More Data Flow Analysis

Last Time

- Data Flow Analysis
- Data Flow Frameworks
- Constant Propagation Framework
- Reaching Definitions

Today

Iterative Worklist Algorithm via Reaching Definitions

- Why it works
- What it computes

Work List Iterative Algorithm

for $v \in V$
    IN(v) $= \emptyset$
    OUT(v) $=$ GEN(v)
endfor
worklist $\leftarrow v \in V$
while (*worklist* $\neq \emptyset$)
    pick and remove a node v from *worklist*
    oldout(v) $=$ OUT(v)
    IN(v) $= \bigcup$ (OUT(p)), $p \in$ PRED(v)
    OUT(v) $=$ GEN(v) $\bigcup$ (IN(v) - KILL(v))
    if oldout(v) $\neq$ OUT(v) then
        worklist $\leftarrow$ *worklist* $\cup$ SUCC(v)
endwhile

Work List Iterative Algorithm

Questions

- Does this always terminate?
- How fast (or slow) is it?
- What answer does it compute?
- How fast can we make it?

Termination

Why does the iterative data flow algorithm terminate?

Sketch of proof for reaching definitions

1. each node is initialized to $\emptyset$
2. a definition has only one statement that generates it
3. $\mathcal{F}$ is associative $\Rightarrow$ $\mathcal{F}$ is monotone
   $\Rightarrow$ each $x \in$ *Reaching definitions* can be added once
4. $N * (E+1)$ trips to take a definition to every node

Consequence of finite descending chain property

Question: How do we generalize this proof?

Correctness and Quality of Solution

Does it compute the answer we want?

Definition: For each basic block b

MOP(b) $= \sqcap f_p(\top)$, for all paths p "reaching" b

- Paths that reach a block are reachable in the control flow graph, which may be conservative.
- Perfect Solution = meet over *real* paths taken during program execution
- MOP $\leq$ Perfect Solution
- In some sense, MOP is best feasible solution
- Not guaranteed to achieve MOP solution
- MOP is undecidable, even for monotonic framework
- Reduction to Modified Post's Correspondence Problem

Reference: "Monotone Data Flow Analysis Frameworks," J.B. Kam and J.D. Ullman, Acta Informatica 7:305-317, 1977.

Quality of Solution

Maximal Fixed Point (MFP)

- Any iterative data-flow problem that satisfies admissible function requirements when it converges to a solution and terminates, will have reached a Maximal Fixed Point solution.
- MFP is unique, regardless of order of propagation
- If distributive, MFP = MOP
- Otherwise, MFP $\leq$ MOP
- So, MFP $\leq$ MOP $\leq$ Perfect Solution

How fast can we make the iterative algorithm?

Execution time of iterative framework

- For each basic block: # successors (predecessors) + constant bit vector operations
- Number of visits to basic block: length of longest acyclic path
- What is the complexity equation? $O(n^2)$

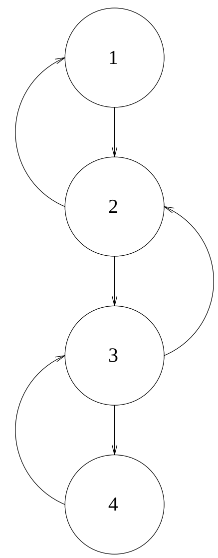
Where is unnecessary work being performed?

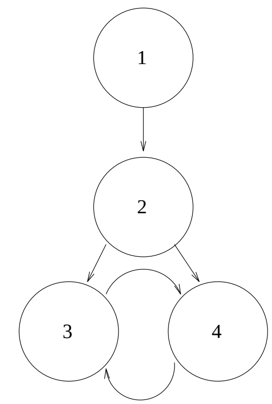
- Iteration over every node on each pass.
- Testing for altered sets on each pass.
- Extra pass to detect stabilization.

