# EXERCISE #21

## *POINTS-TO ANALYSIS REVIEW*

### Write your name and answer the following on a piece of paper

*Draw the points-to relation from Andersen's analysis on the following function*

```
int main(){
        p = &x;
        if (x == 0){
                r = &p;
        } else {
                q = &y;
        }
        s = &q;
        r = s;
}
```

p := &x
r := &p
q := &y
s := &q
r := s

| Assignment | Constraint |
|---|---|
| a = &b | a ⊇ {b} |
| a = b | a ⊇ b |
| a = *b | a ⊇ *b |
| *a = b | *a ⊇ b |

# ADMINISTRIVIA AND ANNOUNCEMENTS

# DEPENDENCE RELATIONS

EECS 677: Software Security Evaluation

Drew Davidson

# CLASS PROGRESS

ANALYSIS UNDERLYING OUR ENFORCEMENT/EVALUATION NEEDS

# LAST TIME: POINTS-TO ANALYSIS

### REVIEW: LAST LECTURE

## CONSIDER WHERE EACH POINTER MIGHT POINT

**Efficiency vs Precision**

- Dataflow facts as points-to sets
- Andersen's algorithm:
  - Flow-insensitive
  - Worst-case cubic time
- Steensgard's algorithm:
  - Flow-insensitive
  - Near-linear time

# LAST TIME: ANDERSEN'S ALGORITHM

## REVIEW: LAST LECTURE

### IN PRACTICE

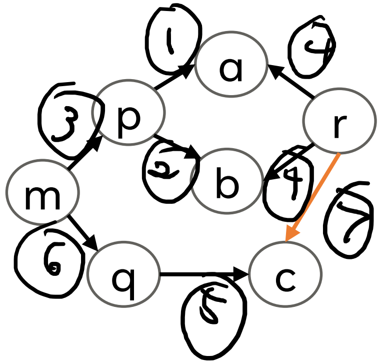
**Step 1**
List pointer-related operations
**Step 2**
Induce set of subset constraints
**Step 3**
Solve system of constraints

### REACHABILITY FORMULATION

**Step 1**
List pointer-related operations
**Step 2**
Saturate points-to graph
**Step 3**
Compute node reachability

| Program | Constraints | Initial | Final |
|---|---|---|---|
| p = &a | p ⊇ {a} | pts(a) = { } | pts(a) = { } |
| p = &b | p ⊇ {b} | pts(b) = { } | pts(b) = { } |
| m =&p; | m ⊇ {p} | pts(m) = { } | pts(m) = { p, q } |
| r = *m; | r ⊇ *m | pts(p) = { } | pts(p) = { a, b } |
| q = &c; | q ⊇ {c} | pts(q) = { } | pts(q) = { c } |
| m = &q | m ⊇ {q} | pts(r) = { } | pts(r) = { a, b, c } |

| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | a ⊇ {b} | loc(b) ∈ pts(a) |
| a = b | a ⊇ b | pts(a) ⊇ pts(b) |
| a = *b | a ⊇ *b | ∀v∈pts(b). pts(a) ⊇ pts(v) |
| *a = b | *a ⊇ b | ∀v∈pts(a). pts(v) ⊇ pts(b) |



# LAST TIME: ANDERSEN'S ALGORITHM

## REVIEW: LAST LECTURE

### In Practice

**Step 1**

List pointer-related operations

**Step 2**

Induce set of subset constraints

**Step 3**

Solve system of constraints

### Reachability formulation

**Step 1**

List pointer-related operations

**Step 2**

Saturate points-to graph

**Step 3**

Compute node reachability

# LAST TIME: STEENGARD'S ALGORITHM

## REVIEW: LAST LECTURE

### In Practice

**Step 1**

List pointer-related operations

**Step 2** equality

Induce set of ~~subset~~ constraints

**Step 3**

Solve system of constraints

### Reachability formulation

**Step 1**

List pointer-related operations

**Step 2** 1-out

Saturate points-to graph

**Step 3**

Compute node reachability

Andersen's

| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | $a \supseteq \{b\}$ | $loc(b) \in pts(a)$ |
| a = b | $a \supseteq b$ | $pts(a) \supseteq pts(b)$ |
| a = *b | $a \supseteq *b$ | $\forall v \in pts(b).\ pts(a) \supseteq pts(v)$ |
| *a = b | $*a \supseteq b$ | $\forall v \in pts(a).\ pts(v) \supseteq pts(b)$ |

