

Structural (White-box) Testing

(Chapter 6)

Outline

- White-box vs black-box testing
- How can testing be done systematically if we have access to the source code ?

Structural vs Functional Testing

- Testing in the small = we test individual modules (in the large = whole-system test)
- WHITE BOX (structural) testing
 - partitioning criteria based on module's internal code
 - tests *what the program does*
 - derives test cases from program code
- BLACK BOX (functional) testing
 - partitioning criteria based on the module's specification
 - tests *what the program is supposed to do*

Structural Coverage Testing

- (In)adequacy criteria
 - If significant parts of program structure are not tested, testing is inadequate
- Control flow coverage criteria
 - Statement coverage
 - Edge coverage
 - Condition coverage
 - Path coverage

Statement-coverage criterion

- Select a test set T such that every elementary statement in P is executed at least once by some d in T
 - an input d executes many statements
 - try to minimize the number of test cases still while preserving the desired coverage

Example

```
read (x); read (y);
if x > 0 then
    write ("1");
else
    write ("2");
end if;
if y > 0 then
    write ("3");
else
    write ("4");
end if;
```

D = ?
complete coverage partitions ?
are the sets minimal ?

$D = \mathbb{Z} \times \mathbb{Z}$

Partition = $\{x > 0 \wedge y > 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\} \cup$
 $\{x \leq 0 \wedge y > 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\} \cup$
 $\{x > 0 \wedge y \leq 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\} \cup$
 $\{x \leq 0 \wedge y \leq 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\}$

Minimal for complete statement coverage

$\{x > 0 \wedge y > 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\} \cup$
 $\{x \leq 0 \wedge y \leq 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\}$

or, alternatively,

$\{x > 0 \wedge y \leq 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\} \cup$
 $\{x \leq 0 \wedge y > 0 \mid x \in \mathbb{Z}, y \in \mathbb{Z}\}$

Weakness of statement-coverage

```
if x < 0 then
  x := -x;
end if;
z := x;
```

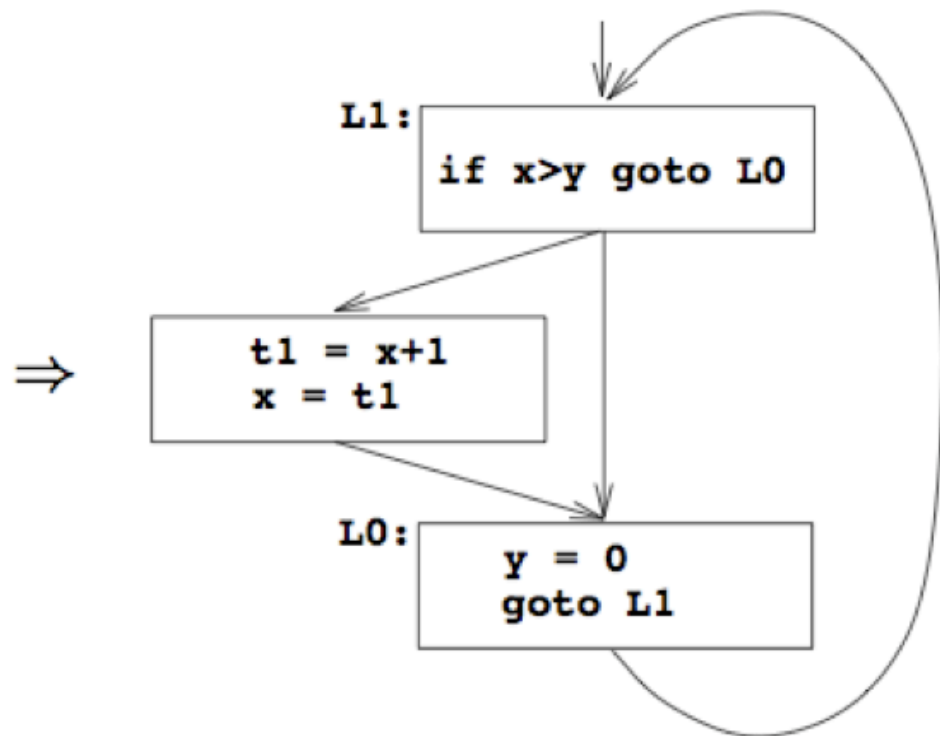
{<x=-3>} covers all statements

Problem: it does not exercise the case when x is positive and the then branch is not entered

Control flow graphs

- edges represent statements
- nodes at the ends of an edge represent *entry into/exit from* a statement

