MONTANA STATE UNIVERSITY

10/16/2025

Malicious Code Analysis

Fangtian Zhong

CSCI 591

Gianforte School of Computing

Norm Asbjornson College of Engineering

E-mail: fangtian.zhong@montana.edu

Overview

01 Control Flow Graph

02 Program Dependency Graph

03 Call Graph

Part One

01

Control Flow Graph

2025-10-27

Control flow graphs

The most commonly used program representation.

Vulnerability analysis

Malware analysis

Program Optimization

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

```
begin
    int x, y, power;
    float z;
    scanf("%d %d", &x, &y);
    if (y<0)
      power=-y;
    else
      power=y;
    z=1;
    while (power! = 0) {
      z=z*x;
      power = power-1;
    }
    if (y<0)
      z=1/z;
    printf("%f", z);
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

Basic blocks: Example

Example: Computing maximum

```
main:
    call _CRT_INIT
    push    rbp
    mov    rbp, rsp
    sub    rsp, 32
    mov    rcx, 100
    mov    rdx, 200
    call    print_max
    xor    eax, eax
    call ExitProcess
```

Basic blocks: Example (contd.)

Basic blocks

Block	Lines	Entry point	Exit point
1	2, 3,4,5,6,7,8	2	8
2	9,10	9	10

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.

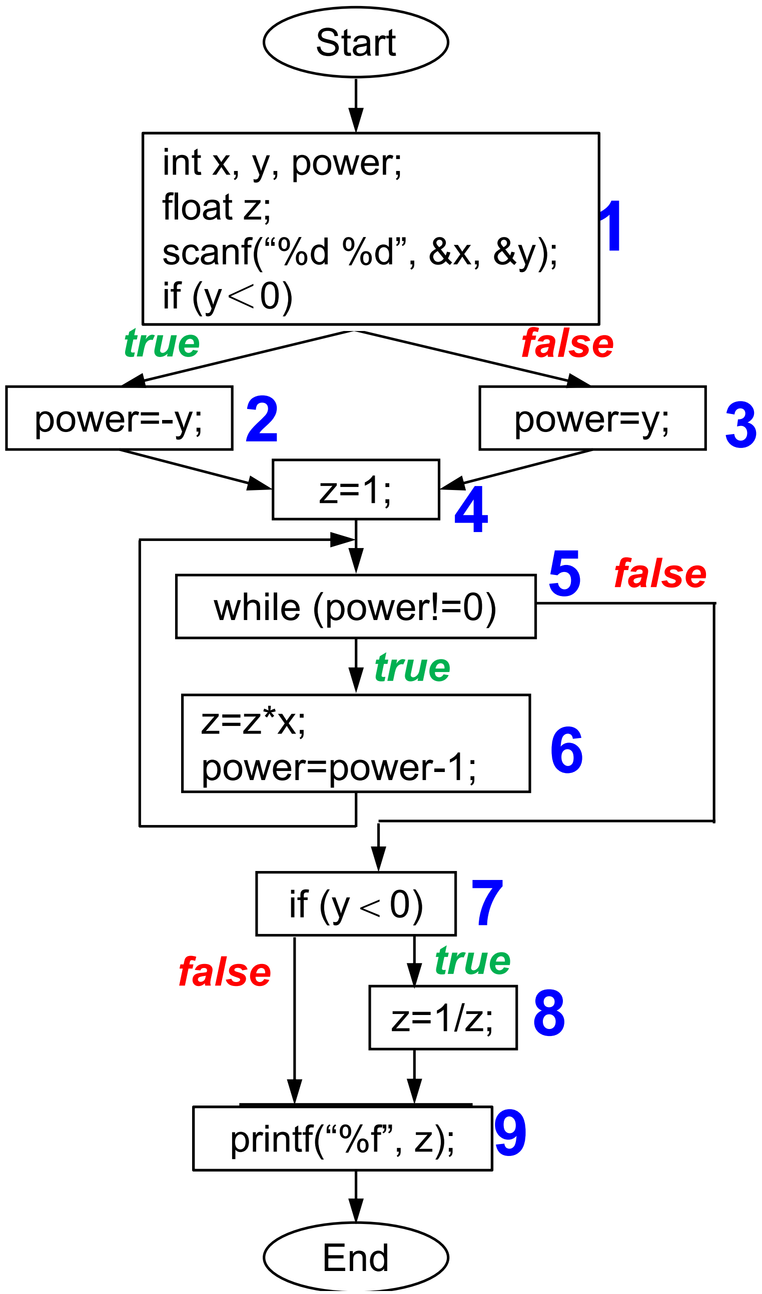
- An edge (i, j) connecting basic blocks b_i and b_j implies that control can go from block b_i to block b_j.

- 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={Start, 1, 2, 3, 4, 5, 6, 7, 8, 9, End}

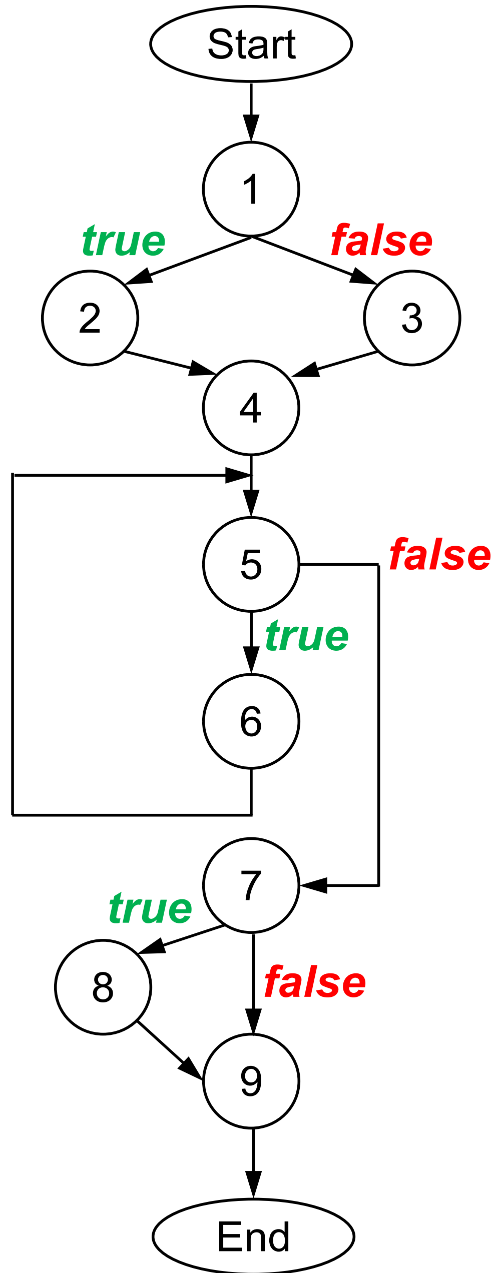
- E={(Start,1), (1, 2), (1, 3), (2,4), (3, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 8), (7, 9), (9, End)}



CFG Example

- Same CFG with statements removed.

- N={Start, 1, 2, 3, 4, 5, 6, 7, 8, 9, End}
