UTD

Mathematical Preliminaries

Dr. Mark C. Paulk

SE 4367 – Software Testing, Verification, Validation, and Quality Assurance

Topics: Software Testing

<u>Part I: Preliminaries</u>

1. Software Testing

2. Mathematical

- **Predicates and Boolean Expressions**
- **Control Flow Graph**
- **Execution History**
- **Dominators and Post-Dominators**
- **Program Dependence Graph**
- **Strings, Languages, and Regular Expressions**
- **Tools**

Mathur Example
Boiler Shutdown Conditions

a) The water level in the boiler is below X lbs.
b) The water level in the boiler is above Y lbs.
c) A water pump has failed.
d) A pump monitor has failed.
e) Steam meter has failed.

The boiler is to be shut down when

- **(a or b) or**
- **(c or d) and e**

(c or d) is considered degraded mode

A Boolean Expression

The following Boolean expression E when true must force a boiler shutdown

E = a + b + (c + d)e

where the + sign indicates "OR" and a multiplication indicates "AND"

The goal of predicate-based test generation is to generate tests from a predicate p that guarantee the detection of any error that belongs to a class of errors in the coding of p.

Mathur Printer Example

Consider the requirement "*if the printer is ON and has paper then send document to printer*."

Consists of a condition part and an action part

The following predicate represents the condition part of the statement.

p_r: (printerstatus = ON) $\wedge$ (printertray != empty)

a) printerstatus = ON
b) printertray != empty
E = ab

Test Generation from Predicates

Generating tests to detect faults in the coding of conditions

Condition is represented formally as a predicate

condition + action

- **predicate is the condition part of the statement**

Operators

relop (relational operator) in {<, >, ≤, ≥, =, ≠}

bop (Boolean operator) in {$\wedge$, $\vee$, $\cancel{\vee}$, $\neg$**}**

If clear from context, leave out $\wedge$

- **write** $\vee$ **as +**
- **write** $\neg$ **as ~, !,** $\overline{a}$

A predicate can be converted to a Boolean expression by replacing each relational expression with a distinct Boolean variable.

Examples of Boolean Notation

a $\wedge$ b $\vee$!c
a AND b OR NOT c
ab + !c

(a $\wedge$ b $\wedge$ c) $\vee$ (!d $\wedge$!e $\wedge$!f)
(a AND b AND c) OR ((NOT d) AND (NOT e) AND (NOT f))
abc + !d!e!f

Note the priorities of operators

- **parentheses**
- **NOT**
- **AND**
- **OR**

Simple vs Compound Predicates

Simple predicate

- **a Boolean variable or a relational expression**
- **$(x<0)$**

Compound predicate

- **join one or more simple predicates using bop**
- **(gender == “female” $\wedge$ age > 65)**

Boolean expression

- **one or more Boolean variables joined by bop**
- **$(a \wedge b \vee !c)$**

Singular

A Boolean expression is singular if each variable in the expression occurs only once.

$E = e_1$ bop e_2 bop … bop e_k

e_i and e_j are mutually singular if they do not share any variable

e_i is a singular component of E iff e_i is singular and is mutually singular with every other component of E

e_i is nonsingular iff it is nonsingular by itself and mutually singular with the remaining components of E

DNF and CNF

A Boolean expression is in disjunctive normal form (DNF) if it is represented as a sum of product terms

pq + ~rs

A Boolean expression is in conjunctive normal form (CNF) if it is represented as a product of sums

(p + ~r)(p + s)(q + ~r)(q + s)

Any Boolean expression in CNF can be converted to an equivalent DNF and vice versa

