Comp 512
Spring 2011

Beyond Regional Optimization

Global Data Flow Analysis with Applications

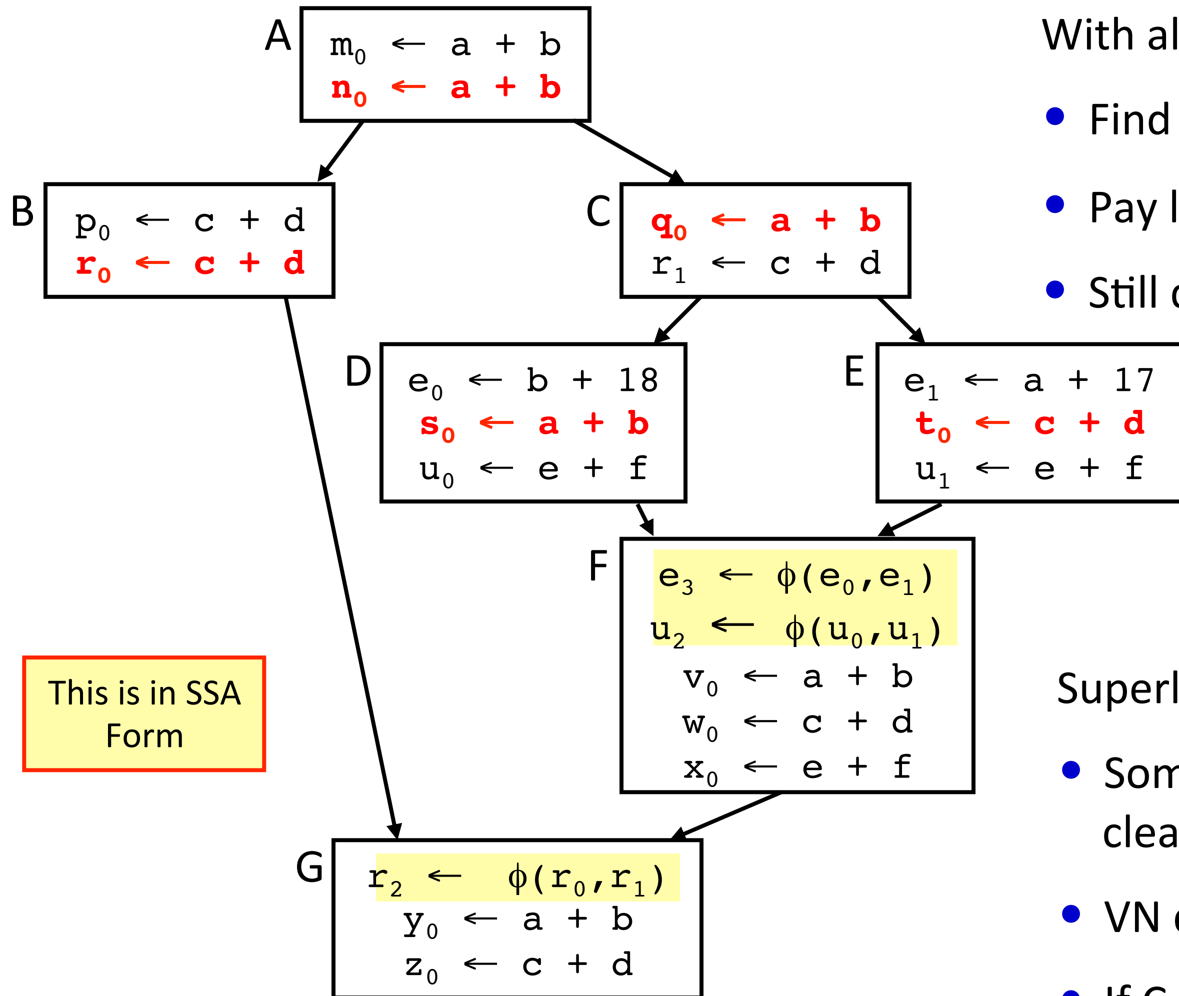
Last Lecture

- Extended Basic Blocks
- Superlocal Value Numbering
  - > Treat each path as a single basic block
  - > Use a scoped hash table & SSA names to make it efficient
- Loop Unrolling as an example of a loop-based optimization

