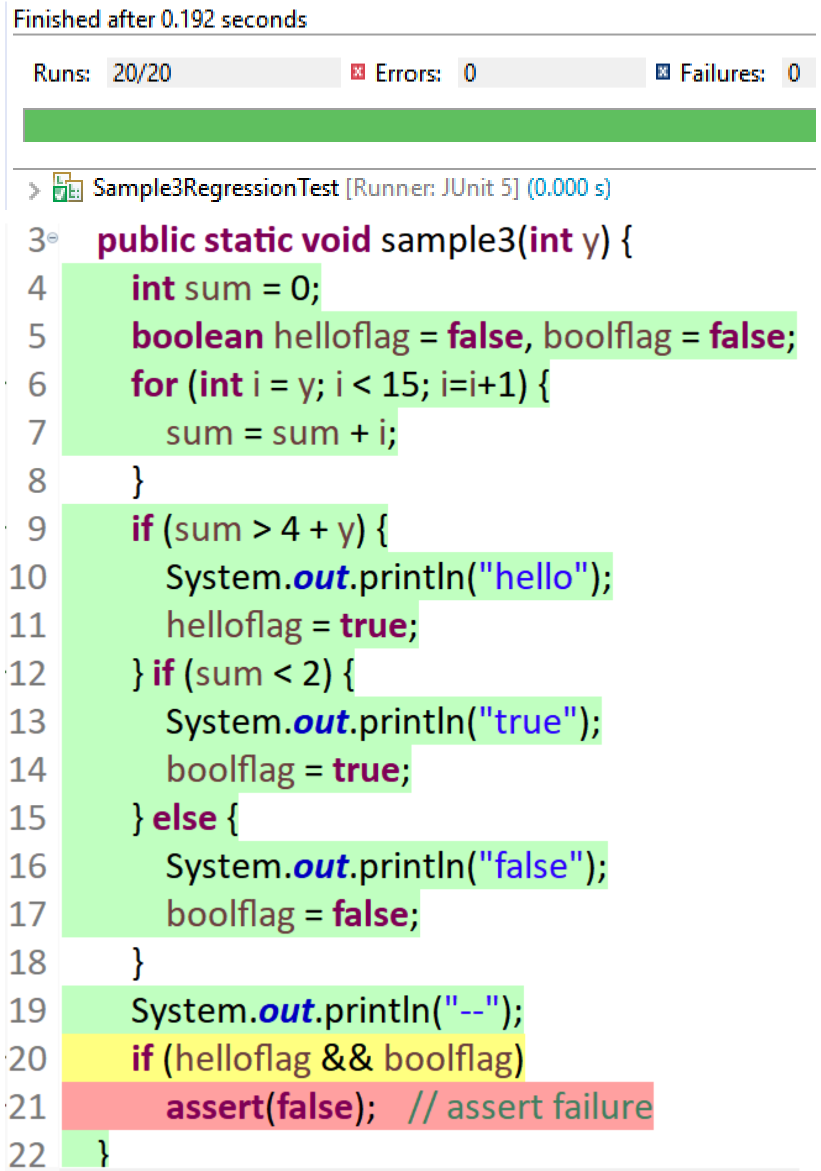
```
public class ClassExampleWithFailure {
  public static int sq(int x) {
    return x*x;
  }
  public static int foo(int x, int y) {
      int z = sq(x);
      if (y> 20 && z == 144) {
       System.out.println("Trigger failure branch");
       assert(false);  // assert failure
      }
    return y*z;
    }
}
```

Coverage by Evosuite Generated Tests

In-Class Exercise 1

Evosuite vs Randoop?

```
public static void sample3(int y) {
    int sum = 0;
    for (int i = y; i < 15; i=i+1) {
        sum = sum + i;
    }
    if (sum > 4 + y)
        System.out.println("hello");
    if (sum < 2) {
        System.out.println("true");
    } else {
        System.out.println("false");
    }
}
```



