EXERCISE #20

POINTS-TO ANALYSIS REVIEW

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

Draw the exploded supergraph for the following program:

```
class SupClass{
public:
        virtual int fun(SupClass * in){
                in->fun();
                return 1;
        }
};
class SubA : public SupClass{
        int fun(SupClass * in){
                return 2;
        }
};
class SubB : public SupClass{
        void fun(SupClass * in){
                return 3;
        }
};
int main(){
        SupClass * s = new SubA();
        s->fun();
}
```

ADMINISTRIVIA AND ANNOUNCEMENTS

POINTS-TO ANALYSIS

EECS 677: Software Security Evaluation

Drew Davidson

CLASS PROGRESS

ANALYSIS UNDERLYING OUR ENFORCEMENT NEEDS

LAST TIME: INTERPROCEDURAL ANALYSIS

REVIEW: LAST LECTURE

CONSIDER THE EFFECT OF MULTIPLE FUNCTIONS

Simplistic

– Function overturn all global / aliased facts

Supergraph / Context String

– 1-CFA (use a call-chain of 1)

Summary Information

- Use GMOD and GREF to cut down on imprecision

direct

&

indirect

```c
int g;
int v1;
int v2;
int fn(int a){
        if (a > 1){
                return 0;
        }
        return 1;
}

int main(){
        g = 1;
        v1 = fn(1);
        v1 = fn(v1);
        v2 = v1 / g;
        return v2 / v1;
}
```

LAST TIME: GMOD & GREF COMPUTATION

REVIEW: LAST LECTURE

Globals Only

Step 1

Compute IMOD and IREF

Step 2

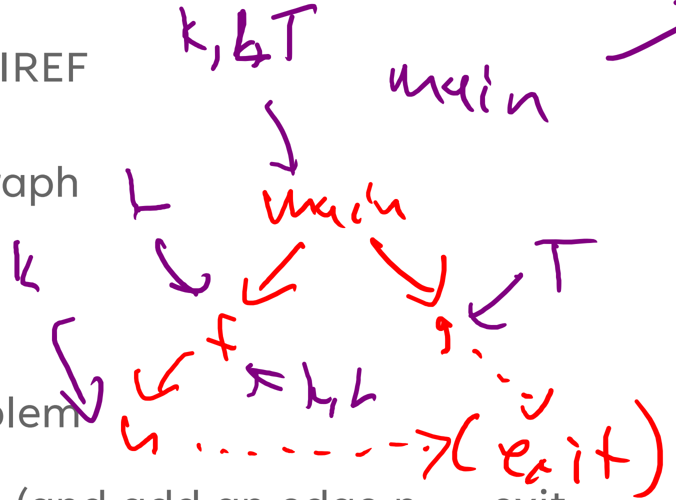
Build (simple) call graph

Step 3

Collapse Cycles

Step 4

Solve a fixpoint problem



Add a new exit node (and add an edge n ⟶ exit for each node n with no outgoing edge.

A lattice element is a set of (global) variables.

The lattice meet is set union.

The "initial" dataflow fact for both GMOD and GREF (the fact that holds at the exit node) is the empty set.

for all nodes n, the dataflow functions for n are:

GMOD: fn(S) = S U IMOD(n)

GREF: fn(S) = S U IREF(n)

main → fn:

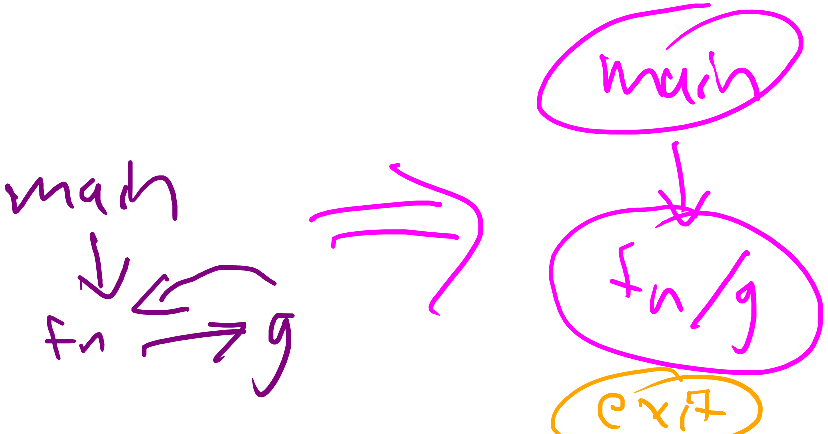
$a = \perp$

$b = a$

$\rightleftarrows$ g

$IMOD_{fn}(a, b)$

$IREF_{fn}(a)$

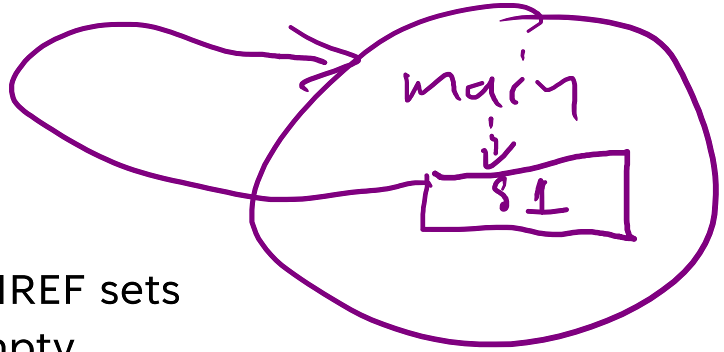


LAST TIME: GMOD & GREF COMPUTATION

REVIEW: LAST LECTURE

GLOBALS, LOCALS & VALUE-PASSING

GREF will change, GMOD doesn't need to change



Step 1
New IREF: include locals and formals

Step 2
Build call-site multigraph

~~**Step 3**~~
~~Collapse Cycles~~

Step 3
Solve a fixpoint problem

Init all node GREF sets to their IREF sets
Init all call site GREF sets to empty
Put all nodes and call sites on a worklist
Iterate until the worklist is empty.

Each time a node n is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then add all call sites to n to the worklist (if not already there).
Similarly, each time a call site s is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then the node that contains s is added to the worklist (if not already there).

GREF(n) = (U all GREF(s) s.t. s is a call site in n) U IREF(n)
GREF(s) = GREF(called node m) with all formals mapped back to the corresponding actuals

LAST TIME: GMOD & GREF COMPUTATION

REVIEW: LAST LECTURE

GLOBALS, LOCALS & VALUE-PASSING

GREF will change, GMOD doesn't need to change

Init all node GREF sets to their IREF sets
Init all call site GREF sets to empty
Put all nodes and call sites on a worklist
Iterate until the worklist is empty.

Each time a node n is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then add all call sites to n to the worklist (if not present).
Each time a call site s is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then the node that contains s is added to the worklist (if not present).

LAST TIME: GMOD & GREF COMPUTATION

REVIEW: LAST LECTURE

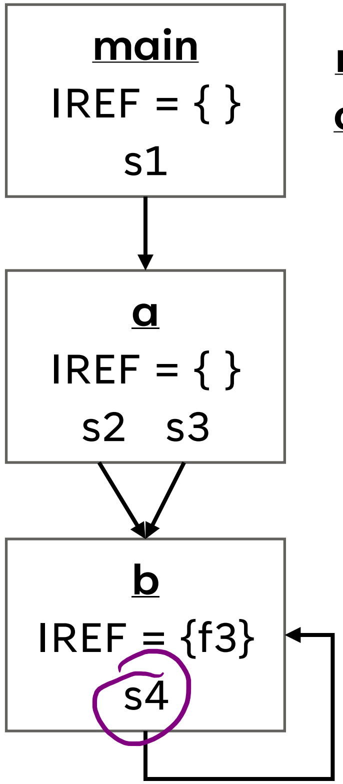
GLOBALS, LOCALS & VALUE-PASSING

GREF will change, GMOD doesn't need to change

Init all node GREF sets to their IREF sets
Init all call site GREF sets to empty
Put all nodes and call sites on a worklist
Iterate until the worklist is empty.

