Everything Else About Data Flow Analysis

Flow- and Context-Sensitivity
Logical Representation

Pointer Analysis

Interprocedural Analysis

Three Levels of Sensitivity

- In DFA so far, we have cared about where in the program we are.
  - Called *flow-sensitivity*.
- But we didn't care how we got there.
  - Called *context-sensitivity*.
- We could even care about neither.
  - Example: where could x ever be defined in this program?

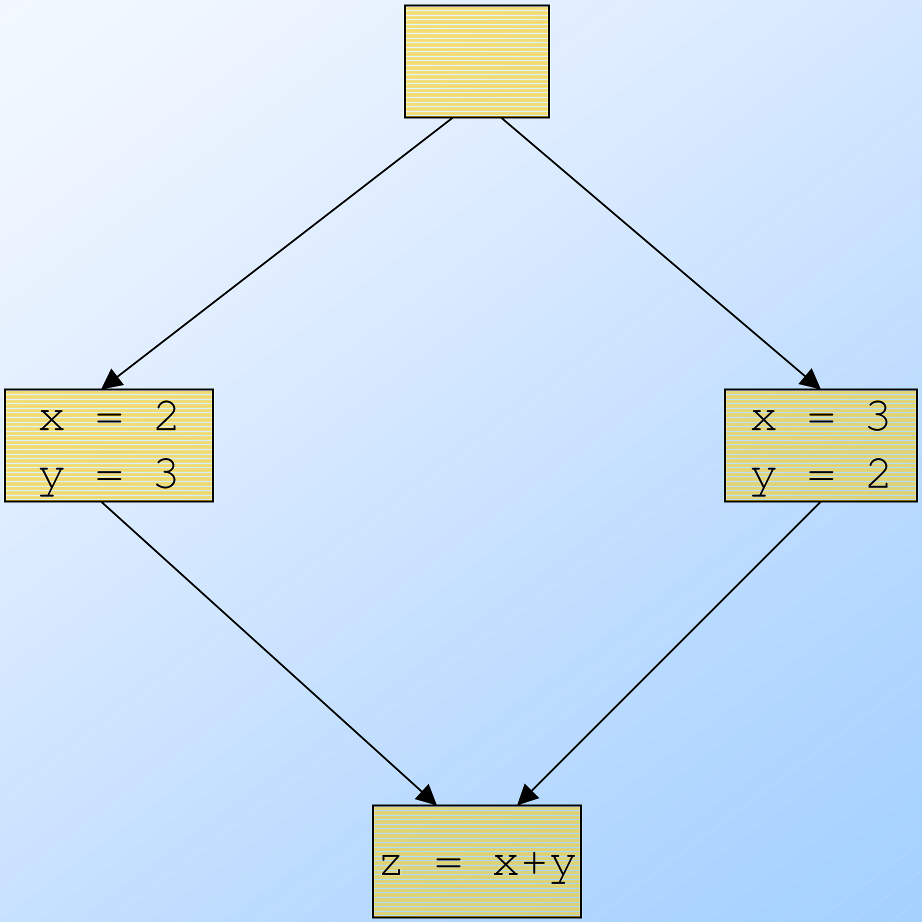
Flow/Context Insensitivity

- Not so bad when program units are small (few assignments to any variable).
- Example: Java code often consists of many small methods.
  - Remember: you can distinguish variables by their full name, e.g., class.method.block.identifier.

Context Sensitivity

- Can distinguish paths to a given point.
- Example: If we remembered paths, we would not have the problem in the constant-propagation framework where x+y = 5 but neither x nor y is constant over all paths.