Problem: Nodes may be visited in any order

Reference: "Analysis of a Simple Algorithm for Global Flow Problems," M. Hecht and J. Ullman, Proceedings of the ACM Conference on Principles of Programming Languages, Oct. 1973.

Examples





How fast can we make the iterative algorithm?

To avoid unnecessary work:

- Bound number of visits by visiting a node roughly *after* all its predecessors

  (reverse PostOrder for forward data-flow problem; conceptually, PostOrder for backward problem).

- Change to algorithm:

```
change = true;
while (change)
   change = false;
   for each basic block in rPostOrder:
     solve for b
     if (old ≠ new) change = true;
   end for
end while
```

- How does this improve performance?

PostOrder and Reverse PostOrder

Step1: PostOrder

```
main()
    count = 1;
    Visit (root);

Visit(n)
    mark n as visited
    for each successor s of n not yet visited
        Visit(s);
    PostOrder(n) = count;
    count = count + 1;
```

Step 2: rPostOrder

```
for each node n
    rPostOrder(n) = NumNodes - PostOrder(n)
```

Depth-first search $\approx$ rPostOrder

"Rapid" Data-Flow Problems

Necessary and sufficient condition for "rapid" stabilization of iterative framework:

$$\forall f,g \in \mathcal{F}, \forall x \in L, \quad fg(\bot) \succeq g(\bot) \sqcap f(x) \sqcap x$$

An equivalent condition:

$$\forall f \in \mathcal{F}, \forall x \in L, \quad f(x) \succeq x \sqcap f(\top)$$

For Reaching Definitions:

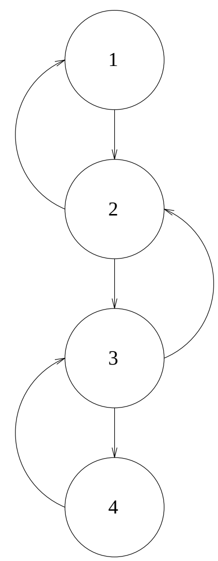
$$\begin{array}{rcl} f(x) & \stackrel{?}{\succeq} & x \sqcap f(\top) \\ a \cup (x-b) & \stackrel{?}{\succeq} & x \cup (a \cup (\top - b)) \\ a \cup (x-b) & \stackrel{?}{\succeq} & x \cup a \\ x - b & \succeq & x \end{array}$$

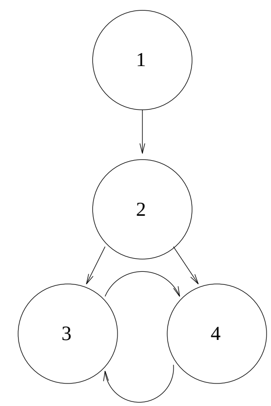
$\Rightarrow$ Reaching definitions is *rapid*

"Rapid" data-flow problems stabilize in at most d(G)+2 passes over the control flow graph, (iff for forward problems you use rPostOrder, and backwards problems use PostOrder).

Loop Interconnectiveness

- $d(G)$ = maximum number of retreating edges on any acyclic path on graph G
- d is the degree of *loop interconnectiveness*
- d is unique for *reducible* flow graphs





Node Listing

key: iterate exactly enough times to transmit information along any *simple paths* of *CFG*.

A node listing [Kennedy 75]