- **the two Boolean expressions above are equivalent**

$AST(p_r)$

Abstract syntax tree

Each leaf node represents a Boolean variable or relational expression

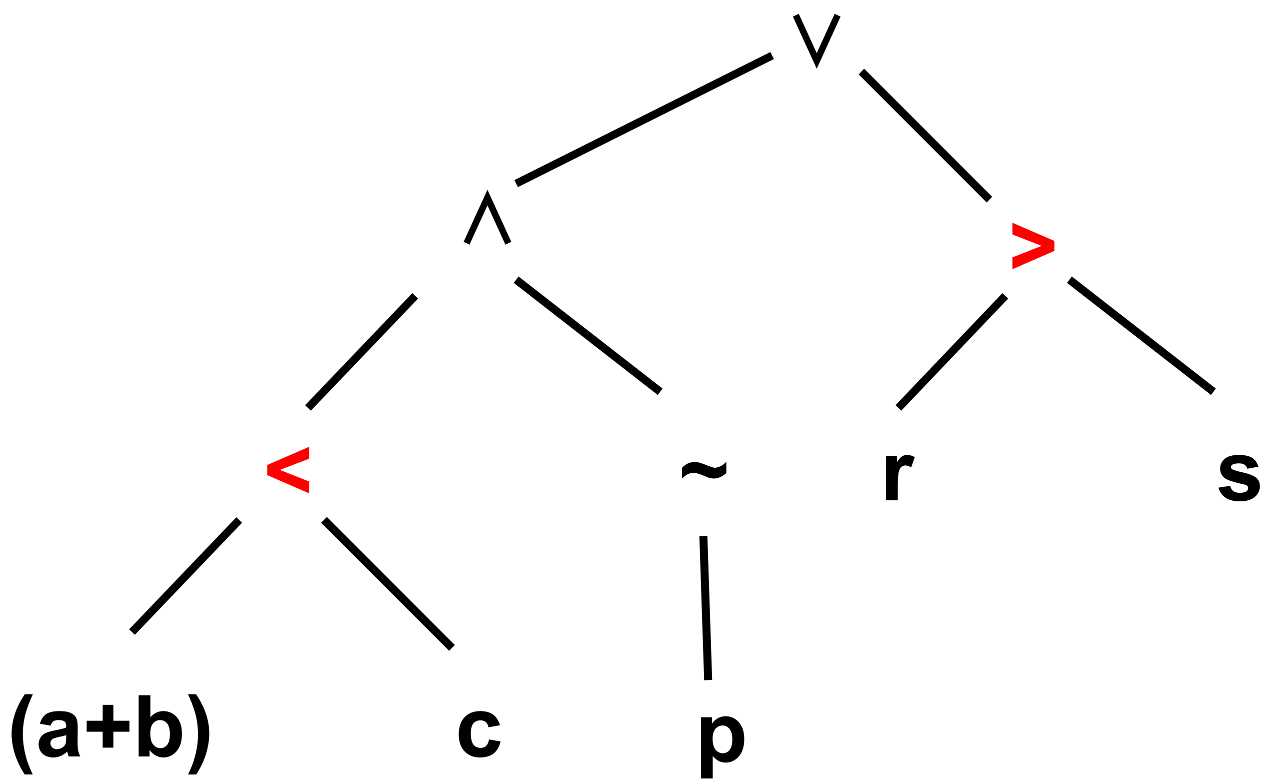
An internal node is a Boolean operator

Draw the AST for

$$(a + b < c) \wedge (\sim p) \vee (r > s)$$

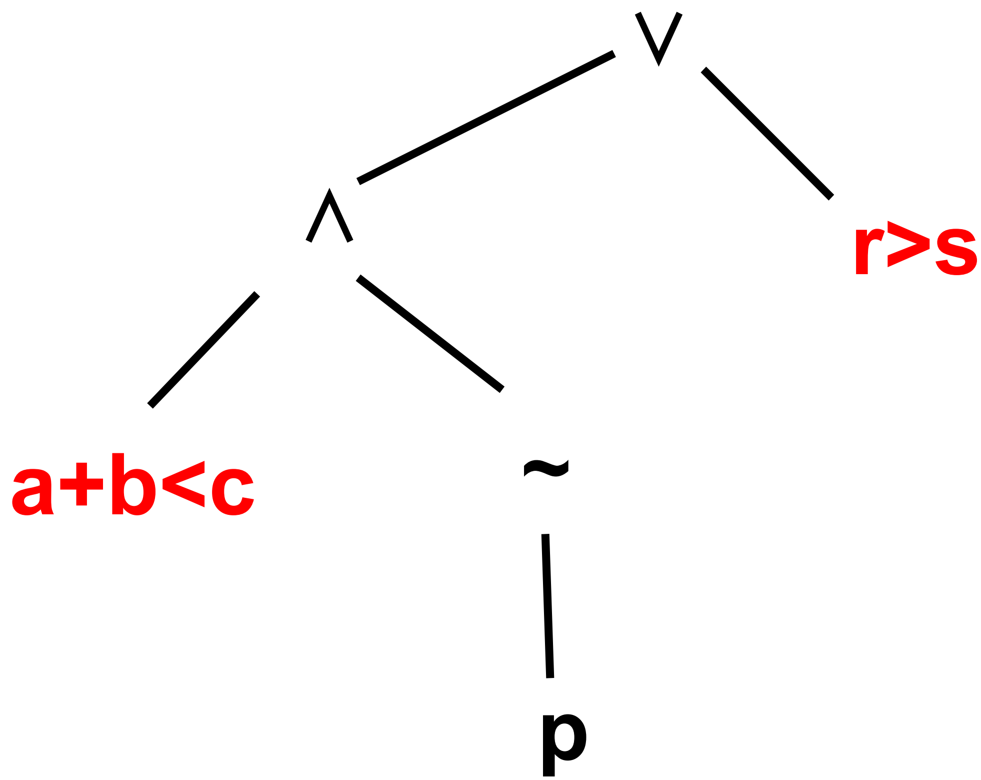
Note that a,b,c, r,s are numeric; p is Boolean

$$AST\,((a + b < c) \land (\sim p) \lor (r > s))$$



Mathur, Fig 2.1

$$AST\,((a + b < c)\ \land\ (\sim p)\ \lor\ (r > s))$$



Modified Mathur, Fig 2.1
- internal nodes are Boolean operators

Control-Flow Graph (CFG)

Captures flow of control within a program

A basic block in program P is a sequence of consecutive statements with a single entry and a single exit point.