The Example Again



An Interprocedural Example

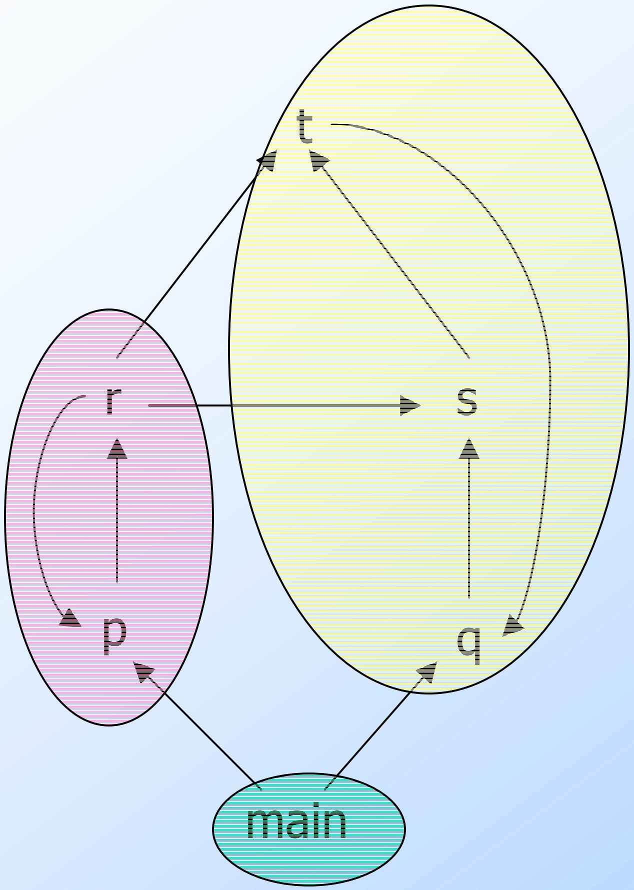
```
int id(int x) {return x;}
void p() {a=2; b=id(a);…}
void q() {c=3; d=id(c);…}
```

- If we distinguish p calling id from q calling id, then we can discover b=2 and d=3.
- Otherwise, we think b, d = {2, 3}.

Context-Sensitivity --- (2)

- Loops and recursive calls lead to an infinite number of contexts.
- Generally used only for interprocedural analysis, so forget about loops.
- Need to collapse strong components of the calling graph to a single group.
- "Context" becomes the sequence of groups on the calling stack.

Example: Calling Graph



Contexts:

Green
Green, pink
Green, yellow
Green, pink, yellow

Comparative Complexity

- Insensitive: proportional to size of program (number of variables).
- Flow-Sensitive: size of program, squared (points times variables).
- Context-Sensitive: worst-case exponential in program size (acyclic paths through the code).

Logical Representation

- We have used a set-theoretic formulation of DFA.
  - IN = set of definitions, e.g.
- There has been recent success with a logical formulation, involving predicates.
- Example: Reach(d,x,i) = "definition d of variable x can reach point i."

Comparison: Sets Vs. Logic

- Both have an efficiency enhancement.
  - Sets: bit vectors and boolean ops.
  - Logic: BDD's, incremental evaluation.
- Logic allows integration of different aspects of a flow problem.
  - Think of PRE as an example. We needed 6 stages to compute what we wanted.

Datalog --- (1)

Atom = Reach(d,x,i)

Predicate

Arguments:
variables or constants

Literal = Atom or NOT Atom

Rule = Atom :- Literal & … & Literal

Make this
atom true
(the *head*).

The *body* :
For each assignment of values
to variables that makes all these
true …

Example: Datalog Rules

```
Reach(d,x,j) :- Reach(d,x,i) &
                StatementAt(i,s) &
                NOT Assign(s,x) &
                Follows(i,j)
Reach(s,x,j) :- StatementAt(i,s) &
                Assign(s,x) &
                Follows(i,j)
```

Datalog --- (2)

- Intuition: subgoals in the body are combined by "and" (strictly speaking: "join").
- Intuition: Multiple rules for a predicate (head) are combined by "or."

Datalog --- (3)

- Predicates can be implemented by relations (as in a database).
- Each tuple, or assignment of values to the arguments, also represents a propositional (boolean) variable.

EDB Vs. IDB Predicates

- Some predicates come from the program, and their tuples are computed by inspection.
  - Called *EDB*, or *extensional database* predicates.
- Others are defined by the rules only.
  - Called *IDB*, or *intensional database* predicates.

Iterative Algorithm for Datalog

- Start with the EDB predicates = "whatever the code dictates," and with all IDB predicates empty.
- Repeatedly examine the bodies of the rules, and see what new IDB facts can be discovered from the EDB and existing IDB facts.

Seminaive Evaluation

- Remember that a new fact can be inferred by a rule in a given round only if it uses in the body some fact discovered on the previous round.
- Same idea applies to set-theoretic DFA, but the bit-vector implementation makes the idea ineffective.