- E={(Start,1), (1, 2), (1, 3), (2,4), (3, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 8), (7, 9), (9, End)}



Paths

- Consider a flow graph G= (N, E). A sequence of k edges, k>0, (e_1, e_2, … 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 $n_q = n_r$. }

- Complete path: a path from start to end
- Subpath: a subsequence of a complete path

Paths: sample paths through the exponentiation flow graph

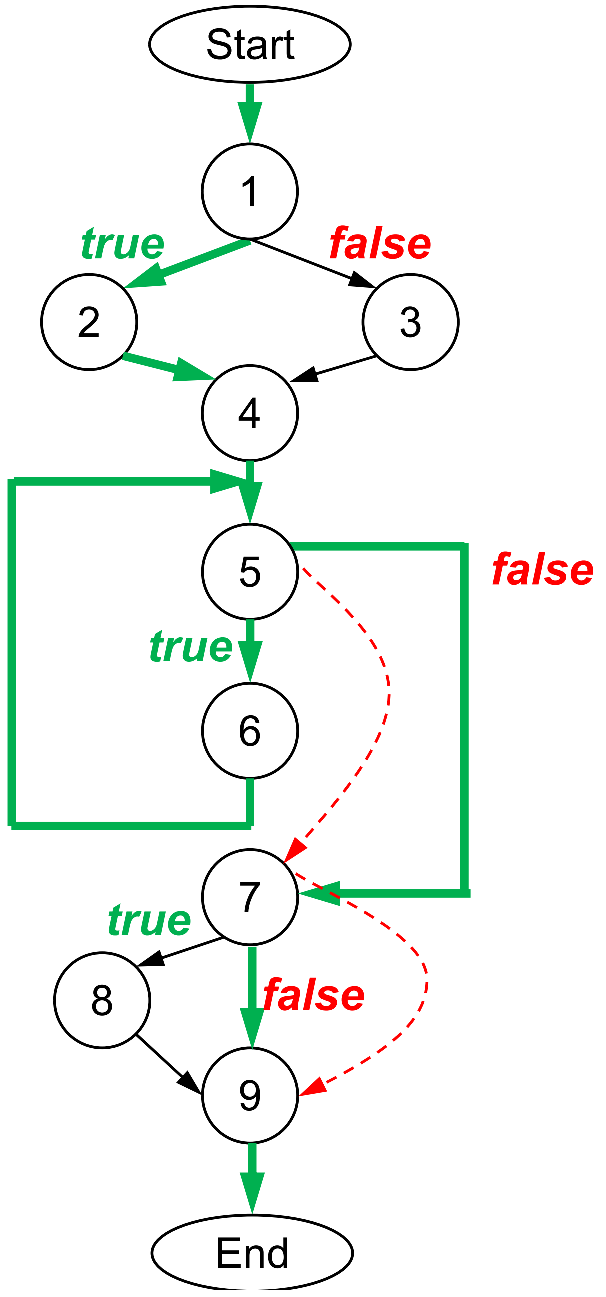
Two feasible and complete paths:

- p_1= (Start, 1, 2, 4, 5, 6, 5, 7, 9, End)
- p_2= (Start, 1, 3, 4, 5, 6, 5, 7, 9, End)

Specified unambiguously using edges:

- p_1= ((Start, 1), (1, 2), (2, 4), (4, 5), (5, 6), (6, 5), (5, 7), (7, 9), (9, End))

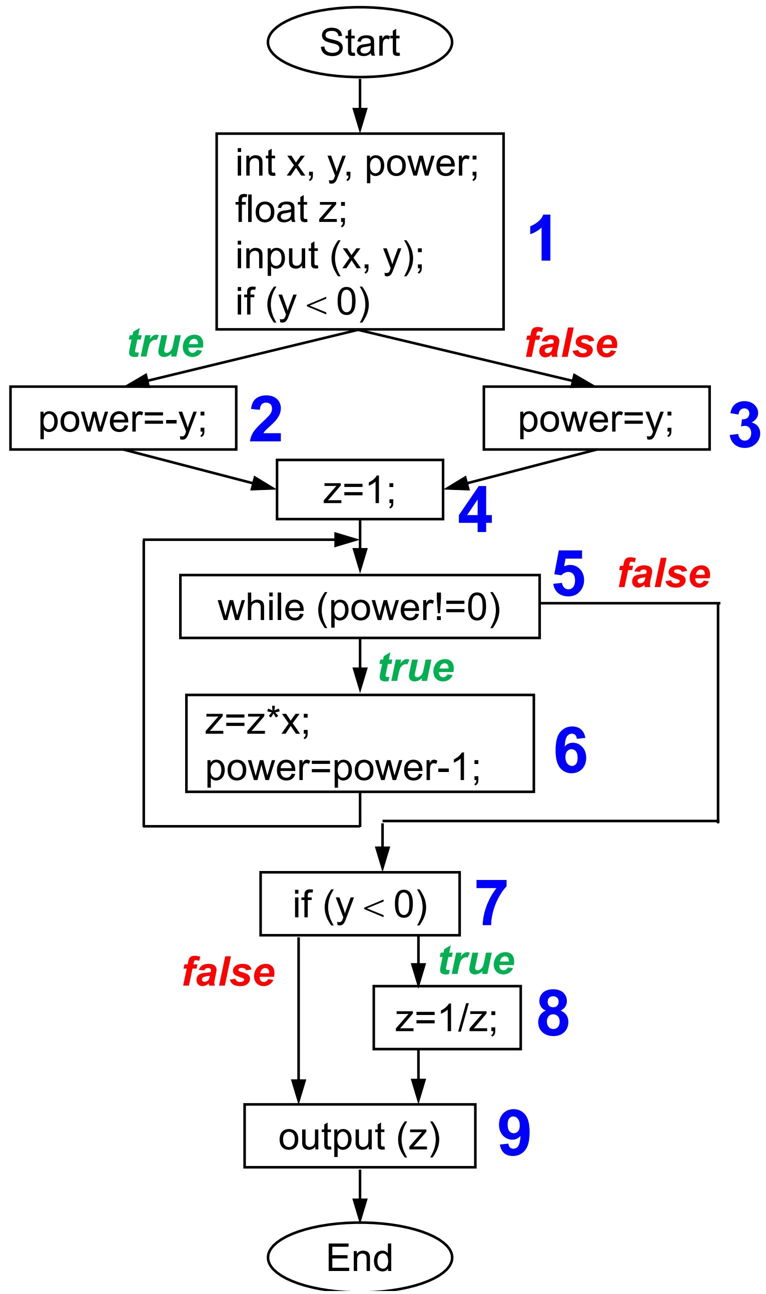
☞ *Green bold edges: complete path.*
☞ *Red dashed edges: subpath.*



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.

- p1= (Start, 1, 3, 4, 5, 6, 5, 7, 8, 9, End)
- p2= (Start, 1, 2, 4, 5, 7, 9, End)