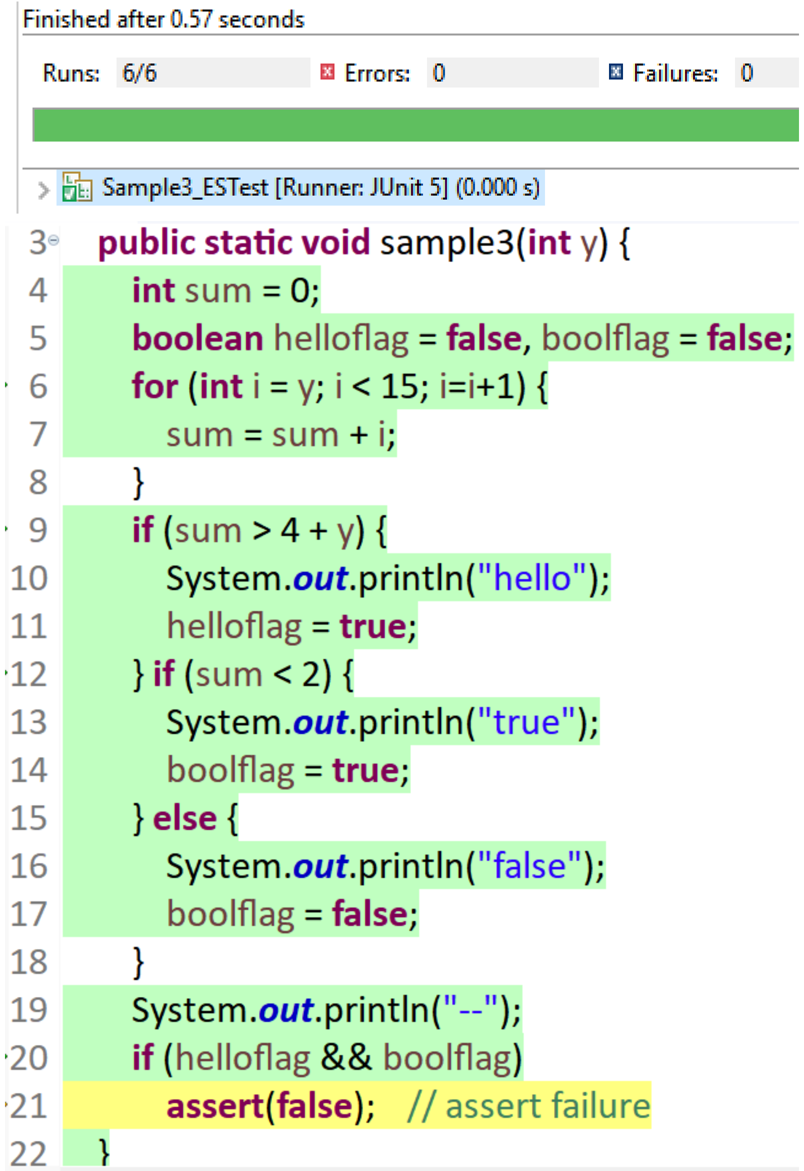
Finished after 0.192 seconds

Runs: 20/20 ☒ Errors: 0 ☒ Failures: 0

> Sample3RegressionTest [Runner: JUnit 5] (0.000 s)

```java
public static void sample3(int y) {
    int sum = 0;
    boolean helloflag = false, boolflag = false;
    for (int i = y; i < 15; i=i+1) {
        sum = sum + i;
    }
    if (sum > 4 + y) {
        System.out.println("hello");
        helloflag = true;
    } if (sum < 2) {
        System.out.println("true");
        boolflag = true;
    } else {
        System.out.println("false");
        boolflag = false;
    }
    System.out.println("--");
    if (helloflag && boolflag)
        assert(false);   // assert failure
}
```

Coverage by Randoop Generated Tests



Finished after 0.57 seconds

Runs: 6/6 ☒ Errors: 0 ☒ Failures: 0

> Sample3_ESTest [Runner: JUnit 5] (0.000 s)

