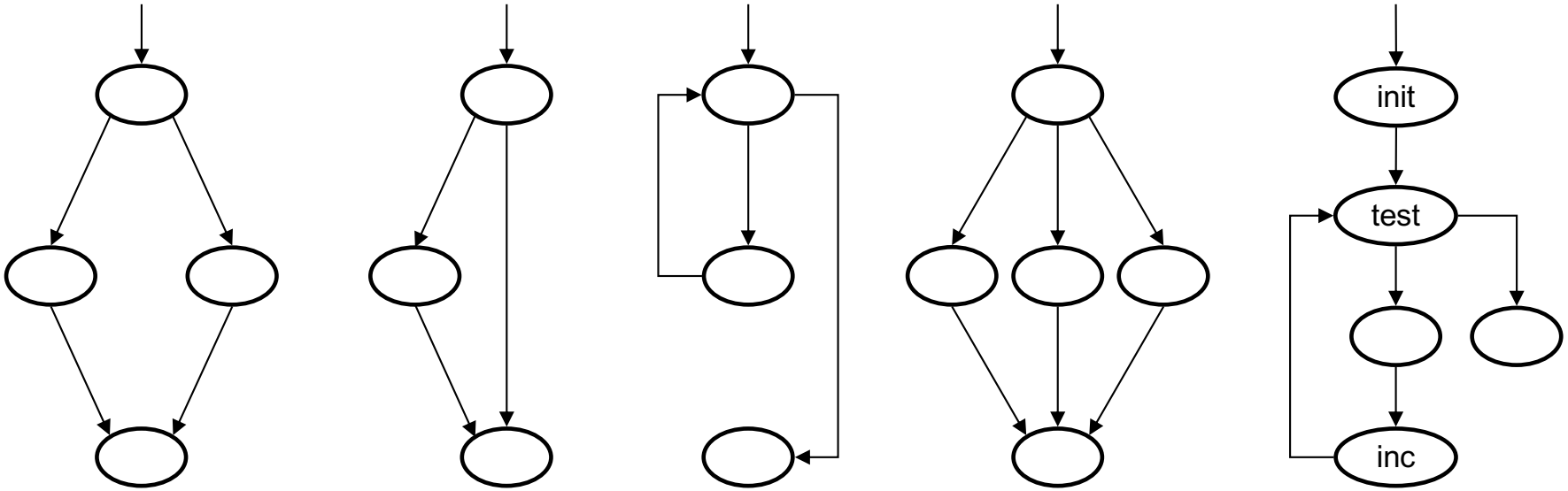
SYSC 4101 / 5105

Graph Criteria—Applications
Structural

Control Flow Graph (CFG)

- Use the program structure, the control flow graph (CFG):
  - "A graph model of a program in which conditional branch instructions and program junctions are represented by nodes and the program segments between such points are represented by links." [Beizer 1990]
- Simplest form:
  - A control flow graph has one node per statement, and an edge from n_1 to n_2 if control can ever flow directly from n_1 to n_2.
- Condensed (preferred) form:
  - A control flow graph has one node per basic block, and an edge from n_1 to n_2 if control can ever flow directly from n_1 to n_2.
  - Basic block: a straight-line piece of code without any control structure
    - When the first line executes, all other lines in the block execute too (unless there is an exception, but this is a different topic)