Number of paths

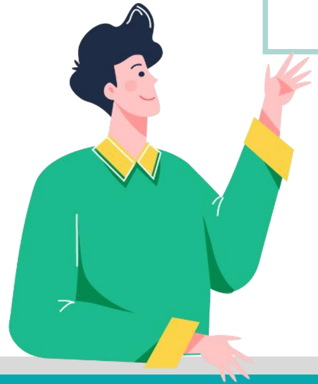
- There can be many distinct paths through a program. A program with no condition contains exactly one path that begins at node Start and terminates at node End.
- Each additional condition in the program can increase the number of distinct paths by at least one.
- Depending on their location, conditions 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 paths from START to Y have to pass X.

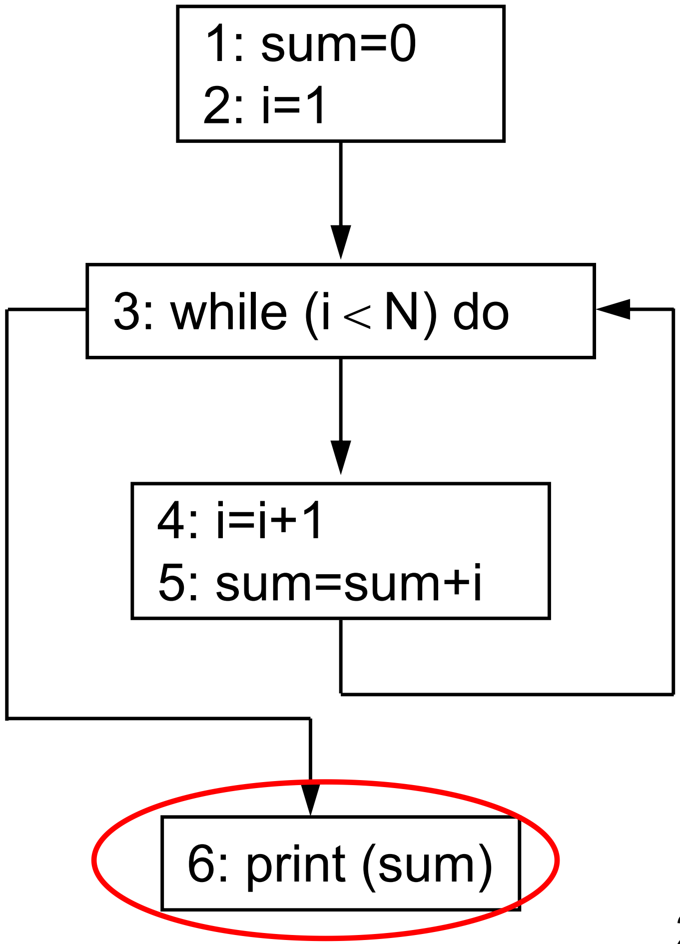


Dominator

- 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}

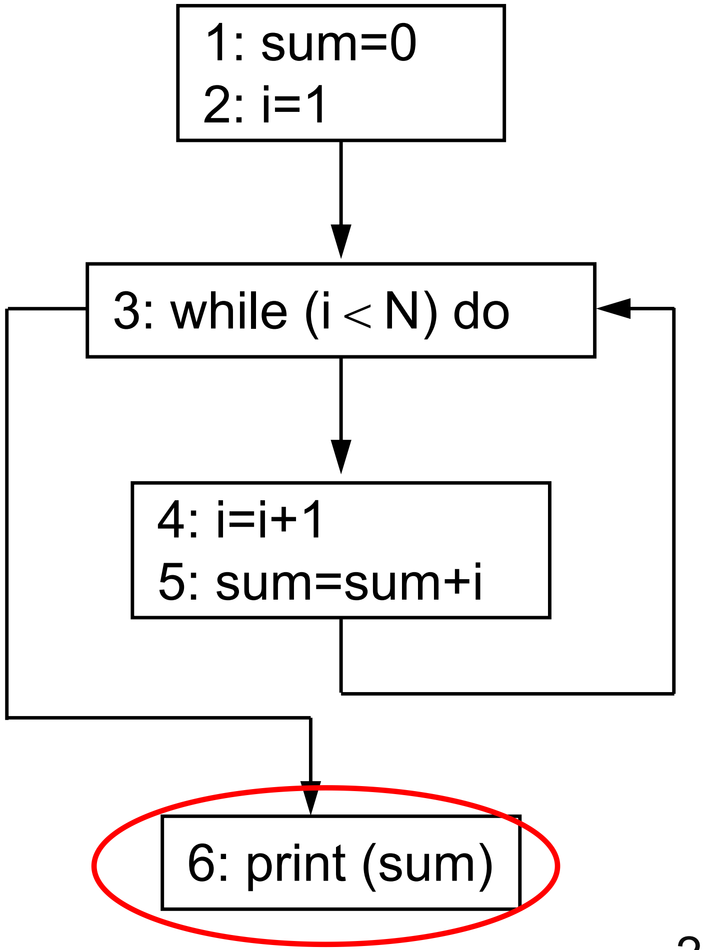


Dominator

- X **strictly dominates** Y if X dominates Y and X!=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}

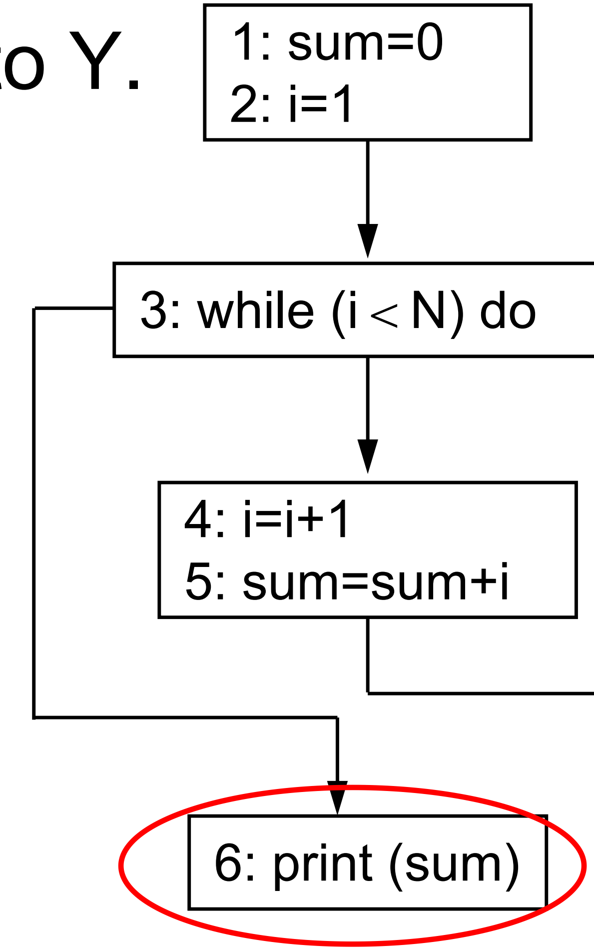


Dominator

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}

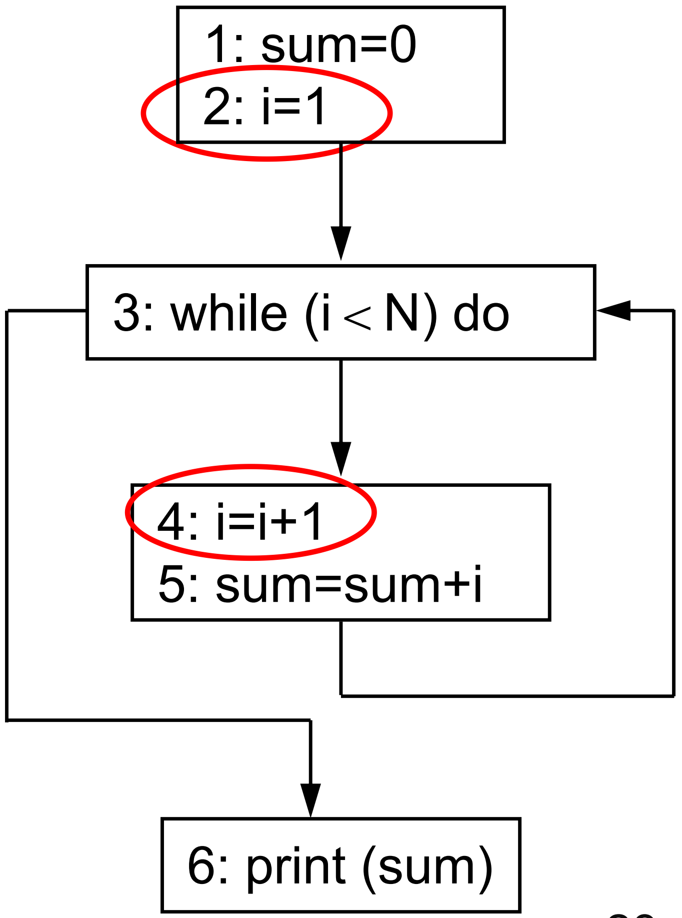


