

Program Representation

Xiangyu Zhang

Why Program Representations

- Original representations
 - Source code (across languages).
 - Binaries (across machines and platforms).
 - Source code / binaries + test cases.
- They are hard for machines to analyze.
- Software are translated into certain representations before analyses are applied.

Outline

- Control flow graphs.
- Program dependence graphs.
- Super control flow graphs.

Control Flow Graph

- Chapter 1.14 of “Foundations of Software Engineering”
- Available on the course website.
- The most commonly used program representation.

Program representation: Basic blocks

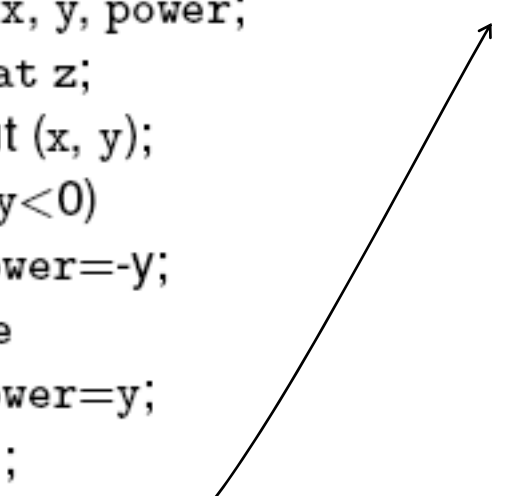
A **basic block** in program P is a sequence of consecutive statements with a single entry and a single exit point. Thus a block has unique entry and exit points.

Control always enters a basic block at its entry point and exits from its exit point. There is no possibility of exit or a halt at any point inside the basic block except at its exit point. The entry and exit points of a basic block coincide when the block contains only one statement.

Basic blocks: Example

Example: Computing x raised to y

```
1  begin
2  int x, y, power;
3  float z;
4  input (x, y);
5  if (y<0)
6    power=-y;
7  else
8    power=y;
9  z=1;
10 while (power!=0){
11   z=z*x;
12   power=power-1;
13 }
14 if (y<0)
15   z=1/z;
16 output(z);
17 end
```



Basic blocks: Example (contd.)

Basic blocks

Block	Lines	Entry point	Exit point
1	2, 3, 4, 5	1	5
2	6	6	6
3	8	8	8
4	9	9	9
5	10	10	10
6	11, 12	11	12
7	14	14	14
8	15	15	15
9	16	16	16

Control Flow Graph (CFG)

A **control flow graph** (or flow graph) G is 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 . We often write $G = (N, E)$ to denote a flow graph G with nodes given by N and edges by E .

Control Flow Graph (CFG)

In a flow graph of a program, each basic block becomes a node and edges are used to indicate the flow of control between blocks.

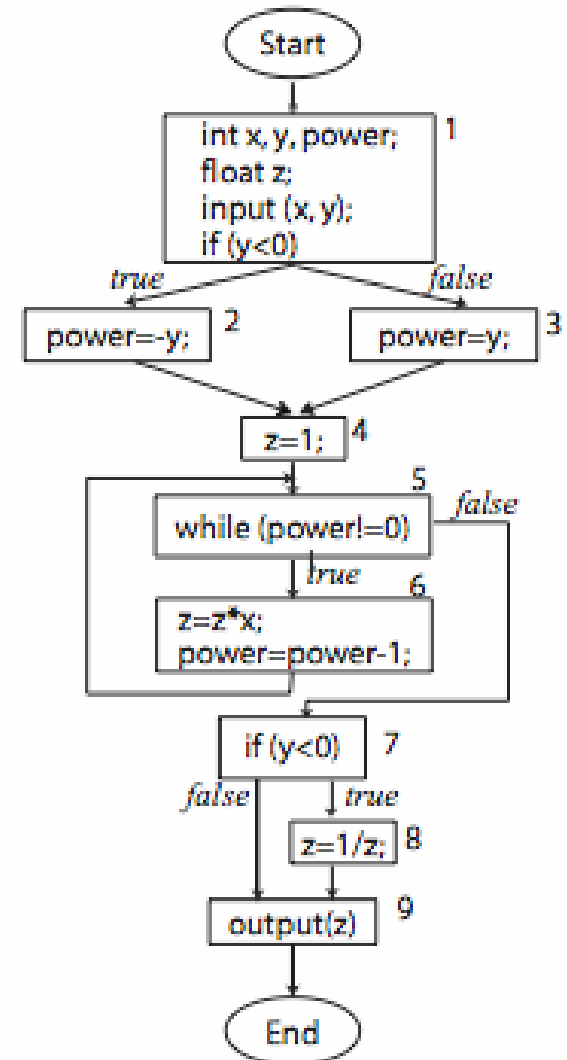
An edge (i, j) connecting basic blocks b_i and b_j implies that control can go from block b_i to block b_j .