- **a block has unique entry and exit points**
- **ignore syntactic markers such as *begin*, *else*, *{*, *}*, *end***
- **some tools treat procedure calls as basic blocks**

Multiple entry/exit means that you enter/exit the basic block from more than one place

- **more than one edge in and/or out**

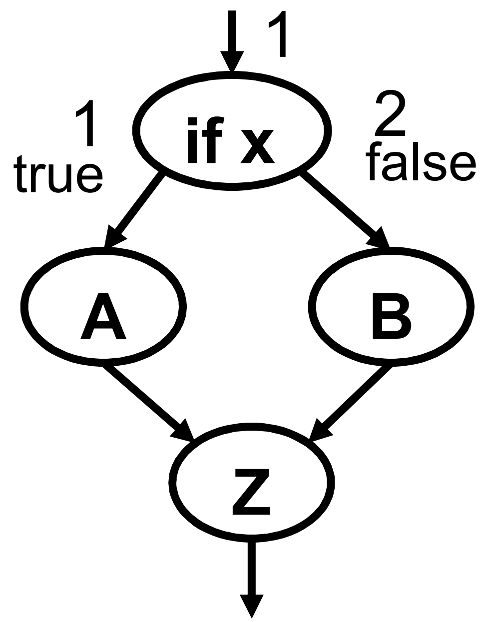
Simple Rules for Building CFGs

1. **An *if* ends the block that it is in.**
   - **after the *if*, the program may flow in two directions: *then* and else clauses**
   - **one entry, two exits**

2. **A *while* statement (pre-test loop) is in a block by itself.**
   - **after the *while*, the program may flow in two directions: the body of the loop or to exit the loop**
   - **the loop statement may be entered from two directions: the code preceding the loop or the end of the loop**
   - **two entries, two exits**

Entry/Exit Flows

if x then
{A}
else
{B}
Z…

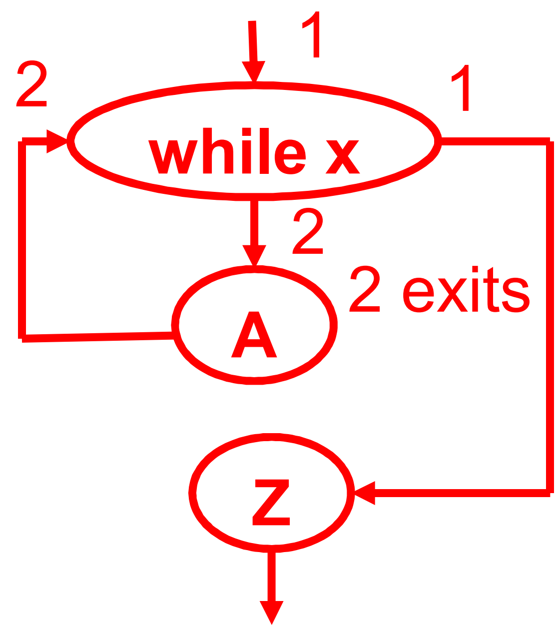


1 entry

2 exits

2 entries

while x loop
{A}
end loop;
Z…



Beginning a Basic Block

Instructions which begin a new basic block include

- **procedure and function entry points**
- **targets of jumps or branches**
  - ***else* clause block of code**
  - ***loop* body**
- **"fall-through" instructions following some conditional branches**
  - ***then* clause block of code**
  - **block of code following an *if-then-else***
- **instructions following ones that throw exceptions**
- **exception handlers**

Ending a Basic Block

Instructions that end a basic block include

- **unconditional and conditional branches, both direct and indirect**
- **the return instruction**
- ***function calls can be at the end of a basic block if they cannot return***
    - ***such as functions which throw exceptions or special calls like C's longjmp and exit***
- ***instructions which may throw an exception***

Basic Blocks Example

```
1)  integer X(3), Y, Z;
2)  input (X(1));
3)  X(2) := X(1) * 2;
4)  X(3) := X(2) * 3;
5)  Y := Z := 0;
6)  for i:=1,3 loop
7)    Y := Y + X(i);
8)    Z := Z + X(i) * X(i);
9)  end loop;
10) output (Y, Z);
11) end;
```

1, 2, 3, 4, 5

6 – loop (pre-test)

7, 8 (9) – loop body

10 (11)

Another Basic Blocks Example

Identify the basic blocks for the following program P1 written in pseudo-code.

Draw the control flow graph.

<u>Program P1</u>

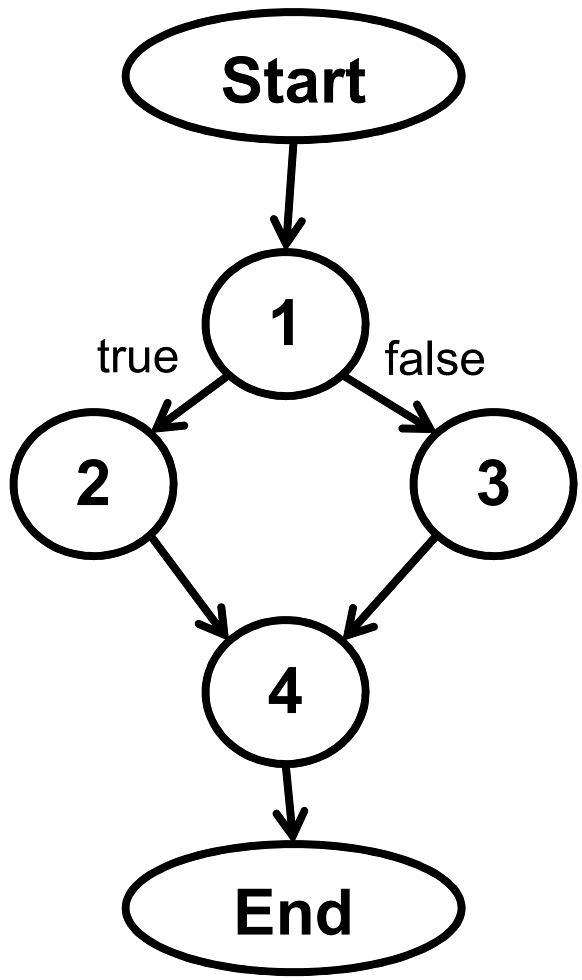
```
1)  integer A, B;
2)  input (A);
3)  if (A > 7)
4)      B = 1;
5)  else
6)      B = 2;
7)  output (A,B);
8)  end;
```