Dominator

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

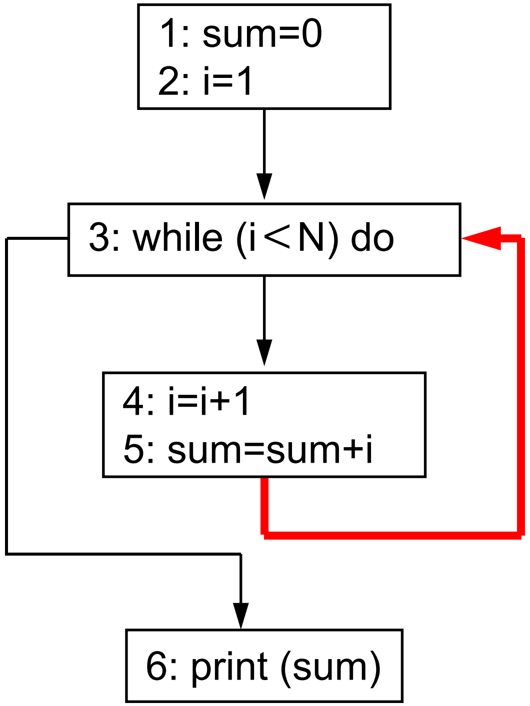
1	sum=0
2	i=1
3	while (i < N) do
4	i=i+1
5	sum=sum+i endwhile
6	**print (sum)**

- 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.



Part Two

02

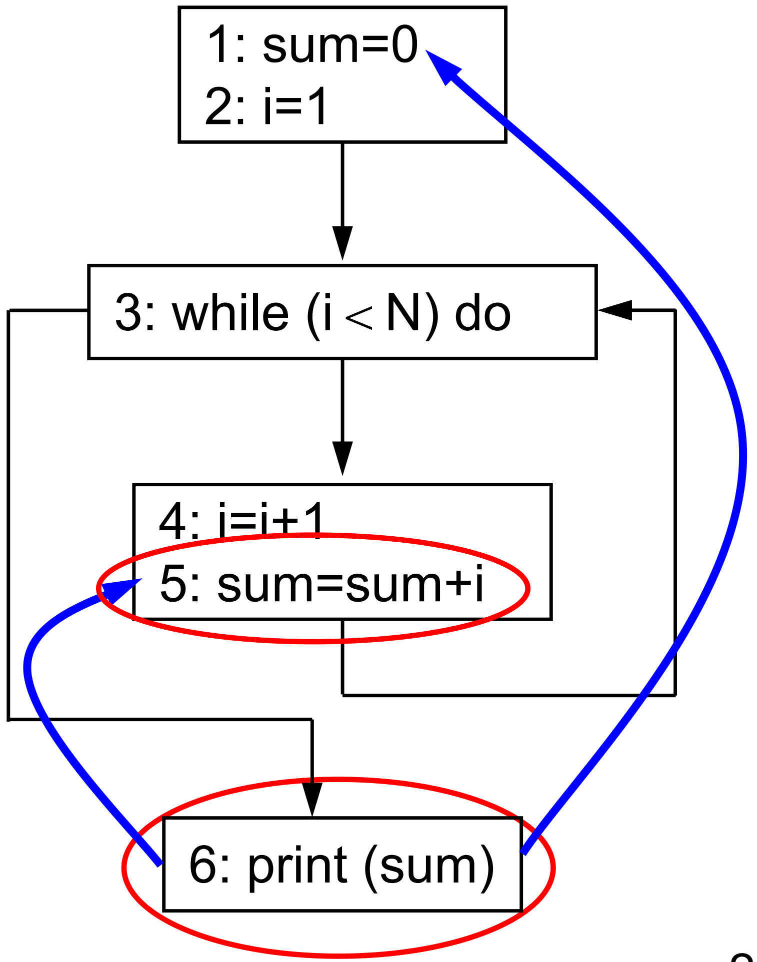
Program Dependency Graph

Program Dependence Graph

- The second widely used program representation.
- Nodes are constituted by statements instead of basic blocks.
- Two types of dependences between statements.
  - Data dependence
  - Control dependence

Data Dependence

- X is data dependent on Y if (1) there is a variable v that is defined at Y and used at X and (2) there exists a path of nonzero length from Y to X along which v is not re-defined.



Computing Data Dependence is Hard in General

Aliasing

- A variable can refer to multiple memory locations/objects.