Steengaard's

| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | $a \supseteq \{b\}$ | $loc(b) \in pts(a)$ |
| a = b | a = b | $pts(a) = pts(b)$ |
| a = *b | a = *b | $\forall v \in pts(b).\ pts(a) = pts(v)$ |
| *a = b | *a = b | $\forall v \in pts(a).\ pts(v) = pts(b)$ |

# LAST TIME: STEENGARD'S ALGORITHM

## REVIEW: LAST LECTURE

### IN PRACTICE

**Step 1**

List pointer-related operations

**Step 2**

Induce set of equality constraints

**Step 3**

Solve system of constraints

### REACHABILITY FORMULATION

**Step 1**

List pointer-related operations

**Step 2**

Saturate 1-out points-to graph

**Step 3**

Compute node reachability

Andersen's

| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | $a \supseteq \{b\}$ | $loc(b) \in pts(a)$ |
| a = b | $a \supseteq b$ | $pts(a) \supseteq pts(b)$ |
| a = *b | $a \supseteq *b$ | $\forall v \in pts(b).\ pts(a) \supseteq pts(v)$ |
| *a = b | $*a \supseteq b$ | $\forall v \in pts(a).\ pts(v) \supseteq pts(b)$ |

Steengaard's

| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | $a \supseteq \{b\}$ | $loc(b) \in pts(a)$ |
| a = b | a = b | $pts(a) = pts(b)$ |
| a = *b | a = *b | $\forall v \in pts(b).\ pts(a) = pts(v)$ |
| *a = b | *a = b | $\forall v \in pts(a).\ pts(v) = pts(b)$ |