Each time a node n is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then add all call sites to n to the worklist (if not present).
Each time a call site s is removed from the worklist, its current GREF set is computed. If that set doesn't match its previous value, then the node that contains s is added to the worklist (if not present).

```
void main() {
S1: call a(v1)
}
void a( f1 ) {
S2: call b(v2, v3)
S3: call b(v4, v5)
}
void b( f2, f3 ) {
  print f3;
S4: call b(g1, g2)
}
```



node or call site	**GREF set**
main	g2
a	g2, v3, v5
b	f3, g2
s1	g2
s2	v3, g2
s3	v5, g2
s4	g2

OVERVIEW

WE'VE SEEN THE NECESSITY OF MULTI-FUNCTION ANALYSIS IN REAL-WORLD PROGRAMS

TIME TO CONSIDER HOW IT IS DONE

BACK TO DATAFLOW

INTERPROCEDURAL ANALYSIS

```
int fn(int a, int b){
        if (a){
                b = gbl;
        }       a
        disclose(b);
}
```

What are the (possible) values of a and gbl?

```
Object * x;
...
int main(){
        x = read_secret();
        mystery();
        disclose(x);

}
```

What is the effect of mystery on x's taintedness?

LECTURE OUTLINE

- May-point v Must-point
- Andersen's Analysis
- Steensgard's Analysis

POINTERS: LOVE TO HATE 'EM

MAY-POINT AND MUST-POINT

may-point(p) : the set of locations to which p <u>might</u> refer

must-point(p) : the set of locations to which p <u>must</u> refer

"EASY" SOLUTION: DATAFLOW FOR MAY-POINT

MAY-POINT AND MUST-POINT

✗ Interprocedural analysis on this scale is likely infeasible

✓ Flow-insensitive

LECTURE OUTLINE

- May-point v Must-point
- Andersen’s Analysis
- Steensgard’s Analysis

SUBSET CONSTRAINTS

ANDERSEN'S ANALYSIS

A Flow-insensitive Algorithm

Each statement adds a constraint over the points-to sets

End up with a (solvable) system of constraints

Program

```
p = &a;
q = p;
p = &b;
r = p;
```

SUBSET CONSTRAINTS

ANDERSEN'S ANALYSIS

Constraint type	Assignment	Constraint	Meaning
Base	a = &b	$a \supseteq \{b\}$	$loc(b) \in pts(a)$
Simple	a = b	$a \supseteq b$	$pts(a) \supseteq pts(b)$
Complex	a = *b	$a \supseteq *b$	$\forall v \in pts(b).\ pts(a) \supseteq pts(v)$
Complex	*a = b	$*a \supseteq b$	$\forall v \in pts(a).\ pts(v) \supseteq pts(b)$

SUBSET CONSTRAINTS

ANDERSEN'S ANALYSIS

A Flow-insensitive Algorithm

Each statement adds a constraint over the points-to sets

End up with a (solvable) system of constraints

Program	**Constraints**	**Initial**	**Final**
p = &a;	p ⊇ {a}	pts(p) = $\cancel{\emptyset}$ {a,b}	pts(p) = {a,b}
q = p;	q ⊇ p	pts(q) = $\cancel{\emptyset}$ ~~{a}~~ {a,b}	pts(q) = {a,b}
p = &b;	p ⊇ {b}	pts(r) = $\cancel{\emptyset}$ {a,b}	pts(r) = {a,b}
r = p;	r ⊇ p	pts(a) = ∅	pts(a) = ∅
		pts(b) = ∅	pts(b) = ∅