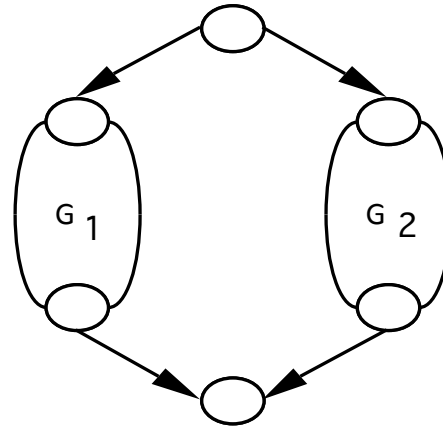
```
L1:  if x > y goto L0
      t1 = x+1
      x = t1
L0:  y = 0
      goto L1
```



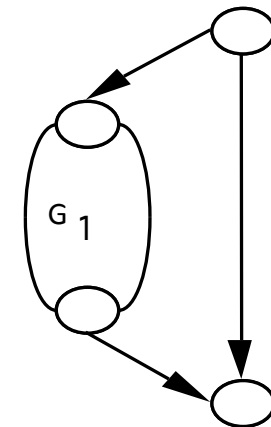
Control graph construction rules



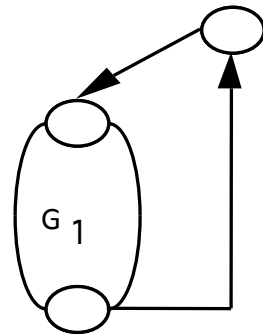
I/O, assignment,
or procedure call



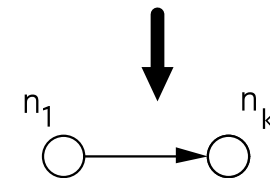
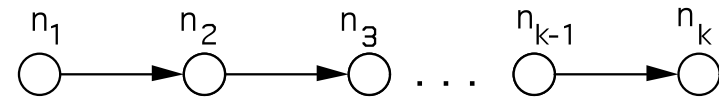
if-then-else



if-then



while loop



collapse sequential
statements

Example: CFG for Euclid's algorithm

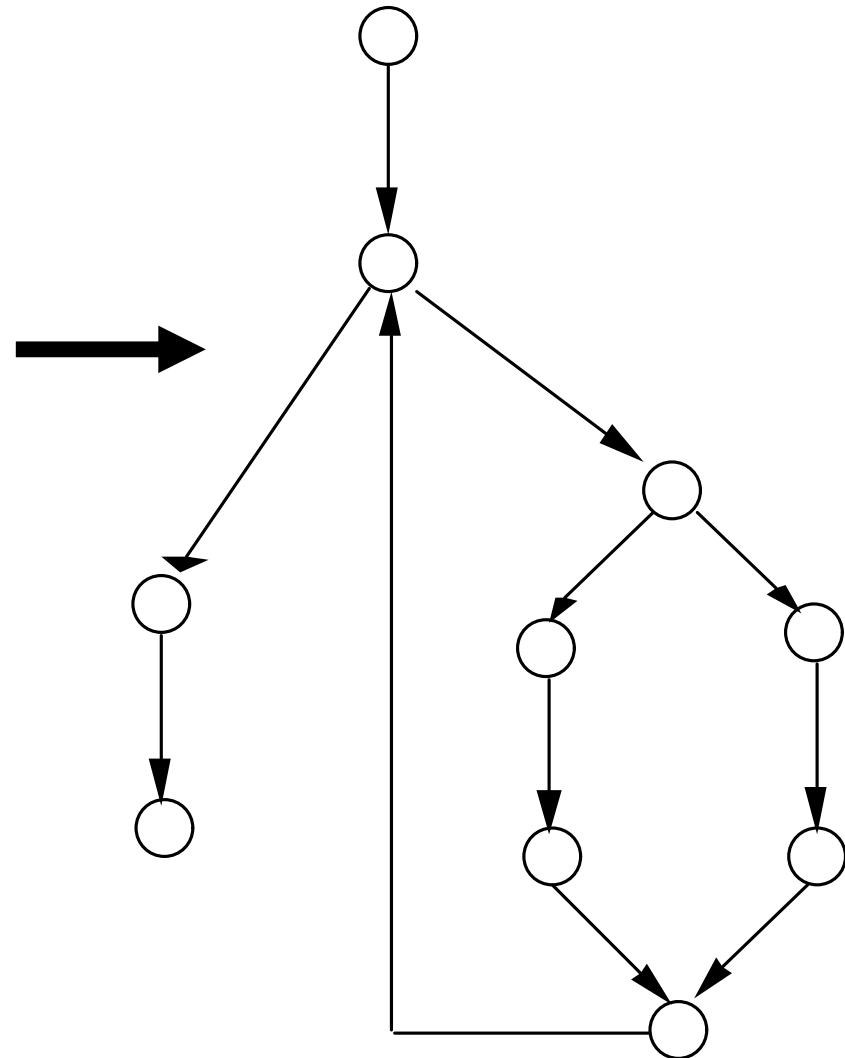
```
begin
  read (x); read (y);
  while x ≠ y {
    if x > y then
      x := x - y;
    else
      y := y - x;
    end if;
  }
  gcd := x;
end;
```