# LAST TIME: STEENGARD'S ALGORITHM

## REVIEW: LAST LECTURE

### In Practice

**Step 1**

List pointer-related operations

**Step 2**

Induce set of equality constraints

**Step 3**

Solve system of constraints

### Reachability formulation

**Step 1**

List pointer-related operations

**Step 2**

Saturate 1-out points-to graph

**Step 3**

Compute node reachability

p = &a

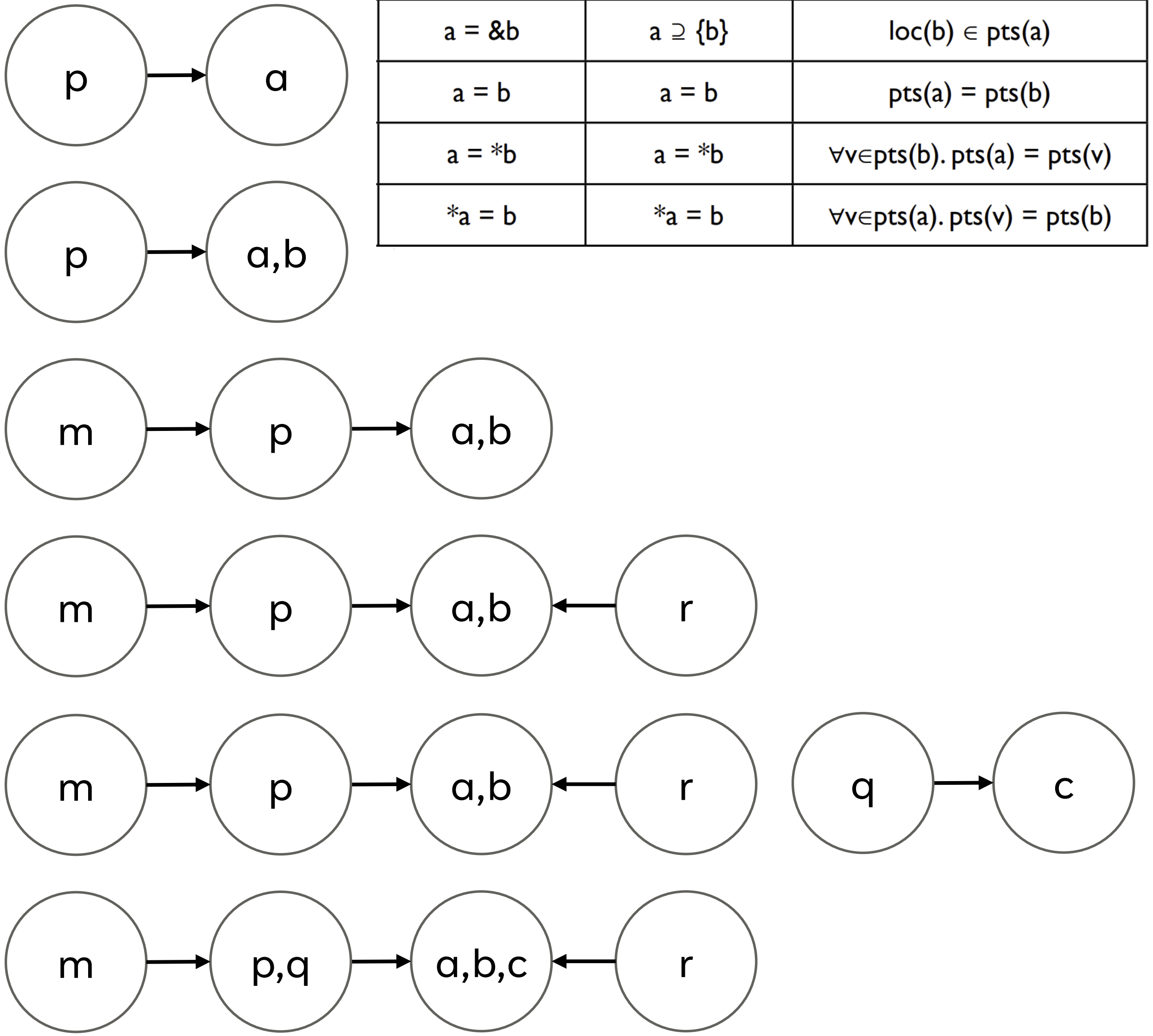
p = &b

m = &p

r = *m

q = &c

m = &q




| Assignment | Constraint | Meaning |
|---|---|---|
| a = &b | a ⊇ {b} | loc(b) ∈ pts(a) |
| a = b | a = b | pts(a) = pts(b) |
| a = *b | a = *b | ∀v∈pts(b). pts(a) = pts(v) |
| *a = b | *a = b | ∀v∈pts(a). pts(v) = pts(b) |

# THIS LECTURE

FOCUSING OUR ANALYSIS ON PARTICULAR PROGRAM ASPECTS OF INTEREST

# LECTURE OUTLINE

- Dependence relations
- Control dependence graphs (CDGs)

# WHY DOES STATEMENT X DO Y?

## DEPENDENCE RELATIONS

### Often interested in a subset of program behavior

What “influenced” statement X?

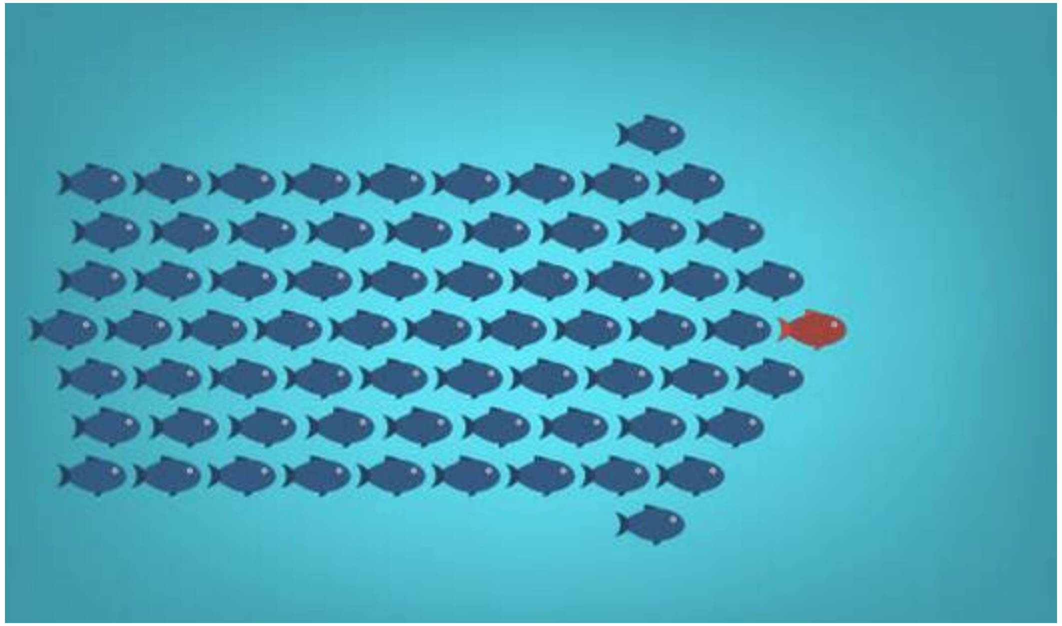
What did statement X “influence”?

### Useful in a variety of contexts

Consider a pointer... what might make it null?