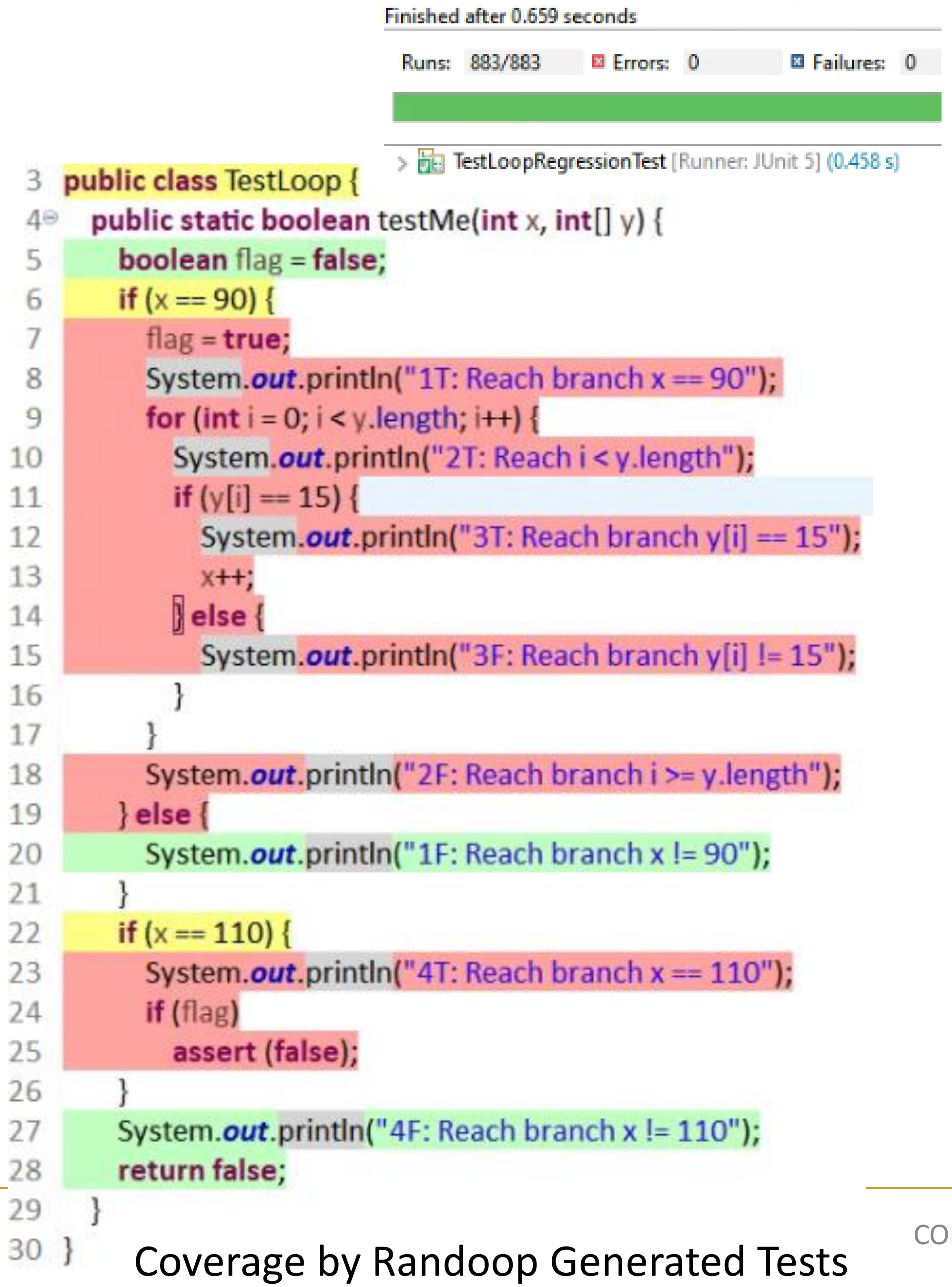
```java
public static void sample3(int y) {
    int sum = 0;
    boolean helloflag = false, boolflag = false;
    for (int i = y; i < 15; i=i+1) {
        sum = sum + i;
    }
    if (sum > 4 + y) {
        System.out.println("hello");
        helloflag = true;
    } if (sum < 2) {
        System.out.println("true");
        boolflag = true;
    } else {
        System.out.println("false");
        boolflag = false;
    }
    System.out.println("--");
    if (helloflag && boolflag)
        assert(false);   // assert failure
}
```

Coverage by Evosuite Generated Tests

In-Class Exercise 2 - TestLoop

Evosuite vs Randoop?