We also assume that there is a node labeled **Start** in N that has no incoming edge, and another node labeled **End**, also in N , that has no outgoing edge.

CFG Example

$N = \{\text{Start}, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{End}\}$

$E = \{(\text{Start}, 1), (1, 2), (1, 3), (2, 4), (3, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 8), (7, 9), (9, \text{End})\}$

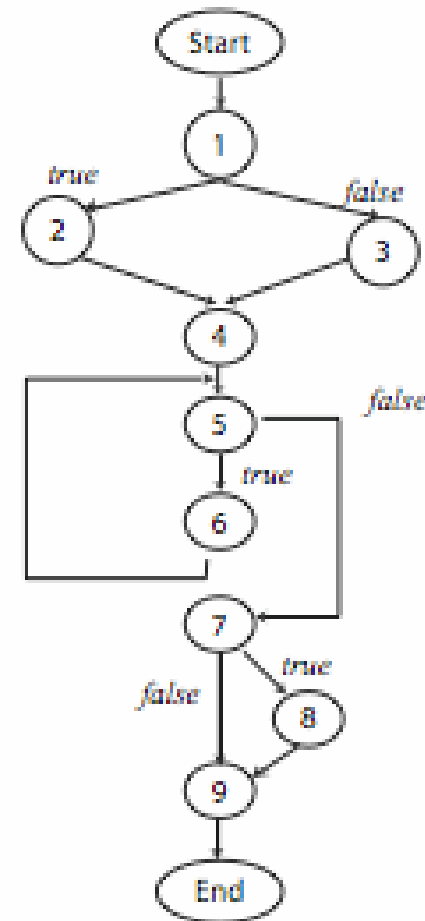


CFG Example

Same CFG with statements removed.

$N = \{\text{Start}, 1, 2, 3, 4, 5, 6, 7, 8, 9, \text{End}\}$

$E = \{(\text{Start}, 1), (1, 2), (1, 3), (2, 4), (3, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 8), (7, 9), (9, \text{End})\}$



Paths

Consider a flow graph $G = (N, E)$. A sequence of k edges, $k > 0$, (e_1, e_2, \dots, e_k) , denotes a path of length k through the flow graph if the following sequence condition holds.

Given that 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 $n_q = n_r$. }

Paths: sample paths through the exponentiation flow graph

Two feasible and complete paths:

$p_1 = (\text{Start}, 1, 2, 4, 5, 6, 5, 7, 9, \text{End})$

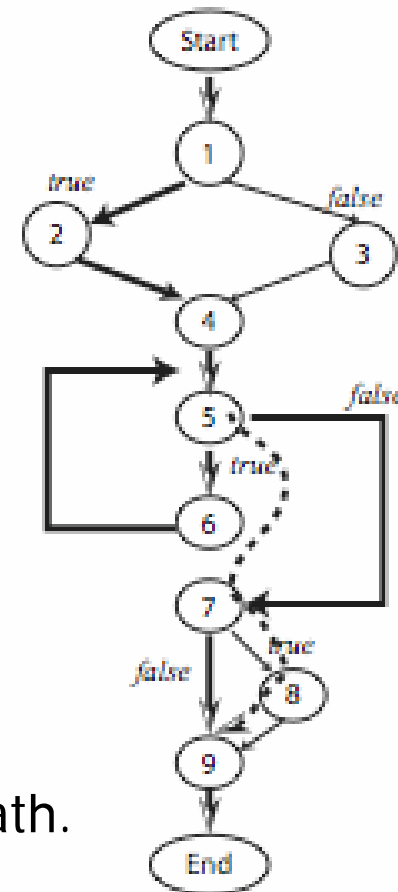
$p_2 = (\text{Start}, 1, 3, 4, 5, 6, 5, 7, 9, \text{End})$

Specified unambiguously using edges:

$p_1 = ((\text{Start}, 1), (1, 2), (2, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 9), (9, \text{End}))$

Bold edges: complete path.

Dashed edges: subpath.