Basics of Control-Flow Graph Structures



if-then-else

? : operator in Java/C…

if-then

while

switch

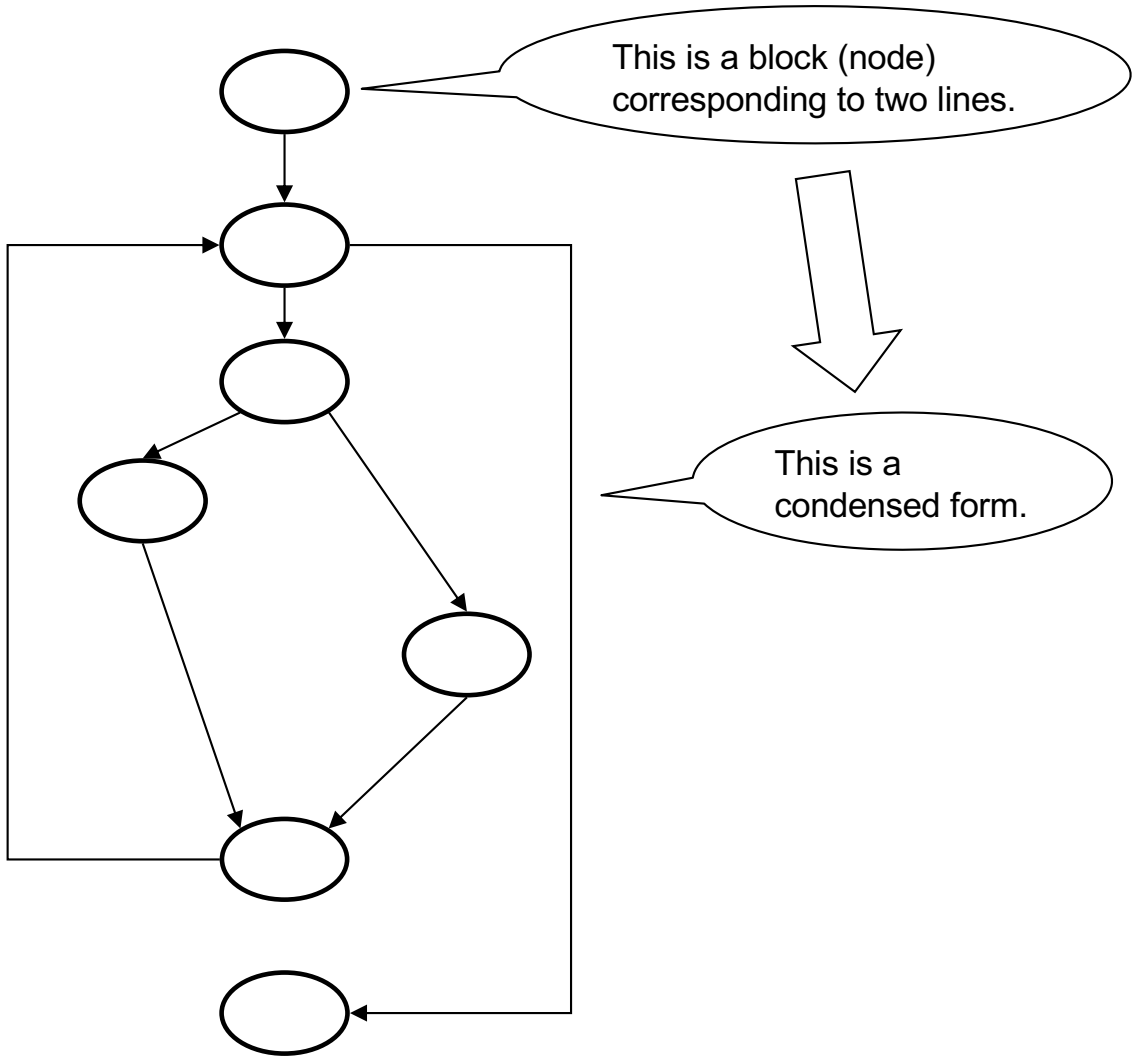
for(init;test;inc)

General convention for control flow: true branch on the left, unless there are layout constraints.

With these principles in mind, you should be able to build the control-flow for other programming constructs (programming language dependent), e.g., goTo, even in the presence of exceptions.

Control Flow Graph (Example)

```
read(x);read(y)

while x ≠ y loop

  if x>y then

        x := x – y;
  else

        y := y – x;


  end if;
end loop;

gcd := x;
```