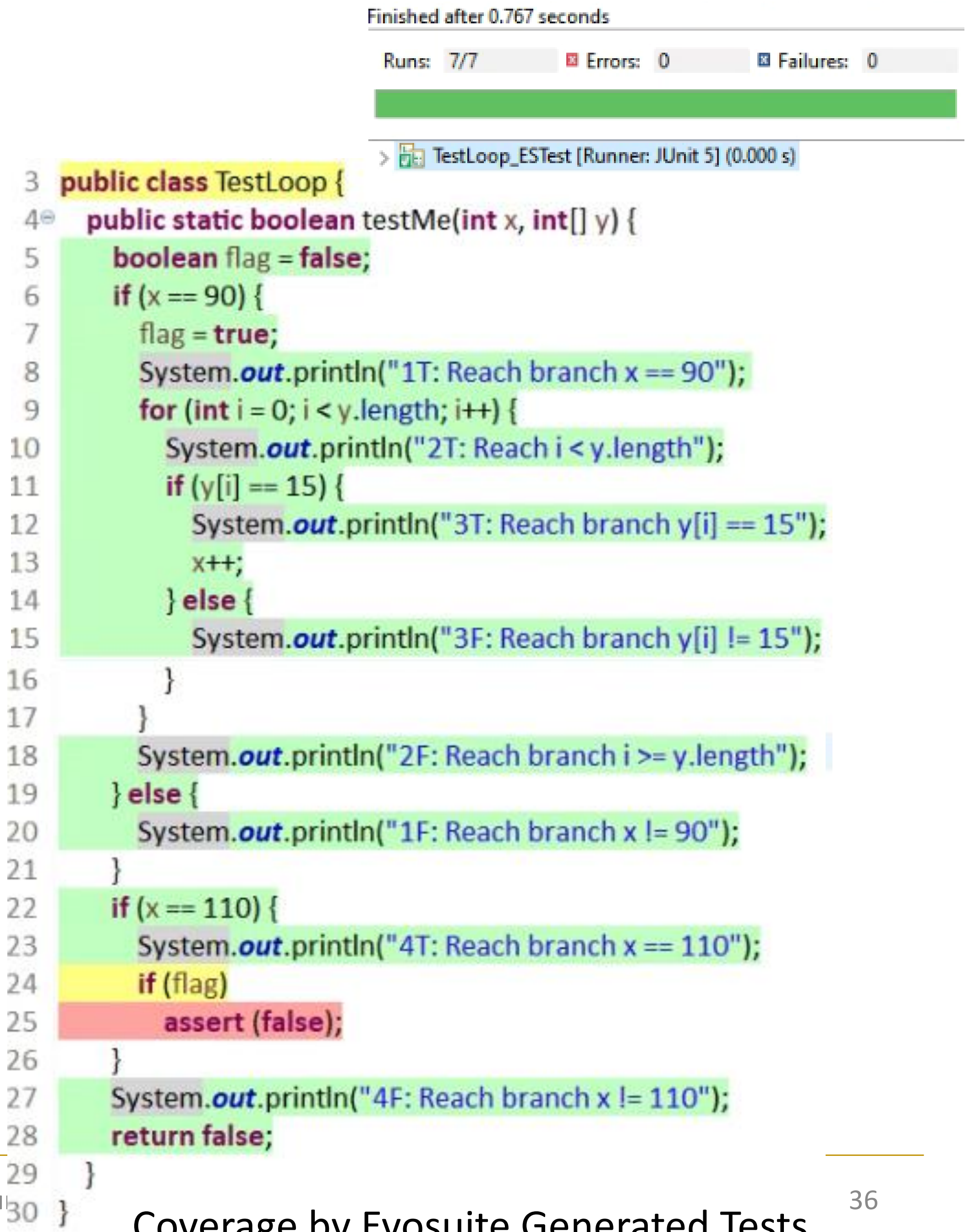
```
public static boolean testMe(int x, int[] y) {
        boolean flag = false;
        if (x == 90) {
                flag = true;
                for (int i=0; i<y.length; i++) {
                        if (y[i] == 15) {  x++; } else { }
                }
        } else { }
        if (x == 110) {
                if (flag)
                        assert(false);
        }
        return false;
}
```



```
Finished after 0.659 seconds
Runs: 883/883   Errors: 0   Failures: 0
> TestLoopRegressionTest [Runner: JUnit 5] (0.458 s)
```

```java
public class TestLoop {
  public static boolean testMe(int x, int[] y) {
    boolean flag = false;
    if (x == 90) {
      flag = true;
      System.out.println("1T: Reach branch x == 90");
      for (int i = 0; i < y.length; i++) {
        System.out.println("2T: Reach i < y.length");
        if (y[i] == 15) {
          System.out.println("3T: Reach branch y[i] == 15");
          x++;
        } else {
          System.out.println("3F: Reach branch y[i] != 15");
        }
      }
      System.out.println("2F: Reach branch i >= y.length");
    } else {
      System.out.println("1F: Reach branch x != 90");
    }
    if (x == 110) {
      System.out.println("4T: Reach branch x == 110");
      if (flag)
        assert (false);
    }
    System.out.println("4F: Reach branch x != 110");
    return false;
  }
}
```

Coverage by Randoop Generated Tests



```
Finished after 0.767 seconds
Runs: 7/7   Errors: 0   Failures: 0
> TestLoop_ESTest [Runner: JUnit 5] (0.000 s)
```

```java
public class TestLoop {
  public static boolean testMe(int x, int[] y) {
    boolean flag = false;
    if (x == 90) {
      flag = true;
      System.out.println("1T: Reach branch x == 90");
      for (int i = 0; i < y.length; i++) {
        System.out.println("2T: Reach i < y.length");
        if (y[i] == 15) {
          System.out.println("3T: Reach branch y[i] == 15");
          x++;
        } else {
          System.out.println("3F: Reach branch y[i] != 15");
        }
      }
      System.out.println("2F: Reach branch i >= y.length");
    } else {
      System.out.println("1F: Reach branch x != 90");
    }
    if (x == 110) {
      System.out.println("4T: Reach branch x == 110");
      if (flag)
        assert (false);
    }
    System.out.println("4F: Reach branch x != 110");
    return false;
  }
}
```

Coverage by Evosuite Generated Tests