This Lecture

- Dominator Trees
  - → Computing dominator information
  - → Global data-flow analysis
- Dominator-based Value Numbering
  - → Enhance the Superlocal Value Numbering algorithm so that it can cover more blocks
  - → Not truly a global optimization, but a good application of dominators

Superlocal Value Numbering



With all the bells & whistles

- Find more redundancy
- Pay little additional cost
- Still does nothing for F & G

Superlocal techniques

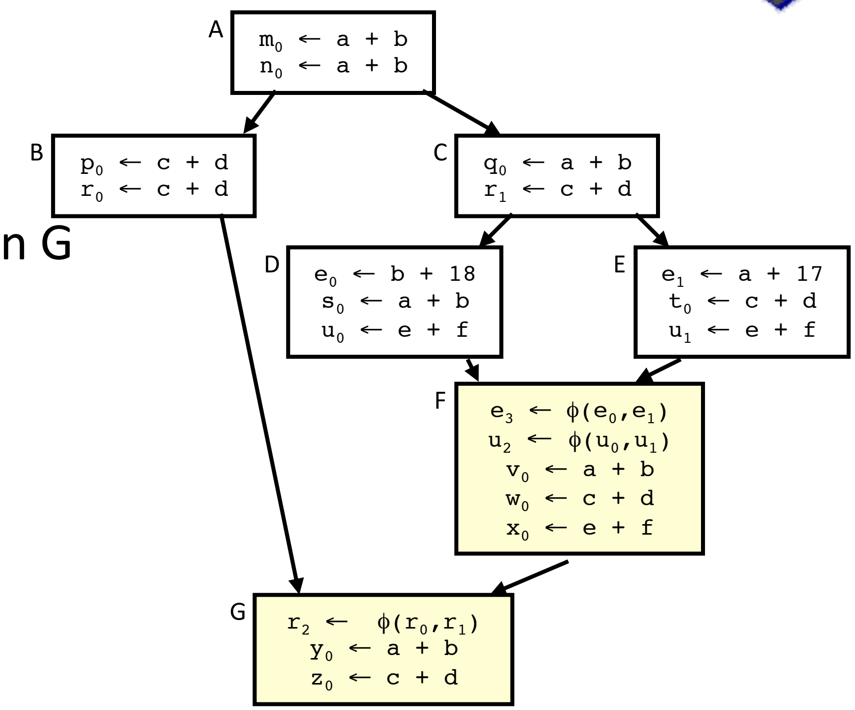
- Some local methods extend cleanly to superlocal scopes
- VN does not back up
- If C adds to A, it's a problem

What About Larger Scopes?

We have not helped with F or G

- Multiple predecessors

- Must decide what facts hold in F and in G
  - → For G, combine B & F?
  - → Merging state is expensive
  - → Fall back on what's known



Dominators

Definitions

In a flow graph, x dominates y if and only if every path from the entry of the control-flow graph to the node for y includes x

- By definition, x dominates x
- We associate a DOM set with each node
- $|\text{DOM}(x)| \geq 1$

Immediate dominator

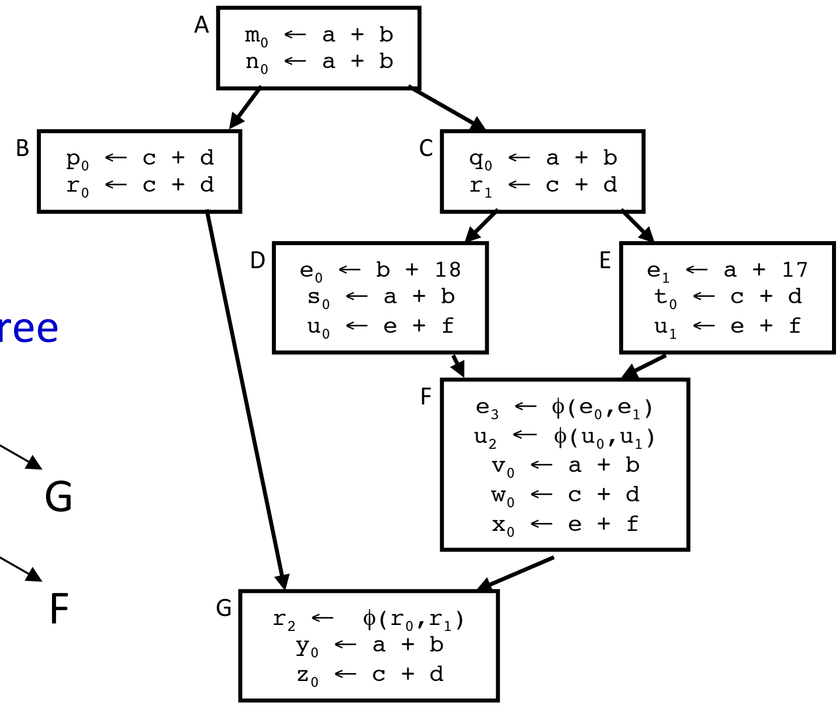
- For any node x, there must be a y in DOM(x) closest to x
  - → Unless $x = n_0$, $x \neq \text{IDOM}(x)$
- We call this y the immediate dominator of x
- As a matter of notation, we write this as IDOM(x)