```
int x, y, z, ...;
int * p;
x=...;
y=...;
p=& x;
p=p+z
...=*p;
```

```
foo (ClassX x, ClassY y) {
    x.field=...;
    ...=y.field;
}
```

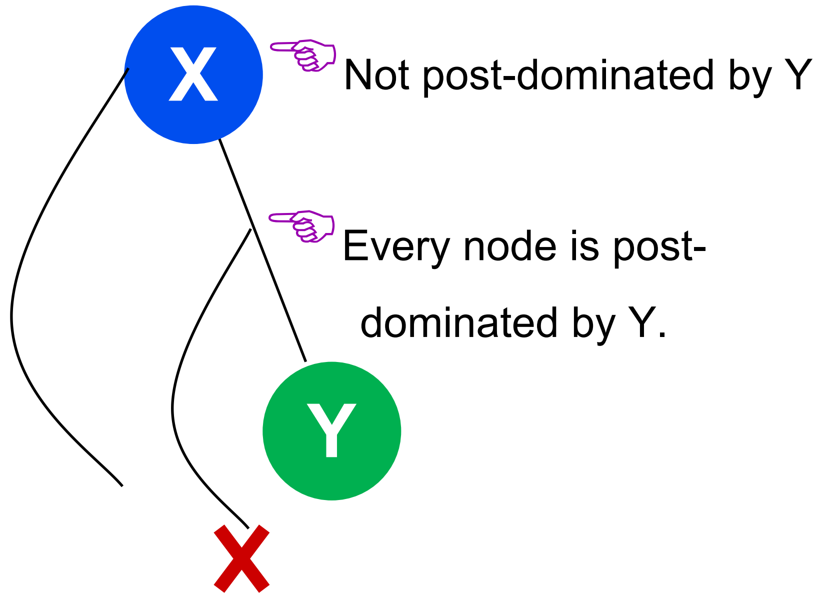
```
foo (o, o);
```