### Assisting Scalabilty

Don’t get lost in details unrelated to my pointer / bug



# APPLICATIONS
## DEPENDENCE RELATIONS

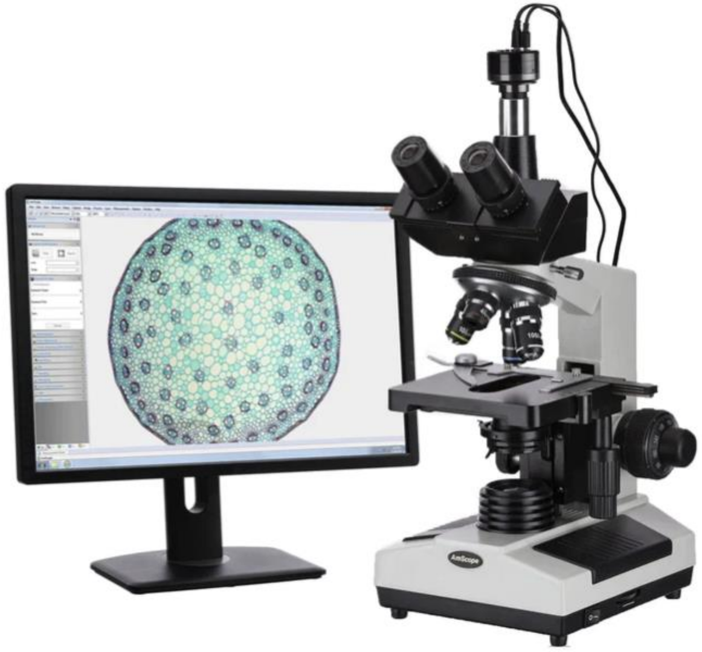
### Proactive

What causes my program to crashing?

Does this statement leak data?

### Reactive

Zoom in on a suspicious operation



# LECTURE OUTLINE

- Dependence relations
- Control Dependence
- Data Dependence

# EXAMPLE

## DEPENDENCE RELATIONS

### CONSIDER THE FOLLOWING PROGRAM

Under what circumstances are various lines executed?

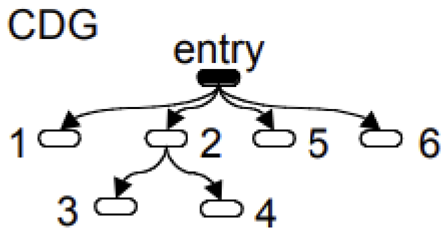
```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

Print statement at line 3: *control-dependent* on line 2

Print statement at line 5: *control-dependent* on entry to function

### CAPTURE THESE INSIGHTS IN A DATA STRUCTURE

The control dependence graph



# BUILDING THE CDG

## DEPENDENCE RELATIONS

### Intuition on Control Dependence

What is the closest statement are you guaranteed to execute?

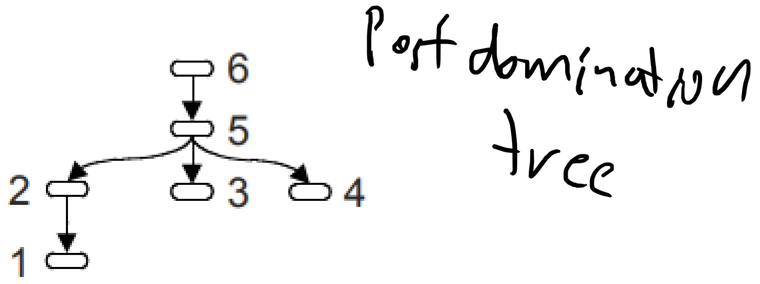
### Postdomination

A Statement Y **postdominates** X ⇔ every path from X is guaranteed to go through Y, denoted X in PDOM(Y)

*Intuitively, X is "destined" to meet Y*

A Statement Y **immediately postdominates** X ⇔ X in PDOM(Y) and there is no intervening postdominator, denoted X in IPDOM(Y)

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```



# BUILDING THE CDG
## DEPENDENCE RELATIONS

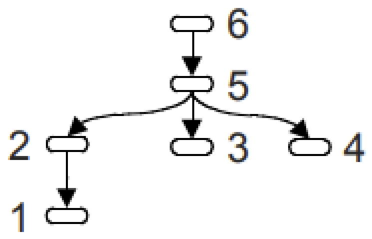
### (IMMEDIATE) FORWARD DOMINATORS

X IN IPDOM(Y) ⇔ Y in IFDOM(X)