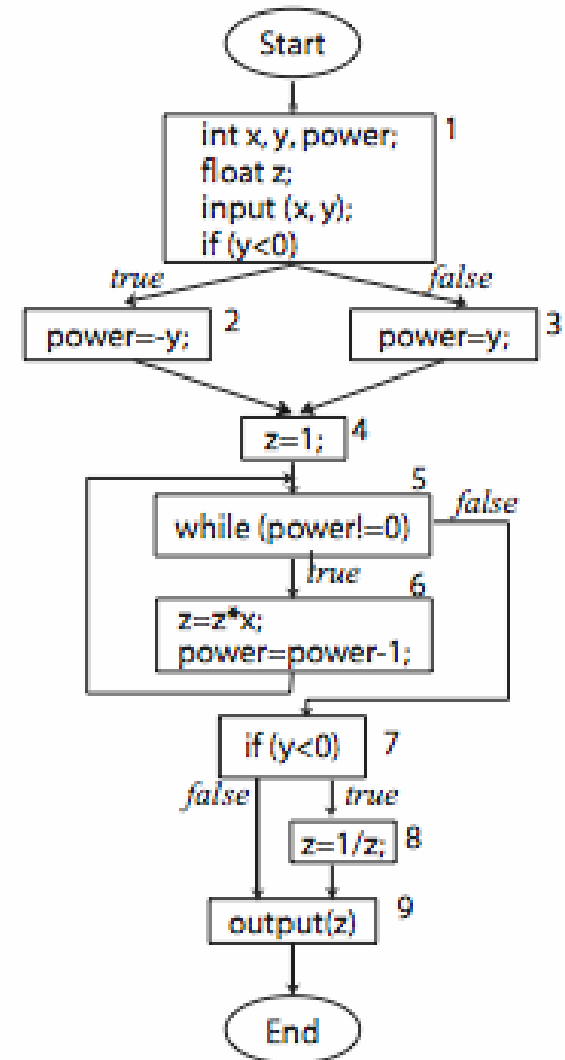


Paths: infeasible paths

A path p through a flow graph for program P is considered **feasible** if there exists at least one test case which when input to P causes p to be traversed.

$p_1 = (\text{Start}, 1, 3, 4, 5, 6, 5, 7, 8, 9, \text{End})$

$p_2 = (\text{Start}, 1, 2, 4, 5, 7, 9, \text{End})$



Number of paths

There can be many distinct paths through a program. A program with no condition contains exactly one path that begins at node Start and terminates at node End.

Each additional condition in the program can increase the number of distinct paths by at least one.

Depending on their location, conditions can have a multiplicative effect on the number of paths.

A Simplified Version of CFG

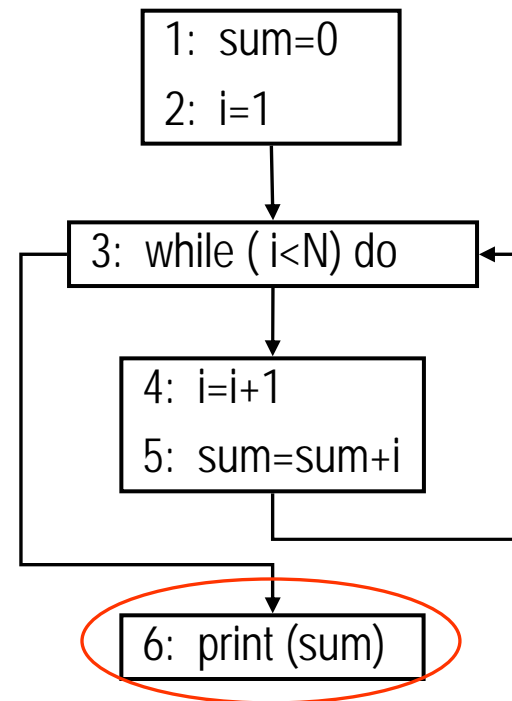
- Each statement is represented by a node
 - For readability.
 - Not for efficient implementation.

Dominator

- X **dominates** Y if all possible program path from START to Y has to pass X.

```
1: sum=0
2: i=1
3: while ( i<N) do
4:     i=i+1
5:     sum=sum+i
   endwhile
6: print(sum)
```

