

**SE 2AA4, CS 2ME3 (Introduction to Software  
Development)**

**Winter 2017**

# **32 White Box Testing Continued (Ch. 6) DRAFT**

Dr. Spencer Smith

Faculty of Engineering, McMaster University

March 19, 2017



# White Box Testing Continued

- Administrative details
- White box testing
  - ▶ Edge coverage
  - ▶ Condition coverage
  - ▶ Path coverage
- Black Box Testing
  - ▶ Informal Example
  - ▶ Formal using MIS for PointT

# Administrative Details

- Today's slide are partially based on slides by Dr. Wassying
- Assignment 5
  - ▶ Implementation due by midnight March 30
  - ▶ If there is a new specification, it is due by 4pm March 31 (under door of ITB/167 is fine for submission)

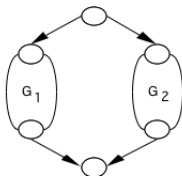
# Edge-Coverage Criterion

- Select a test set  $T$  such that every edge (branch) of the control flow is exercised at least once by some  $d$  in  $T$
- This requires formalizing the concept of the control graph and how to construct it
  - ▶ Edges represent statements
  - ▶ Nodes at the ends of an edge represent entry into the statement and exit

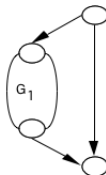
# Control Graph Construction Rules



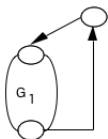
I/O, assignment,  
or procedure call



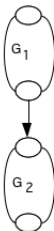
if-then-else



if-then



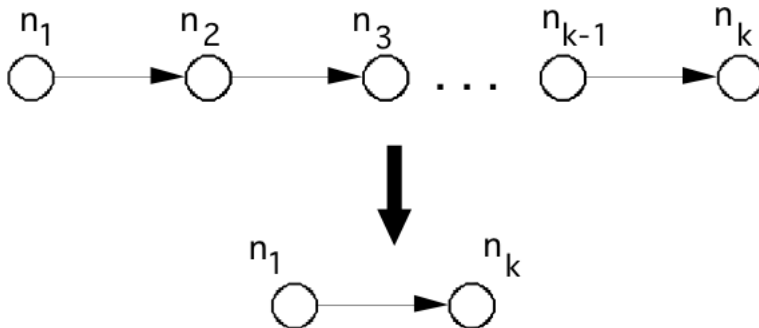
while loop



two sequential  
statements

# Simplification

A sequence of edges can be collapsed into just one edge



# Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

# Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

Do not discover the error ( $<$  instead of  $\leq$ )



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

*// equivalent to*

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

## 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
- This criterion is finer than edge coverage

# Weakness

```
if  $x \neq 0$  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  $T$  that traverses all paths from the initial to the final node of  $P$ 's control flow
- It is finer than the previous kinds of coverage
- However, number of paths may be too large, or even infinite (see while loops)
- Loops
  - ▶ Zero times (or minimum number of times)
  - ▶ Maximum times
  - ▶ Average number of times

# The Infeasibility Problem

- Syntactically indicated behaviours (statements, edges, etc.) are often impossible
- Unreachable code, infeasible edges, paths, etc.
- 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 Problem

- 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!