ANOTHER EXAMPLE

ANDERSEN'S ANALYSIS

A FLOW-INSENSITIVE ALGORITHM

Each statement adds a constraint over the points-to sets

End up with a (solvable) system of constraints

update everything in {a} to include {b}

Program	Constraints	Initial	Final
p = &a	$p \supseteq \{a\}$	pts(p) = { a }	pts(p) = { a }
q = &b	$q \supseteq \{b\}$	pts(q) = { b }	pts(q) = { b }
*p = q;	~~$*p \supseteq q$~~	pts(r) = { c }	pts(r) = { c }
r = &c;	$r \supseteq \{c\}$	pts(s) = ~~∅~~ {a}	pts(s) = { a }
s = p;	$s \supseteq p$	pts(t) = ~~∅~~ {c}	pts(t) = { b, c }
t = *p;	$t \supseteq *p$	pts(a) = ~~∅~~ ~~c~~ b, c	pts(a) = { b, c }
*s = r;	$*s \supseteq r$	pts(b) = ∅	pts(b) = ∅
		pts(c) = ∅	pts(c) = ∅

SOLVING CONSTRAINTS
ANDERSEN'S ANALYSIS

Graph closure on the subset relation

Assgmt.	Constraint	Meaning	Edge
a = &b	$a \supseteq \{b\}$	$b \in pts(a)$	no edge
a = b	$a \supseteq b$	$pts(a) \supseteq pts(b)$	$b \rightarrow a$
a = *b	$a \supseteq *b$	$\forall v \in pts(b).\ pts(a) \supseteq pts(v)$	no edge
*a = b	$*a \supseteq b$	$\forall v \in pts(a).\ pts(v) \supseteq pts(b)$	no edge

OVERHEAD

ANDERSEN'S ANALYSIS

Worst Case: Cubic Time

That's not great

Optimization: Cycle elimination

Detect and collapse SCCs in the points-to relation

LECTURE OUTLINE

- May-point v Must-point
- Andersen’s Analysis
- Steensgard’s Analysis

AN ALTERNATIVE APPROACH

STEENSGARD'S ANALYSIS

Aim for Near-Linear-time points-to Analysis

Going to require us to reduce our search-space somewhat

Intuition: Equality constraints

Do away with the notion of subsets

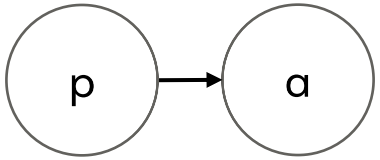
EQUALITY CONSTRAINTS

STEENSGARD'S ANALYSIS

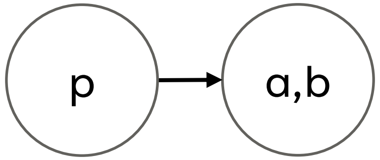
Constraint type	Assignment	Constraint	Meaning
Base	a = &b	$a \supseteq \{b\}$	$loc(b) \in pts(a)$
Simple	a = b	a = b	$pts(a) = pts(b)$
Complex	a = *b	a = *b	$\forall v \in pts(b).\ pts(a) = pts(v)$
Complex	*a = b	*a = b	$\forall v \in pts(a).\ pts(v) = pts(b)$