$DOM(6)=\{1,3, 6\}$

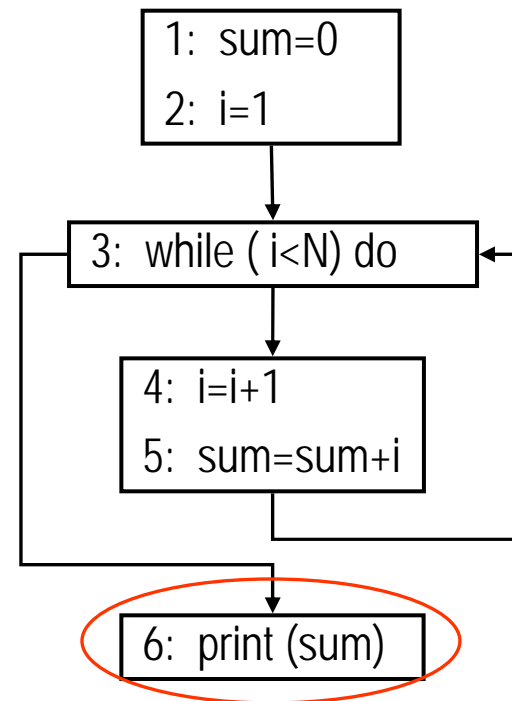


Dominator

- **X strictly dominates Y** if X dominates Y and $X \neq Y$

```
1: sum=0
2: i=1
3: while ( i<N) do
4:     i=i+1
5:     sum=sum+i
   endwhile
6: print(sum)
```

SDOM(6)={1,3}

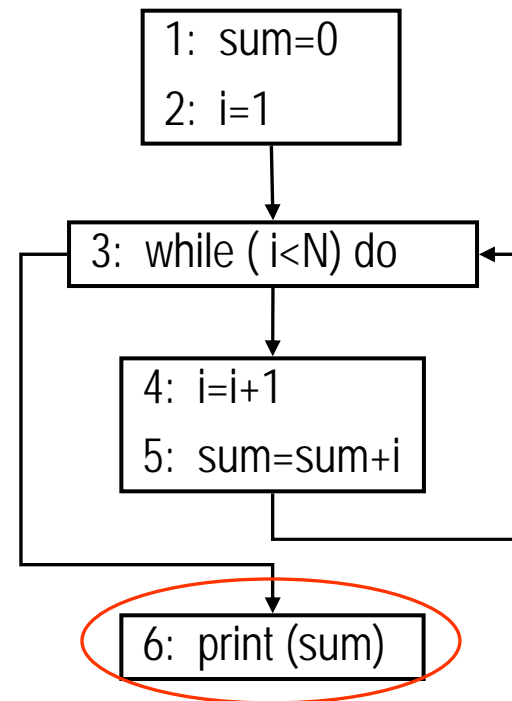


Dominator

- X is the **immediate dominator** of Y if X is the last dominator of Y along a path from Start to Y.

```
1: sum=0
2: i=1
3: while ( i<N) do
4:     i=i+1
5:     sum=sum+i
   endwhile
6: print(sum)
```

IDOM(6)={3}



Postdominator

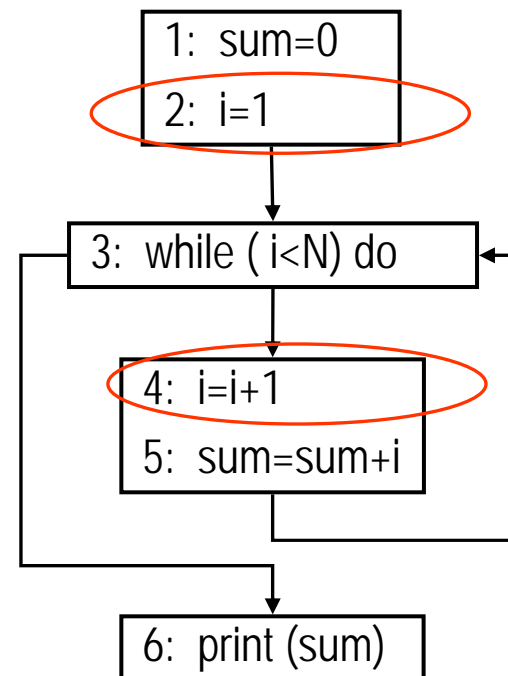
- X **post-dominates** Y if every possible program path from Y to End has to pass X.
 - Strict post-dominator, immediate post-dominance.

```

1:  sum=0
2:  i=1
3:  while ( i<N) do
4:      i=i+1
5:      sum=sum+i
        endwhile
6:  print(sum)

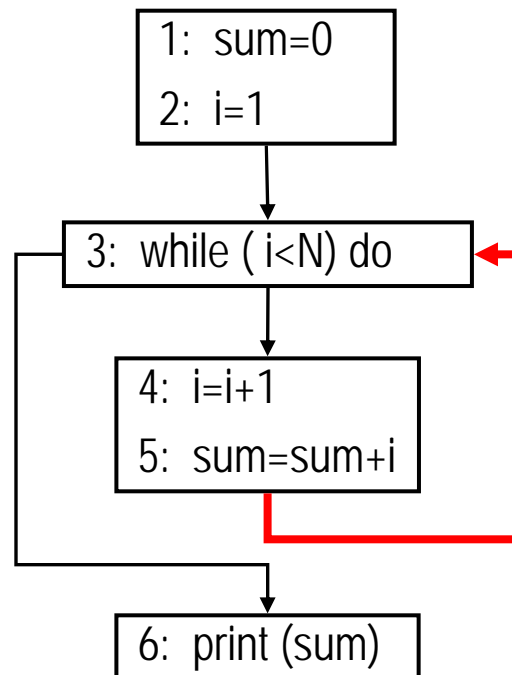
```