```
o1=new ClassX();
o2=new ClassY();
foo (o1, o2);
```

Control Dependence

- Intuitively, Y is control-dependent on X iff X directly determines whether Y executes (statements inside one branch of a predicate are usually control dependent on the predicate).

- X is not strictly post-dominated by Y.
  - **There is a path from X to End that does not pass Y or X==Y.**
- There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y.
  - **No such paths for nodes in a path between X and Y.**



Control Dependence - Example

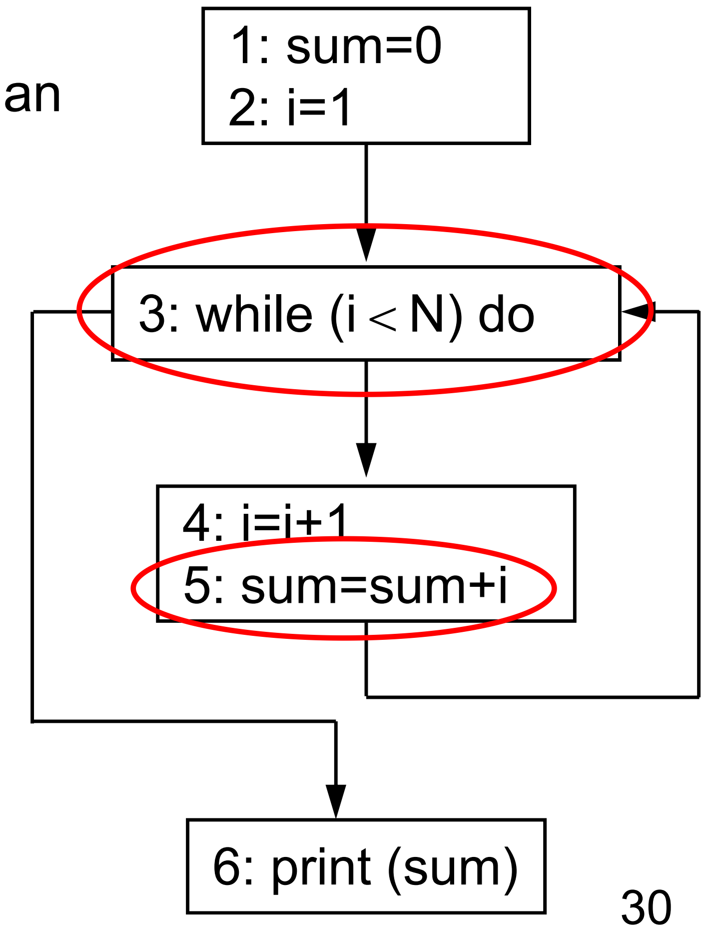
Y is control-dependent on X iff X directly determines whether Y executes.

- X is not strictly post-dominated by Y.
- There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y.

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

☞ CD(5)=3

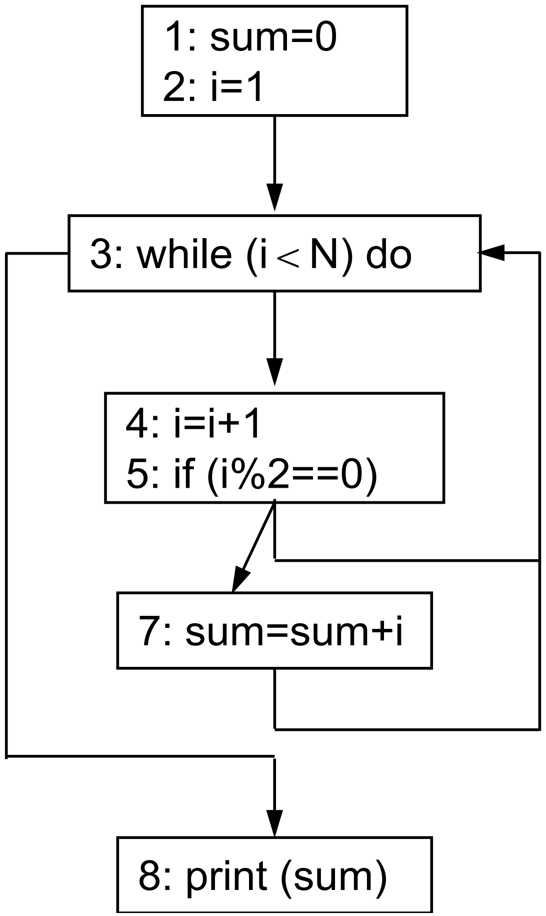
☞ CD(3)=3, **tricky!**



Note: Control Dependence is not Syntactically Explicit

- Y is control-dependent on X iff X directly determines whether Y executes.
  - X is not strictly post-dominated by Y.
  - There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y.

1	sum=0
2	i=1
3	while (i < N) do
4	i=i+1
5	if (i%2==0)
6	continue
7	sum=sum+i endwhile
8	print (sum)



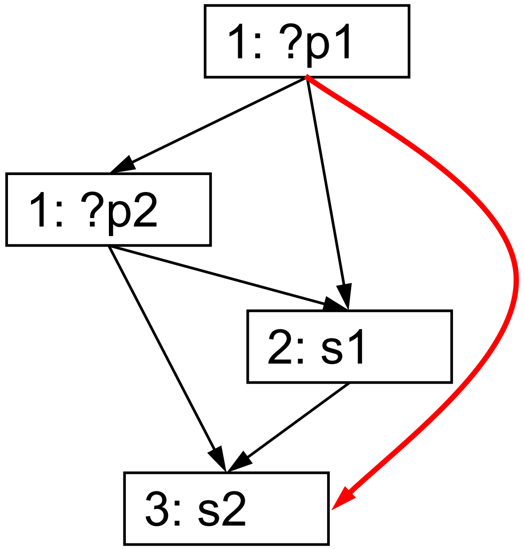
Control Dependence is Tricky!

- Y is control-dependent on X iff X directly determines whether Y executes.
  - X is not strictly post-dominated by Y.
  - There exists a path from X to Y s.t. every node in the path other than X and Y is post-dominated by Y.
- Can one statement control depends on two predicates?

1	if (p1 || p2)
2	s1;
3	s2;

What if?

1	if (p1 && p2)
2	s1;
3	s2;



The Use of PDG

- A program dependence graph consists of control dependence graph and data dependence graph.

- Why it is so important to software reliability?
  - In debugging, what could possibly induce the failure?
  - In security.