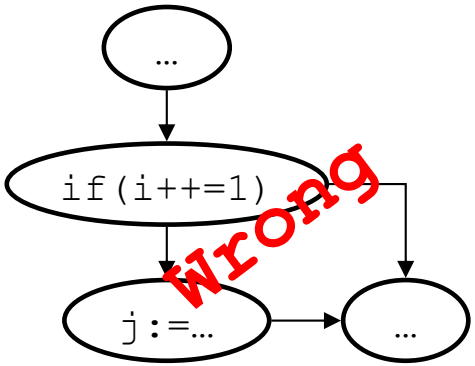


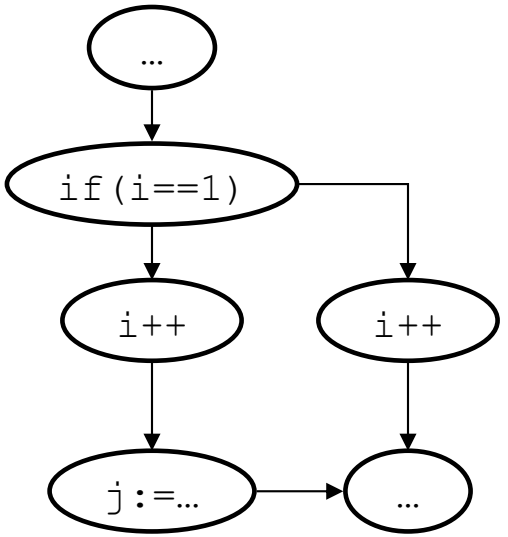
Transformation Issue

From source code to a control flow graph—Issue about branching

- In a CFG, nodes corresponding to branching (if, while, …) should **not** contain any assignment (i.e., definition).
- This is paramount for the identification of data-flow information.
- E.g., …

```
if (i++==1) {
        j := …
}
…
```





Simplest form of CFG

Data Flow Information

- Recall:
  - A def is a location in the program where a value for a variable is stored in memory.
  - A use is a location in the program where a variable's value is accessed.
- A def may occur for variable x in the following situations:
  - x appears on the left side of an assignment (i=1, obj=null),
  - x is an actual parameter in a call and its value is changed within the called method (func_call(m) where m is passed by reference),
  - x is a formal parameter of a method, an implicit def when the method begins execution (void func(m) { // m is defined at function entry … }),
- A use may occur for variable x in the following situations:
  - x appears on the right side of an assignment statement (…=x+y*2),
  - x appears in a conditional test (if x==10),
  - x is an actual parameter to a method (func_call(m)),
  - x is an output of the program (return b;),
  - x is an output of a method in a return statement (return b;) or returned as a parameter (func_call(m) where m is passed by reference).

Data Flow Information (advanced)

What about complex structures?

- Variable v is an array
  - Solution 1: consider the array as a whole
    - if v is used in some way (e.g., access to its length, access to an element), v is considered used,
      - p = myArray[i] +2 use of variable myArray
      - l = myArray.length use of variable myArray
    - if v is modified in some way (e.g., change of an element value, addition, removal of element), v is considered defined.
      - myArray[i] = 10 definition of myArray
      - myArray = new int[10] definition of myArray
  - Solution 2: consider the length and elements as separate variables
      - myArray[i] = 10 definition of myArray[i] only but not of myArray
      - myArray = new int[10] definition of myArray and all its 10 elements
      - p = myArray[i] +2 use of myArray[i] only
      - l = myArray.length use of variable myArray's length
  - Solution 2 is more precise (finer-grained) but data flow more difficult to identify

Data Flow Information (advanced)

What about objects?

- Variable v is an object
  - Solution 1
    - If the reference to the object is used (resp. modified) in some way, or its attribute values are used (resp. modified) in some way, the object is considered used (resp. defined),
  - Solution 2
    - Reference to object and individual attributes considered separately:
      - If the reference to v is used (resp. defined), this is a use (resp. definition) of v, but not its attributes.
      - If an attribute a of the object is used (resp. modified) in some way, a is considered used (resp. defined).
        - » Other attributes are not used/modified
        - » Reference to object is not used/modified
    - Rationale: data flow related to state-based behavior.

Data Flow Information

- The order of definitions and uses in a basic block matter!
- Results:
  - Impacts the identification of du-paths
  - Some uses may be impossible to reach with a definition clear path
  - Some definitions may hide others
    - Some definitions may not be part in any DU-path
- i++; $\approx$ i=i+1;
  - The use happens before the definition
- Consider the following basic block:

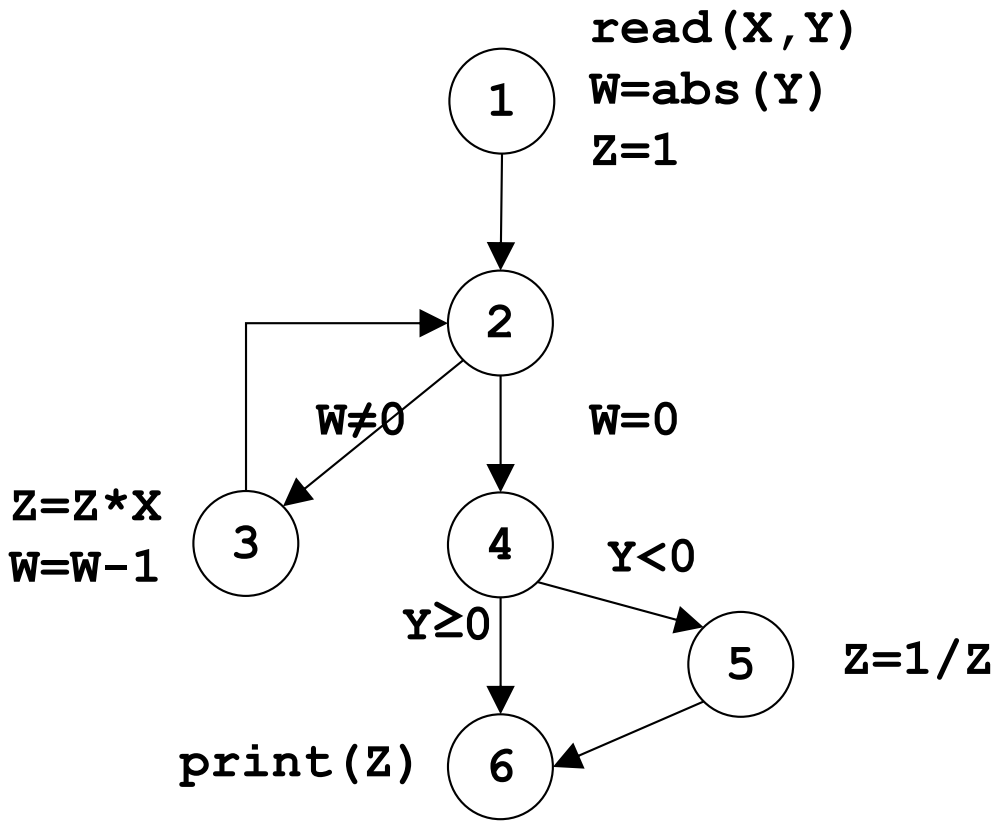
  y=z;

  x=y+2;
  - The second use of y is a local use
  - It is impossible for a def in a previous basic block to reach that use
  - The def in the first statement always reaches the use in the second

Example: Power Function

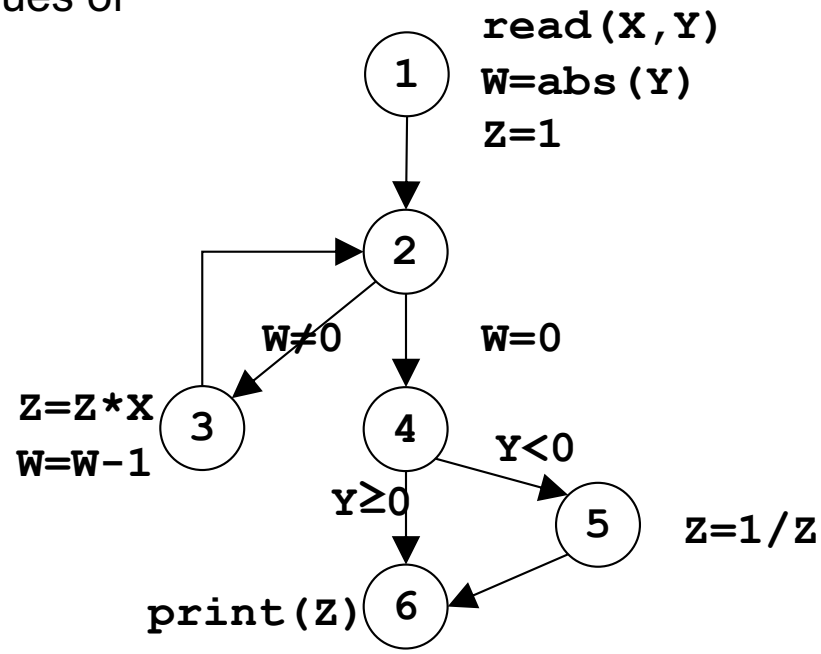
Program computing Z=X^Y

```
BEGIN
  read (X, Y) ;
  W = abs(Y) ;
  Z = 1 ;
  WHILE (W <> 0) DO
    Z = Z * X ;
    W = W - 1 ;
  END
  IF (Y < 0) THEN
    Z = 1 / Z ;
  END
  print (Z) ;
END
```