Edge-coverage criterion

- Compute control flow graph
- Select a test set T such that every edge (branch) of the control flow is exercised at least once by some d in T

Example: Euclid's algorithm

```
begin
  read (x); read (y);
  while x ≠ y {
    if x > y then
      x := x - y;
    else
      y := y - x;
    end if;
  }
  gcd := x;
end;
```



$T = \{?\}$

Example: Euclid's algorithm

```
begin
  read (x); read (y);
  while x ≠ y {
    if x > y then
      x := x - y;
    else
      y := y - x;
    end if;
  }
  gcd := x;
end;
```

D = ?

**complete coverage partitions ?
are the sets minimal ?**

$D = \mathbb{N} \times \mathbb{N}$

Minimal for complete edge coverage

$\{x > y \mid x \in \mathbb{N}, y \in \mathbb{N}\} \cup$
 $\{x \leq y \mid x \in \mathbb{N}, y \in \mathbb{N}\}$

Weakness of edge-coverage

```
found = false; counter = 1;
while (not found) and counter < number_of_items {
    if table (counter) == desired_element {
        found = true;
    }
    counter = counter + 1;
} // end while

if found
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
```

Test cases:

(1) empty table

(2) number_of_items = 3 and desired_element = second item

Testing provides edge coverage, but does not discover the

error ($<$ instead of \leq)!

"Hidden" control-flow edges

```
if c1 and c2
then
  st;
else
  sf;
end if;
```



```
if c1
then
  if c2
  then
    st;
  else
    sf;
  end if;
else
  sf;
end if;
```

Condition-coverage criterion

- Select a test set **T** such that every edge of **P**'s control flow is traversed and all possible values of the constituents of compound conditions are exercised at least once
- Finer than edge coverage
- Example: for linear search problem, make sure we test the case when we exit the loop via
(not found) and (counter > number_of_items)

Weakness of condition-coverage

```
if x ≠ 0 then
  y := 5;
else
  z := z - x;
end if;
if z > 1 then
  z := z / x;
else
  z := 0;
end if;
```

$\{\langle x = 0, z = 1 \rangle, \langle x = 1, z = 3 \rangle\}$
causes the execution of all edges,
but fails to expose the risk of a
division by zero

Path-coverage criterion

- Select a test set T which traverses all paths from the initial to the final node of P 's control flow
- Finer than previous kinds of coverage
- Problem: number of paths may be too large, or even infinite (see while loops)
 - additional constraints must be provided

Example: complete path-coverage

```
if x ≠ 0 then
  y := 5;
else
  z := z - x;
end if;
if z > 1 then
  z := z / x;
else
  z := 0;
end if;
```

$D = \mathbb{Z} \times \mathbb{Z}$

Complete path coverage

$$\begin{aligned} & \{x \neq 0 \wedge z > 1 \mid x \in \mathbb{Z}, z \in \mathbb{Z}\} \cup \\ & \{x \neq 0 \wedge z \leq 1 \mid x \in \mathbb{Z}, z \in \mathbb{Z}\} \cup \\ & \{x = 0 \wedge z > 1 \mid x \in \mathbb{Z}, z \in \mathbb{Z}\} \cup \\ & \{x = 0 \wedge z \leq 1 \mid x \in \mathbb{Z}, z \in \mathbb{Z}\} \end{aligned}$$

E.g.,

$$\begin{aligned} & \{ \langle x = 1, z = 2 \rangle, \\ & \quad \langle x = 1, z = 0 \rangle, \\ & \quad \langle x = 0, z = 2 \rangle, \\ & \quad \langle x = 0, z = 0 \rangle \} \end{aligned}$$

The infeasibility problem

- Syntactically indicated behaviors (statements, edges, etc.) are often impossible
 - unreachable code, infeasible edges, paths, etc.

```
if x < 0
then
    if x >= 0
    then
        // never reached
```

The infeasibility problem

- Adequacy criteria may be impossible to satisfy
 - manual justification for omitting each impossible test case
 - adequacy “scores” based on coverage
 - example: 95% statement coverage

Further problems

- What if the code omits the implementation of some part of the specification?
- White box test cases derived from the code will ignore that part of the specification!

Exercise

- Compute D and R
- Find a minimal T that provides *statement*, *edge* and *path* coverage for this program

```
int P(int x, int y) {  
  int ret;  
  
  if x > y then  
    ret := 3;  
  end if;  
  if x > y + 1 then  
    ret := 3;  
  else  
    ret := 2;  
  end if;  
  
  return ret;  
}
```

Reading for next class

Sections 6.3.4.2 -- 6.3.5.3)