Example: Seminaive

Path(x,y) :- Arc(x,y)

Path(x,y) :- Path(x,z) & Path(z,y)

```
NewPath(x,y) = Arc(x,y); Path(x,y) = ∅;
while (NewPath != ∅) do {
  NewPath(x,y) = {(x,y) | NewPath(x,z)
     && Path(z,y) || Path(x,z) &&
     NewPath(z,y)} - Path(x,y);
  Path(x,y) = Path(x,y) ∪ NewPath(x,y);
}
```

Stratification

- A risk occurs if there are negated literals involved in a recursive predicate.
  - Leads to oscillation in the result.
- Requirement for *stratification* :
  - Must be able to order the IDB predicates so that if a rule with P in the head has NOT Q in the body, then Q is either EDB or earlier in the order than P.

Example: Nonstratification

P(x) :- E(x) & NOT P(x)

- If E(1) is true, is P(1) true?
- It is after the first round.
- But not after the second.
- True after the third, not after the fourth,…

Example: Stratification

- PRE is an example of stratified logic.
- Each of the analyses depends on previous ones, some negatively.
- But there is no recursion or iteration involving negation of the data-flow values we are trying to compute.

PRE Example

Anticipated(B) :- (some rules)

Available(B) :- (some other rules)

Earliest(B) :- Anticipated(B) & NOT Available(B)

Postponable(B) :- (some rules involving Earliest)

Latest(B) :- (some rules involving Earliest,
Postponable, NOT Earliest,
and NOT Postponable)

Used(B) :- (rules involving Latest)

New Topic: Pointer Analysis

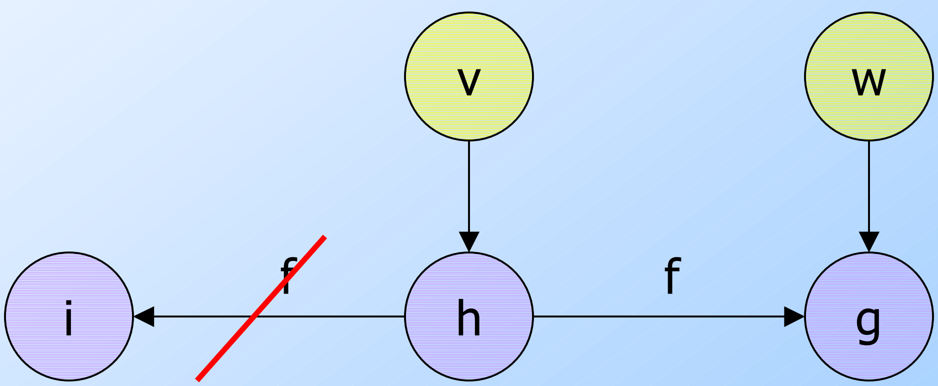
- We shall consider Andersen's formulation of Java object references.
- Flow/context insensitive analysis.
- Cast of characters:
  1. Local variables, which point to:
  2. Heap objects, which may have fields that are references to other heap objects.

Representing Heap Objects

- A heap object is named by the statement in which it is created.
- Note many run-time objects may have the same name.
- Example: `h: T v = new T;` says variable v can point to (one of) the heap object(s) created by statement h.

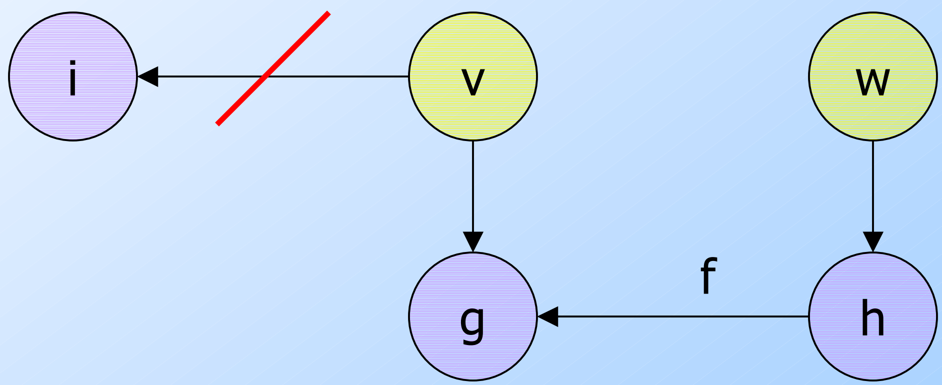
Other Relevant Statements

- `v.f = w` makes the f field of the heap object h pointed to by v point to what variable w points to.