SPDOM(4)={3,6} IPDOM(4)=3



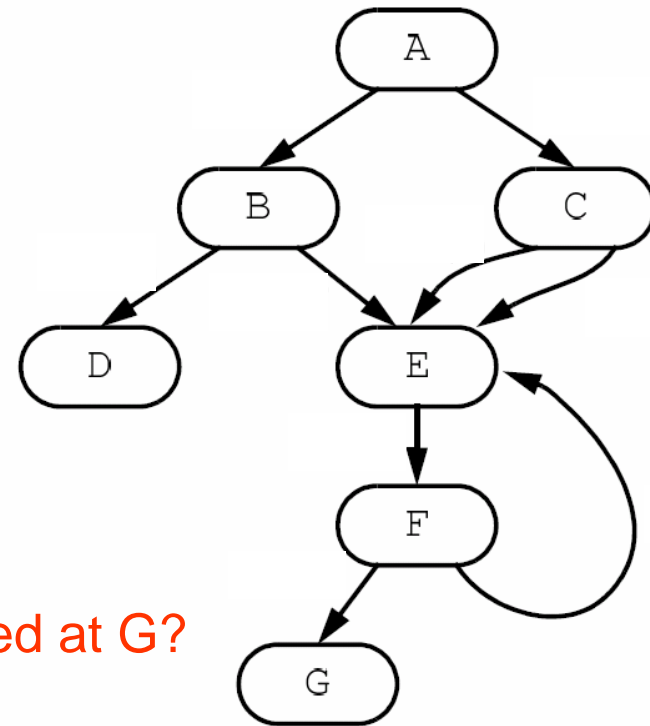
Back Edges

- A back edge is an edge whose head dominates its tail
 - Back edges often identify loops



Call Graph

- Call graph
 - nodes are procedures
 - edges are calls
- Hard cases for building call graph
 - calls through function pointers

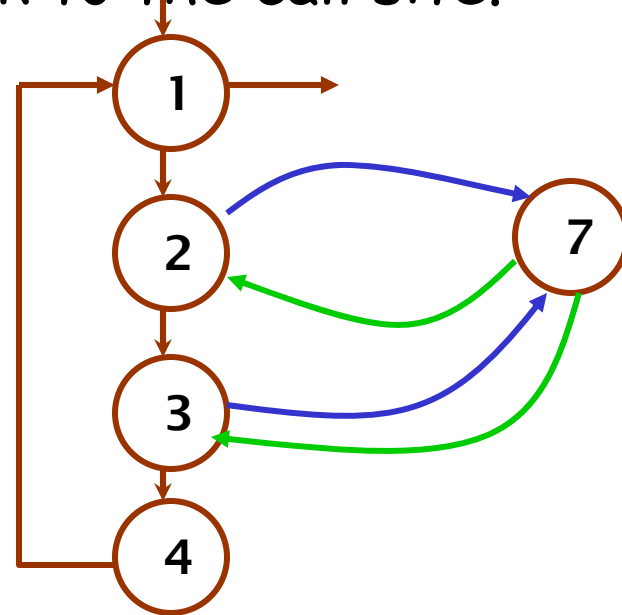


Can the password acquired at A be leaked at G?

Super Control Flow Graph (SCFG)

- Besides the normal intraprocedural control flow graph, additional edges are added connecting
 - Each call site to the beginning of the procedure it calls.
 - The return statement back to the call site.

```
1: for (i=0; i<n; i++) {  
2:   t1= f(0);  
3:   t2 = f(243);  
4:   x[i] = t1 + t2 + t3;  
5: }  
6: int f (int v) {  
7:   return (v+1);  
8: }
```



The Use of SCFG

- When reasoning across function boundaries is needed.
 - A mouse click suddenly drives a desktop application into a coma, and the operating system declares it “not responding”. While the application usually responds eventually, no user actions can be taken during the wait.

Many Other Representations

- Points-to Graph.
- Static single assignment (SSA).