Example: Control-Flow Testing

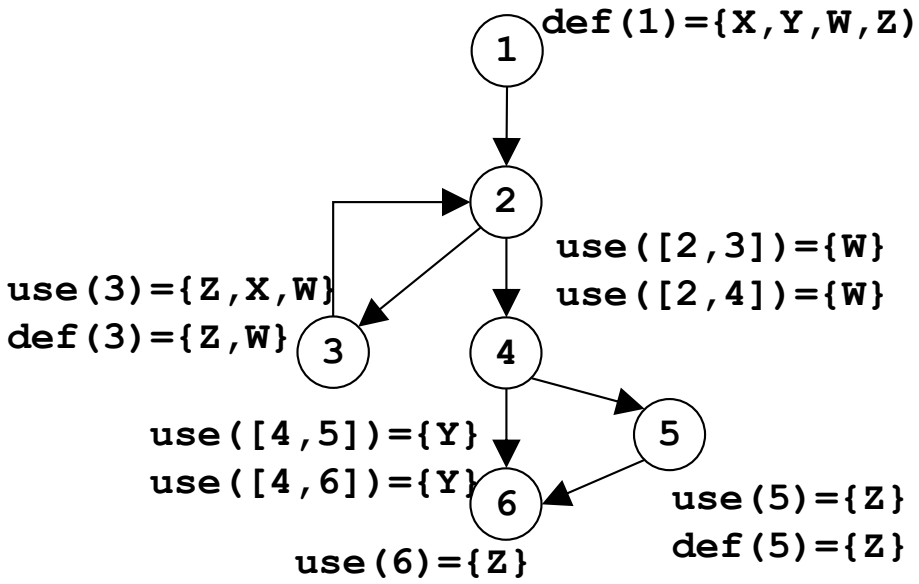
- All paths
  - Infeasible path
    - [1,2,4,5,6]
  - Infinite number of paths :
    - As many ways to iterate 2,(3,2)* as values of abs(Y)
- All branches (a.k.a. all-edges)
  - Two test cases are enough
    - Y<0 : [1,2,(3,2)+,4,5,6]
    - Y≥0 : [1,2,(3,2)*,4,6]
- All statements (a.k.a. all-nodes)
  - One test case is enough
    - Y<0 : [1,2,(3,2)+,4,5,6]



Example: Data-Flow Testing

- du(1,x)={[1,2,3]}
- du(1,y)={~~[1,2,4,5]~~, [1,2,4,6]}
- du(1,w)={[1,2,3], [1,2,4], [1,2,3]}
  - [1,2,3] appears twice to account for the use in edges [2,3] and [2,4], and in node 3.
- du(1,z)={[1,2,3], [1,2,4,5], [1,2,4,6]}
- du(3,w)={[3,2,3], [3,2,3], [3,2,4]}
  - Same comment as above, but for [3,2,3].
- du(3,z)={[3,2,3], [3,2,4,5], [3,2,4,6]}
- du(5,z)={[5,6]}

infeasible



Deriving Input Values

- Not all statements are usually reachable in real world programs
- It is not always possible to decide automatically if a statement is reachable and what is the percentage of reachable statements
- When one does not exercise 100% of test requirements, it is therefore difficult to determine the reason
  - Because the test set is not sufficient?
  - Because there are unreachable elements?
  - Because of problems in the test scaffolding?
- Tools are needed to support this activity – and there exist good tools

Deriving Input Values

- To find test inputs that will execute an arbitrary statement Q within a program source, the tester must work backward from Q through the program's flow of control to an input statement

- For simple programs, this amounts to solving a set of simultaneous inequalities in the input variables of the program, each inequality describing the proper path through one conditional

- Conditionals may be expressed in local variable values derived from the inputs and those must figure in the inequalities as well

- Similar problem as symbolic execution.

Example

```
…
if (x > 3) {
  z = x+y;
  y+= x;
  if (2*z == y) {
    /* statement to be covered */
…
```

Inequalities:

- $x > 3$
- $2(x+y) = x+y$

  $\Leftrightarrow \begin{cases} x = -y \\ x > 3 \end{cases}$

Solution:

$x = 4$

$y = -4$

Problems

- The presence of loops and recursion in the code makes it impossible to write and solve the inequalities in general
- Each pass through a loop may alter the values of variables that appear in a following conditional and the number of passes cannot be determined by static analysis
  - At least this is extremely difficult to do automatically

Tools

- Test generation
  - Telcordia's AETG (no longer available online)
  - Parasoft Jtest, and C++Test
  - Frama-C Ltest (C)
  - Evosuite (Java)
- Code coverage
  - Telcordia's tool (no longer available online)
  - McCabe Test and Coverage Server
  - IPL Cantata and Cantata++
  - Rational Purify/Coverage
  - Rational TestRealTime
  - Open Source Code Coverage Tools in Java: http://java-source.net/open-source/code-coverage
    - They tend to support extremely simple and therefore useless criteria.