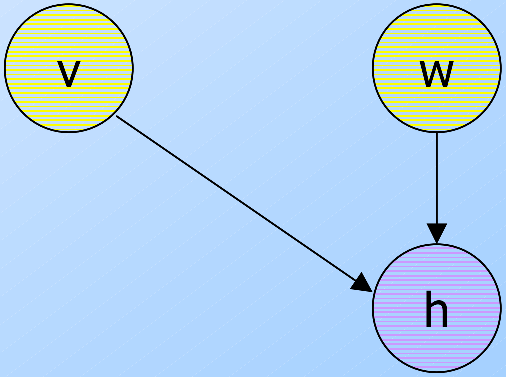
Other Statements --- (2)

- `v = w.f` makes v point to what the f field of the heap object h pointed to by w points to.



Other Statements --- (3)

- `v = w` makes v point to whatever w points to.
  - *Interprocedural Analysis* : Also models copying an actual parameter to the corresponding formal or return value to a variable.



EDB Relations

- The facts about the statements in the program and what they do to pointers are accumulated and placed in several EDB relations.
- Example: there would be an EDB relation Copy(To,From) whose tuples are the pairs (v,w) such that there is a copy statement `v=w`.

Convention for EDB

- Instead of using EDB relations for the various statement forms, we shall simply use the quoted statement itself to stand for an atom derived from the statement.
- Example: "v=w" stands for Copy(v,w).

IDB Relations

- Pts(V,H) will get the set of pairs (v,h) such that variable v can point to heap object h.
- Hpts(H1,F,H2) will get the set of triples (h,f,g) such that the field f of heap object h can point to heap object g.

Datalog Rules

1. Pts(V,H) :- "H: V = new T"
2. Pts(V,H) :- "V=W" & Pts(W,H)
3. Pts(V,H) :- "V=W.F" & Pts(W,G) & Hpts(G,F,H)
4. Hpts(H,F,G) :- "V.F=W" & Pts(V,H) & Pts(W,G)

Example

```
T p(T x) {
  h: T a = new T;
     a.f = x;
     return a;
}
void main() {
  g: T b = new T;
     b = p(b);
     b = b.f;
}
```

Apply Rules Recursively --- Round 1

```
T p(T x) {h: T a = new T;
    a.f = x; return a;}
void main() {g: T b = new T;
    b = p(b); b = b.f;}
```

Pts(a,h)

Pts(b,g)

Apply Rules Recursively --- Round 2

```
T p(T x) {h: T a = new T;
    a.f = x; return a;}
void main() {g: T b = new T;
    b = p(b); b = b.f;}
```

Pts(a,h)
Pts(b,g)

Pts(x,g)
Pts(b,h)

Apply Rules Recursively --- Round 3

```
T p(T x) {h: T a = new T;
    a.f = x; return a;}
void main() {g: T b = new T;
    b = p(b); b = b.f;}
```

Pts(a,h)
Pts(b,g)

Pts(x,g)
Pts(b,h)

Pts(x,h)

Hpts(h,f,g)

Apply Rules Recursively --- Round 4

```
T p(T x) {h: T a = new T;
    a.f = x; return a;}
void main() {g: T b = new T;
    b = p(b); b = b.f;}
```

Pts(a,h)

Pts(b,g)

Pts(x,g)

Pts(b,h)

Pts(x,h)　　Hpts(h,f,g)

Hpts(h,f,h)

Extension to Flow Sensitivity

- IDB predicates need additional arguments B, I.
  - B = block number.
  - I = position within block, 0, 1,…, *n* for *n* -statement block.
    - Position 0 is before first statement, position 1 is between 1st and 2nd statement, etc.

Example of Rules: Flow Sensitive Pointer Analysis

Pts(V,H,B,I+1) :- "B,I: H: V = new T"

Pts(V,G,B,I+1) :- "B,I: W = new T" & V != W & Pts(V,G,B,I)

I is local, H is a global index of object-creating statements.

Pts(V,G,B,I+1) :- "B,I: W.f = X" & Pts(V,G,B,I)

Notice W=V OK

Pts(V,G,B,0) :- Pts(V,G,C,*n*) & "C is a predecessor block of B with *n* statements"

Handles all control-flow information within the flow graph. Hpts similar.

Adding Context Sensitivity

- Include a component C = context.
  - C doesn't change within a function.
  - Call and return can extend the context if the called function is not mutually recursive with the caller.

Example of Rules: Context Sensitive

Pts(V,H,B,I+1,C) :- "B,I: V=W" &
Pts(W,H,B,I,C)

Pts(X,H,B0,0,D) :- Pts(V,H,B,I,C) &
"B,I: call P(…,V,…)" &
"X is the corresponding actual to V in P"
& "B0 is the entry of P" &
"context D is C extended by P"