Program P1

1) integer A, B;
2) input (A);
3) if (A > 7)
4) B = 1;
5) ~~else~~
6) B = 2;
7) output (A,B);
8) ~~end;~~

Basic blocks

1 – 1, 2, 3
2 – 4 (5)
3 – 6
4 – 7 (8)



Control Flow Graph

Flow graph G aka CFG aka program graph

Defined as a finite set N of nodes and a finite set E of edges

An edge (i, j) in E connects two nodes n_i and n_j in N

G= (N, E) denotes a flow graph G with nodes given by N and edges by E

Labeling Edges

If a node in a CFG has more than one edge exiting from it…

- **for example, an IF or WHILE loop**

… implies a decision was made as to control flow

… suggests a predicate was evaluated

Label the edges

- **true / false or t/f is acceptable**
- **using the predicate (x<y) / !(x<y) is more informative and used in data flow graphs**

Start and End in CFGs

Start and End are distinguished nodes

Every other node in G is reachable from Start

- **Start has no incoming edge**

Every node in G has a path terminating at End

- **End has no outgoing edge**

CFG Example

```
1)  integer X(3), Y, Z;
2)  input (X(1));
3)  X(2) := X(1) * 2;
4)  X(3) := X(2) * 3;
5)  Y := Z := 0;
6)  for i:=1,3 loop
7)    Y := Y + X(i);
8)    Z := Z + X(i) * X(i);
9)  end loop;
10) output (Y, Z);
11) end;
```