EQUALITY CONSTRAINTS
STEENSGARD'S ANALYSIS

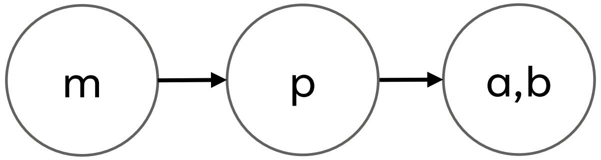
p = &a



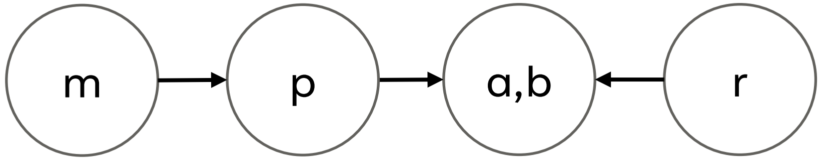
p = &b



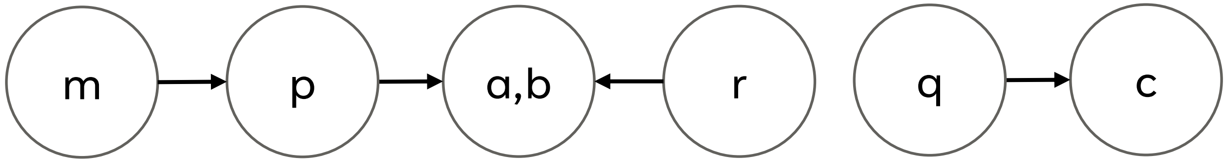
m = &p



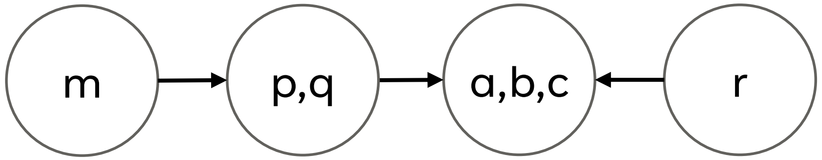
r = *m



q = &c



m = &q

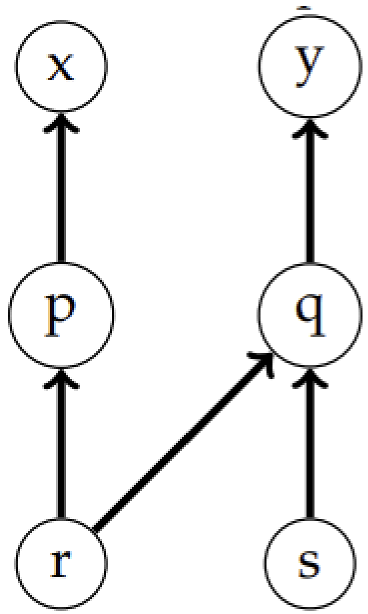


EQUALITY CONSTRAINTS

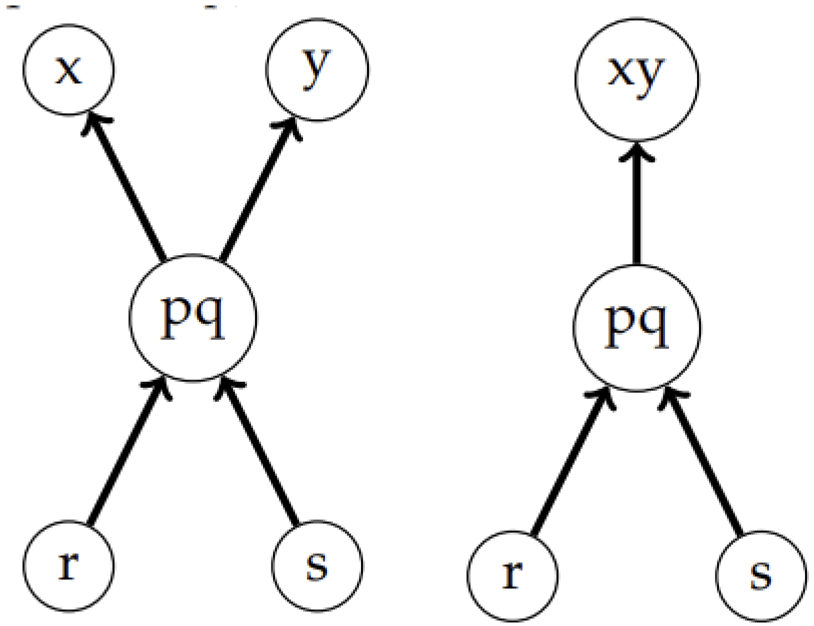
STEENSGARD'S ANALYSIS

$$
\begin{aligned}
1:&\quad p := \&x\\
2:&\quad r := \&p\\
3:&\quad q := \&y\\
4:&\quad s := \&q\\
5:&\quad r := s
\end{aligned}
$$

Andersen's



Steensgard's



WRAP-UP