$$l = (v_1, v_2, \ldots v_m)$$

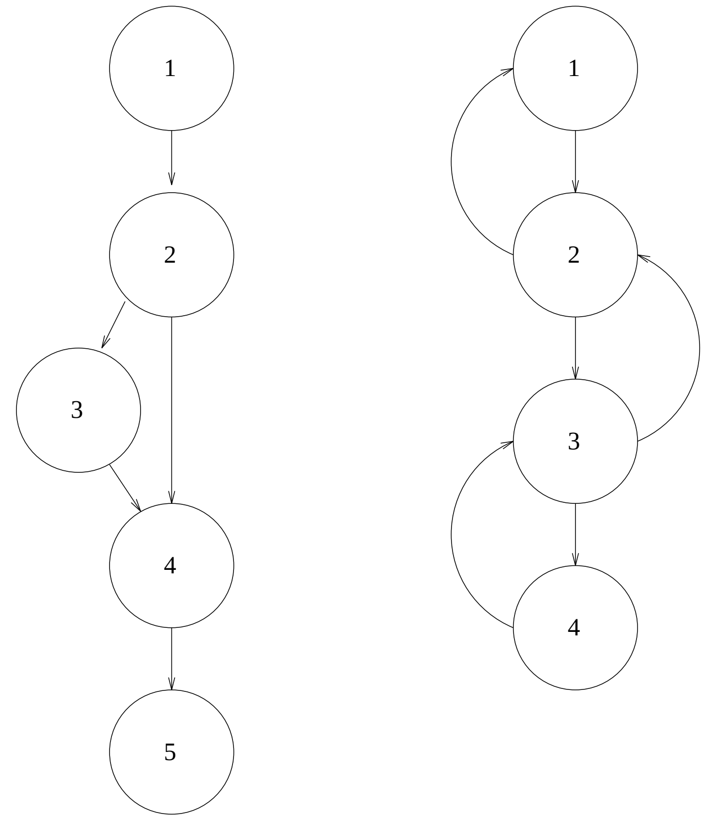
requires that every simple path in *CFG* is in sequence in l, *i.e.,* if $p = (x_1, x_2, \ldots x_k)$ is a simple path then

$$(\exists j_1, j_2, \ldots j_k) | j_i < j_{i+1} \text{ and } x_i = n_{j_i}, 1 \le i \le k,$$

$\forall$ *CFG*, $\exists$ node listing of length $\le n^2, n = |V|$

For a large class of graphs (which ones?), there is an $O(n)$ listing.

Node Listing



"Rapid" Data-Flow Problems

Property of "rapid" data-flow problems

- the "rapid" condition means information stabilizes in two passes around a loop
- d+1 iterations to propagate data, 1 iteration to detect stability
- in practice, $d(G)$ is less than 3 [Knuth]
- in practice, iterative algorithms make a small number of passes
- each pass computes
  - $\mathbf{O}(E)$ meets (sets of size $|defs|$)
  - and $\mathbf{O}(N)$ other operations
- Effectively $O(n)$ complexity

Data-flow hierarchy

"rapid" $\subset$ "fast" $\subset$ distributive $\subset$ monotone

References: "Global Data Flow Analysis and Iterative Algorithms," J.B. Kam and J.D. Ullman, Journal of the ACM, 23(1), Jan. 1976.

Analysis of Data-flow Frameworks

Key things to look for in a data-flow framework

- the domain and its size
- size of a single fact
- forward or backward problem
- model of characteristic function

Representation

- Sets represented by *bit vector*
- **Size of each bit vector:**
  - Available Expressions: # distinct expressions in program
  - Reaching Definitions: # definitions in program
  - Live Variable Analysis: # variables in program

Complexity

- distinguish bit-vector steps from logical steps
- watch out for complex mappings (GEN$\rightarrow$ KILL)

Summary

- Iterative data-flow framework used to solve global data-flow problems.
- Use semi-lattice to represent facts.
- Analysis on semi-lattice with finite descending chains and monotone data-flow framework guarantees termination.
- Monotonic data-flow framework guarantees MFP solution reached.
- Distributivity property necessary to guarantee MOP solution reached.
- rPostOrder (or PostOrder) for "rapid" data-flow problems guarantees bound of O(n(d+2)) complexity.

Next Time

- Live Variable Analysis (backward problem)
- Constant Propagation

Reading: Wegman & Zadeck, Constant Propagation with Conditional Branches, ACM Transactions on Programming Languages and Systems, 13:2, April 1991, pp. 181-210.