Using basic blocks

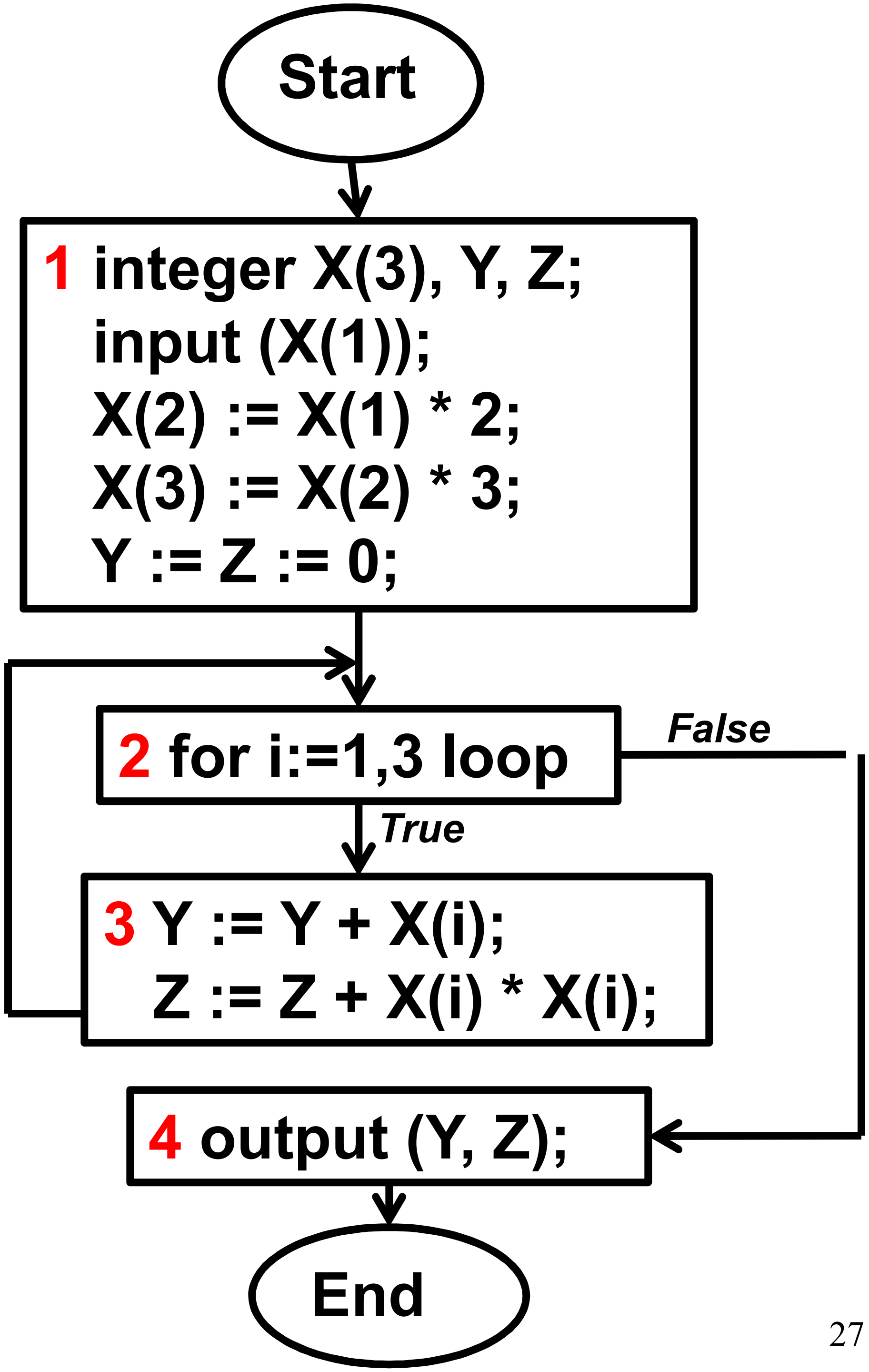
1) 1, 2, 3, 4, 5

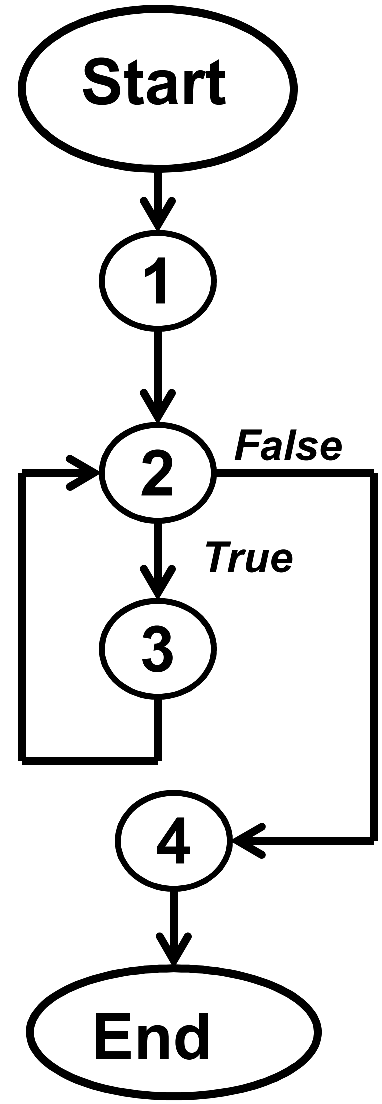
2) 6

3) 7, 8

4) 10

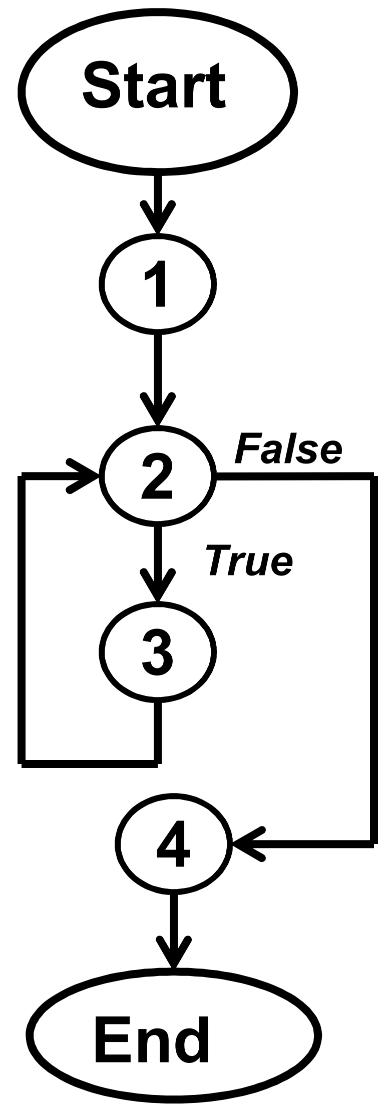
what is the flow graph?


Start
1 integer X(3), Y, Z;
input (X(1));
X(2) := X(1) * 2;
X(3) := X(2) * 3;
Y := Z := 0;
2 for i:=1,3 loop
False
True
3 Y := Y + X(i);
Z := Z + X(i) * X(i);
4 output (Y, Z);
End


Start
1
2
False
True
3
4
End

N = {Start, 1, 2, 3, 4, End}

E = {(Start,1), (1,2), (2,3), (3,2), (2,4), (4,End)}



Another CFG Example

```
input(target);
biggest := 0:
for i=1,n loop
   if (x(i) > biggest) then
            biggest := x(i);
   end if;
output("loop", i);
end loop;
if (biggest > target) then
   output("X has larger value than target");
else
   output("X largest value is", biggest);
end if;
```

CFG Solution

1- input(target);
1- biggest := 0:

2- for i=1,n loop

3-   if (x(i) > biggest) then

4-   biggest := x(i);
           ~~end if;~~

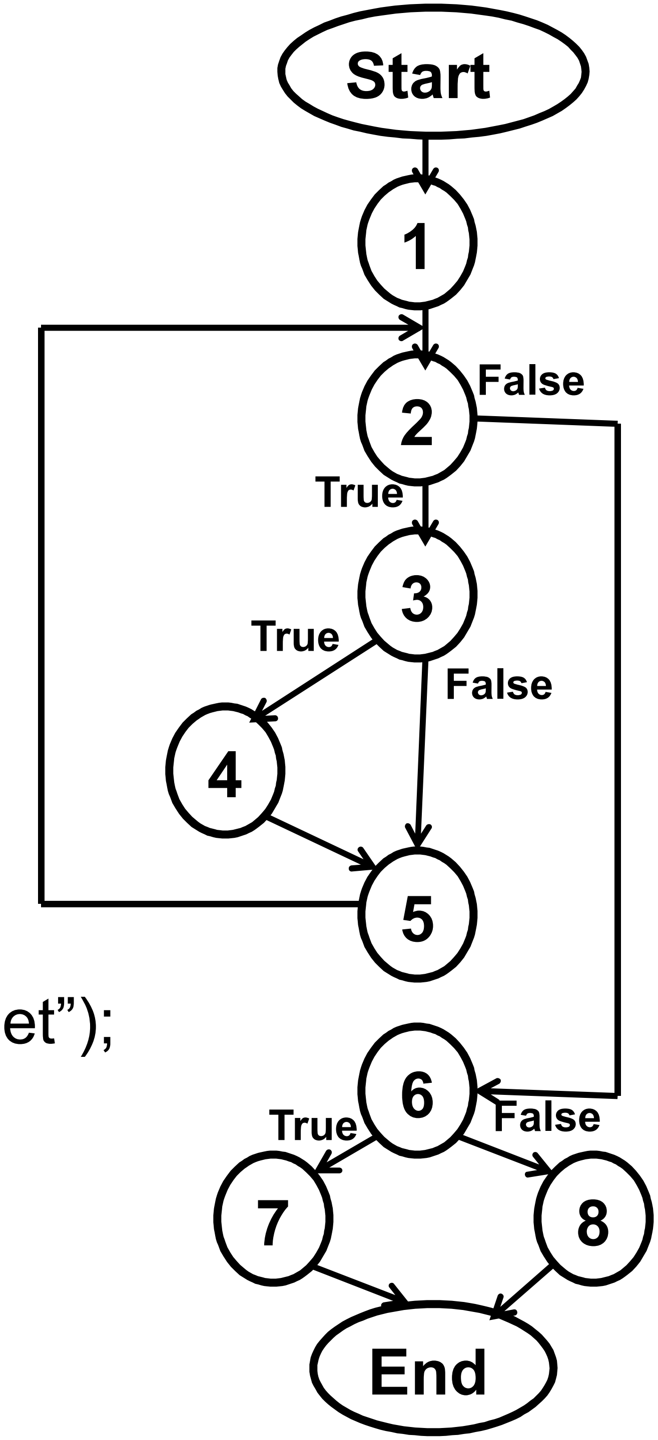
5- output("loop", i);
   ~~end loop;~~

6- if (biggest > target) then

7-   output("X has larger value than target");
   ~~else~~

8-   output("X largest value is", biggest);
   ~~end if;~~

```
1- input(target);
1- biggest := 0:
2- for i=1,n loop
3-        if (x(i) > biggest) then
4-                biggest := x(i);
          end if;
5- output("loop", i);
   end loop;
6- if (biggest > target) then
7-        output("X has larger value than target");
   else
8-        output("X largest value is", biggest);
   end if;
```



Paths

A sequence of k edges, k>0, $(e_1, e_2, \ldots e_k)$, denotes a path of length k through the flow graph G if the sequence condition holds

- **if n_p, n_q, n_r, and n_s are nodes belonging to N, and $0 < i < k$, if $e_i = (n_p, n_q)$ and $e_{i+1} = (n_r, n_s)$ then $n_q = n_r$**

Terms associated with nodes

- **descendant**
  - **there is a path from m to n**
- **proper descendant**
  - **$m \neq n$**
- **ancestor**
- **proper ancestor**

Successors and Predecessors

If there is an edge (n,m) in E, then

- **m is a successor of n**
- **n is a predecessor of m**

The set of all successor nodes of n is denoted succ(n)

The set of all predecessor nodes of n is denoted pred(n)

Complete and Feasible Paths

A path through G is <u>complete</u> if the first node along the path is Start and the terminating node is End.

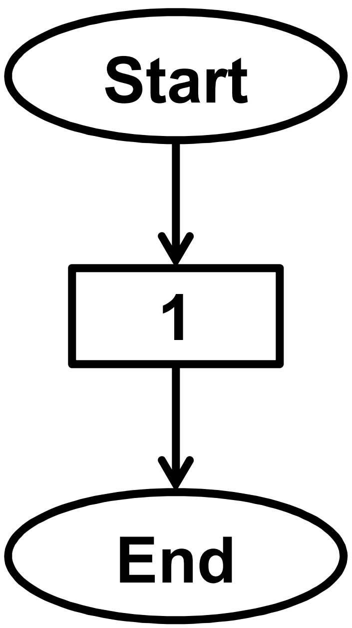
A path p is <u>feasible</u> if there exists at least one test case which when input to program P causes p to be traversed.