```
p=getpassword();
...
send (p);
```

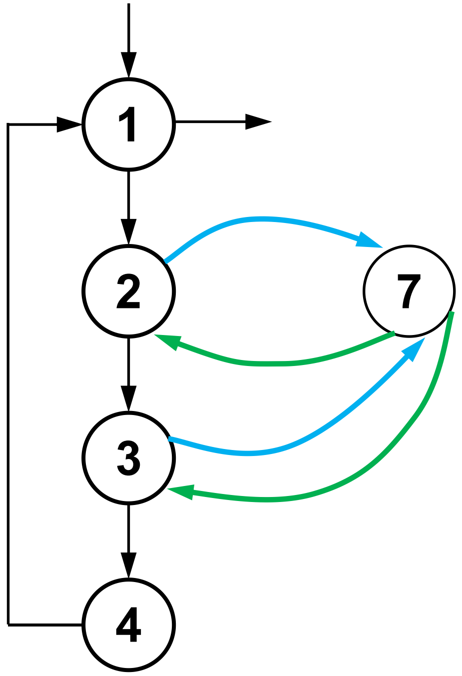
```
p=getpassword();
...
if (p==”zhang”) {
    send (m);
}
```

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(234);
4	x[i]=t1+t2+t3;
5	}
6	int f (int v) {
7	reture (v+1);
8	}



Part Three

03

Call Graph

Call Graph (CG)

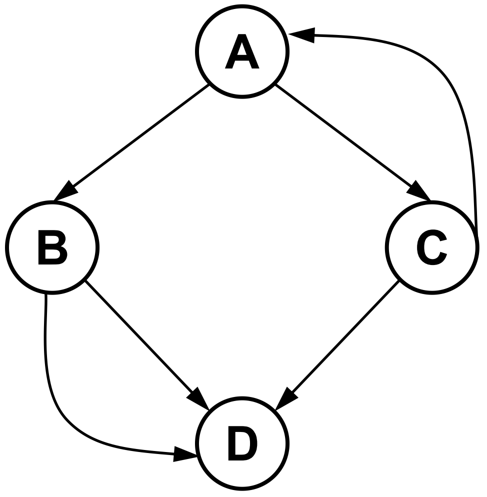
- Y Each node represents a function; each edge represents a function invocation.

```
void A() {
    B();
    C();
}

void C() {
    D();
    A();
}
```

```
viod B() {
L1: D();
L2: D();
}

void D() {
}
```



MONTANA STATE UNIVERSITY

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

10/16/2025