```
1: READ i;
2: if ( i == 1)
3:    PRINT "hi!"
   else
4:    i = 1;
5: PRINT i;
6: end
```

2 in IPDOM 5

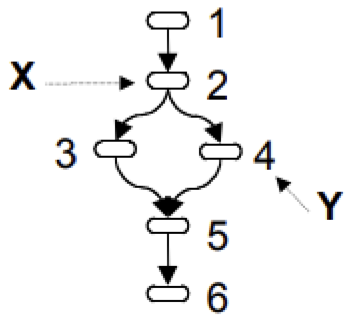
5 in IFDOM 2

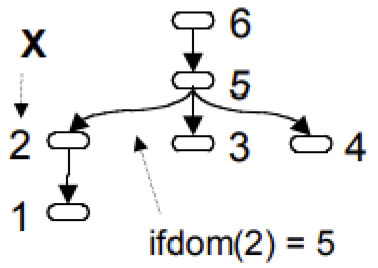


# BUILDING THE CDG

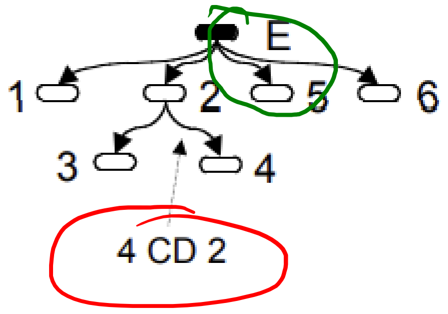
## DEPENDENCE RELATIONS

Y is control dependent on X ⇔ there is a path in the CFG from X to Y that doesn't contain the immediate forward dominator of X




```
1:  READ i;
2:  if ( i == 1)
3:      PRINT "hi!"
    else
4:      i = 1;
5:  PRINT i;
6:  end
```




what 2 does matters for reaching 4

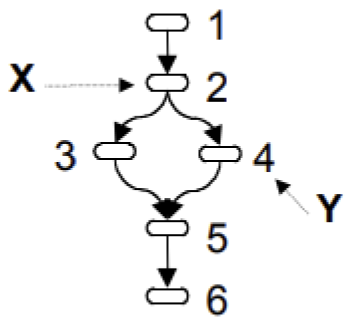
## LECTURE OUTLINE

- Dependence relations
- Control Dependence
- Data Dependence

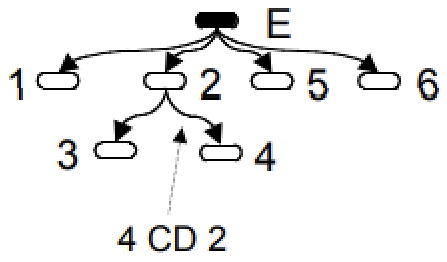
# DATA DEPENDENCE

## DEPENDENCE RELATIONS

Influence is more than control, it’s also what values mattered to your behavior




```
1:  READ i;
2:  if ( i == 1)
3:      PRINT “hi!”
    else
4:      i = 1;
5:  PRINT i;
6:  end
```




Note here: 1 might have set (a value used by) 5, but it’s not control dependent!

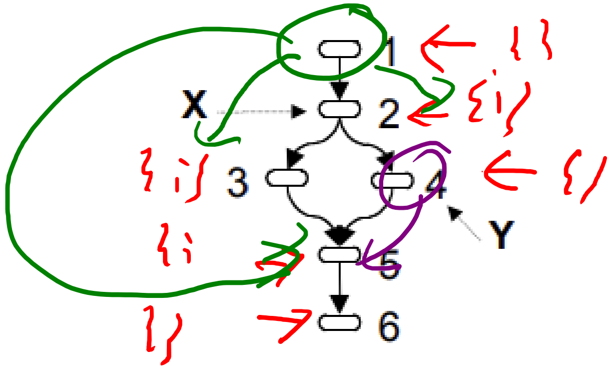
# THE DATA DEPENDENCE GRAPH

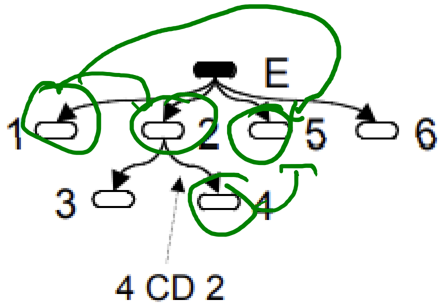
## DEPENDENCE RELATIONS

Depiction of the *reaching definitions* of each statement

```
1:  READ i;
2:  if ( i == 1)
3:      PRINT "hi!"
    else
4:      i = 1;
5:  PRINT i;
6:  end
```

PRINT i




CDG + DDG = PDG

# NEXT TIME
## DEPENDENCE RELATIONS

### Consider the Program Slice

Forward Slice: the portions of the program a given statement influences

Backwards Slice: the portions of the program influenced by a give statement

# WRAP-UP