- **if no such test case exists path p is considered <u>infeasible</u>**

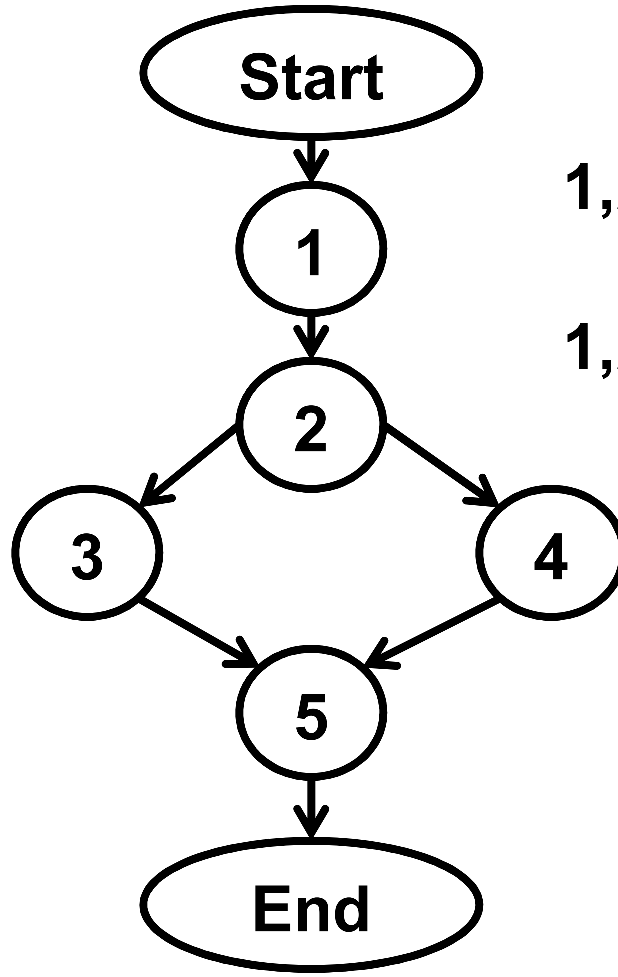
Whether a path p through program P is feasible is in general an undecidable problem.

Questions About Paths

What happens to the number of paths when you put an IF statement in a program with no existing control structures?

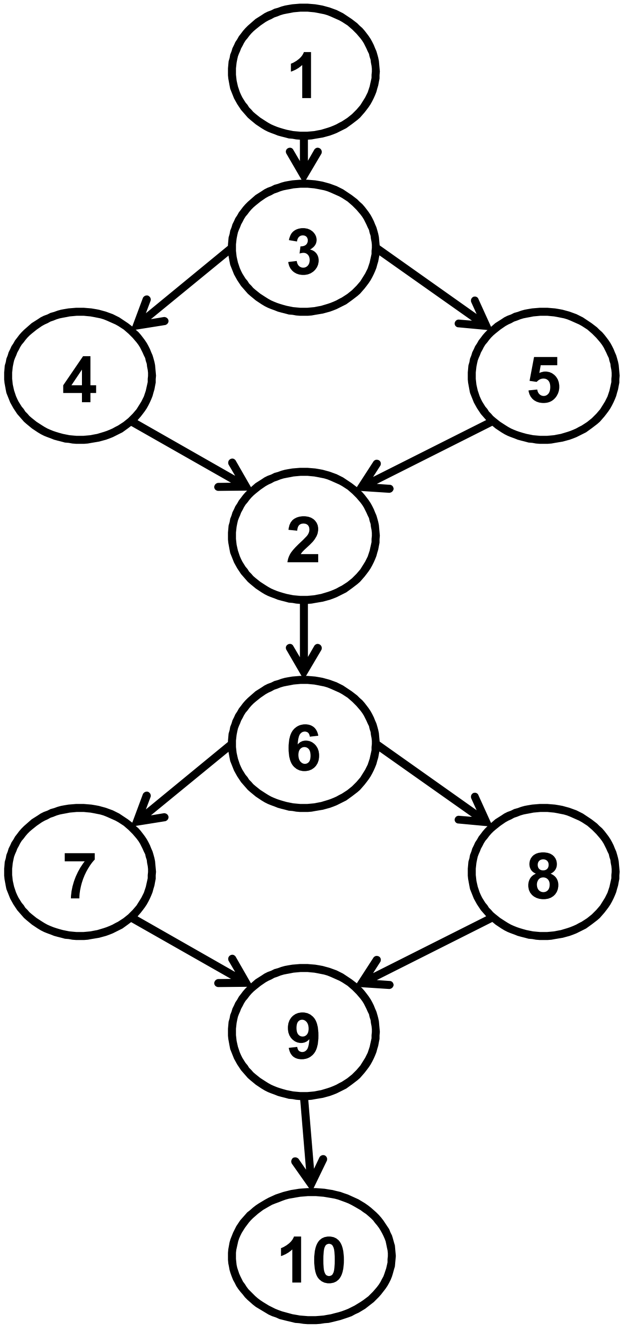


Start,1,End



1,2,3,5

1,2,4,5



Sequential IFs

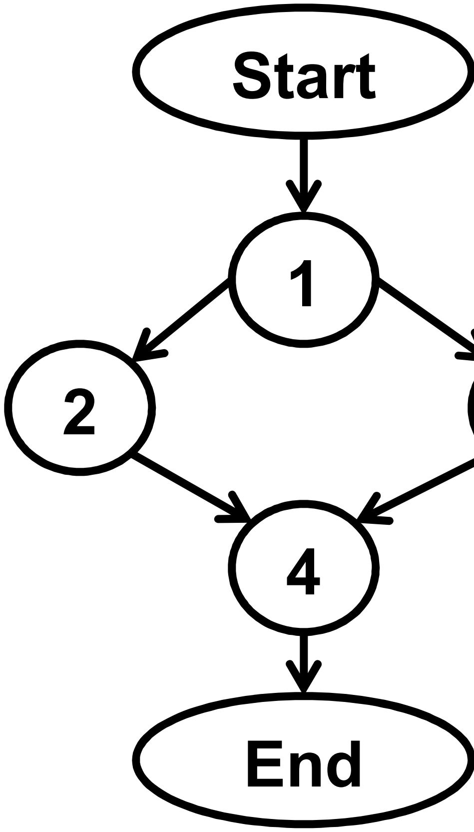
1,3,4,2,6,7,9,10

1,3,4,2,6,8,9,10

1,3,5,2,6,7,9,10

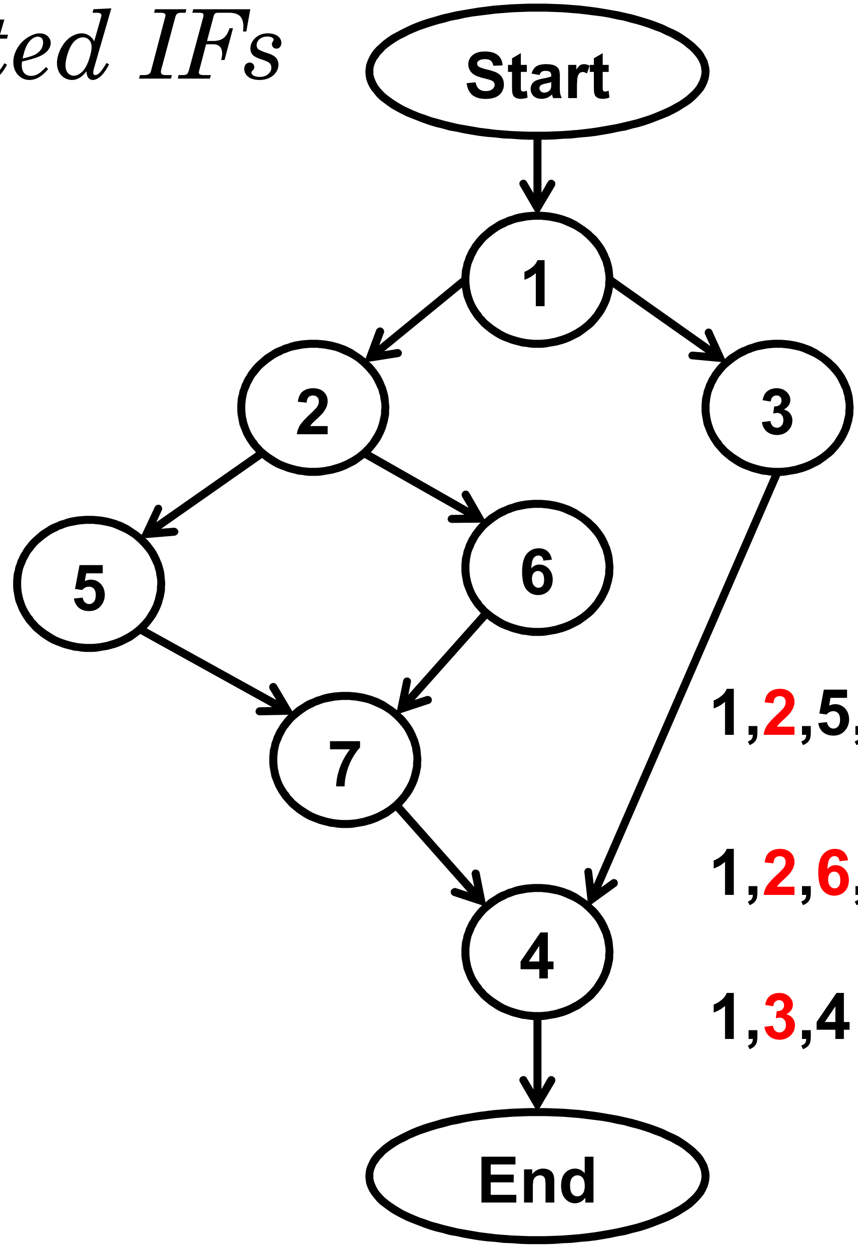
1,3,5,2,6,8,9,10

Nested IFs



1,2,4

1,3,4



1,2,5,7,4

1,2,6,7,4

1,3,4

Loops

What happens if you put a LOOP in a program?

- **adds n paths**
- **each execution of the loop adds a new path**
- **paths are dynamic (not static)**
- **if “n” is the number of iterations, n could be any value from 0 to the maximum size integer on the computer to “infinite”**
  - **do while (true)**

Structured Program Theorem
(Böhm-Jacopini Theorem)

Any algorithm can be expressed using only three control structures.

- **executing one subprogram, and then another subprogram (sequence)**
- **executing one of two subprograms according to the value of a Boolean expression (selection)**
- **executing a subprogram until a Boolean expression is true (iteration)**

These can be represented, respectively, by the concatenation, union, and star operations of a regular expression.

Cyclomatic Complexity and Nonstructured Programs

There are some specific conditions where an unstructured construct works best.

- **McCabe page 315, Donald Knuth**

The cyclomatic complexity of a nonstructured program is at least 3.

The cyclomatic complexity of a structured program is at least 1.

Properties of Cyclomatic Complexity

$V(G) \geq 1$

$V(G)$ is the maximum number of linearly independent paths in G.

Inserting or deleting functional statements to G does not affect $V(G)$.

G has only one path if and only if $V(G) = 1$.

Inserting a new edge in G increases $V(G)$ by 1.

$V(G)$ depends only on the decision structure of G.

Summary – Things to Remember

Boolean algebra
- **DNF and CNF**
- **singular and mutually singular**

Drawing ASTs

Drawing CFGs

Complete and feasible paths

Structured program theorem

Questions and Answers