Original idea: R.T. Prosser. "Applications of Boolean matrices to the analysis of flow diagrams," *Proceedings of the Eastern Joint Computer Conference, Spartan Books, New York, pages 133-138, 1959.*

Dominators

Dominators have many uses in analysis & transformation

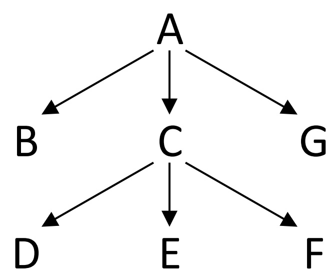
- Finding loops
- Building SSA form
- Making code motion decisions



Dominator sets

Block	Dom	IDom
A	A	–
B	A,B	A
C	A,C	A
D	A,C,D	C
E	A,C,E	C
F	A,C,F	C
G	A,G	A

Dominator tree



We'll look at how to compute dominators later

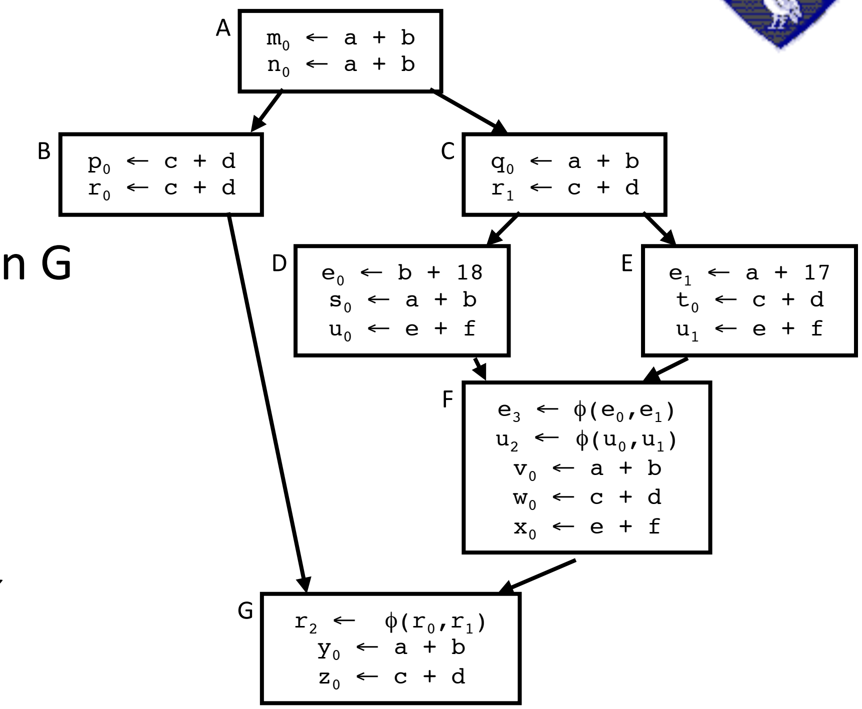
Dominators Can Improve Superlocal Value Numbering

We have not helped with F or G

- Multiple predecessors

- Must decide what facts hold in F and in G
  - → For G, combine B & F?
  - → Merging state is expensive
  - → Fall back on what's known

- Can use table from IDOM(*x*) to start *x*
  - → Use C for F and A for G
  - → Imposes a Dom-based application order



Leads to <u>D</u>ominator <u>VN</u> <u>T</u>echnique (DVNT)

Dominator Value Numbering

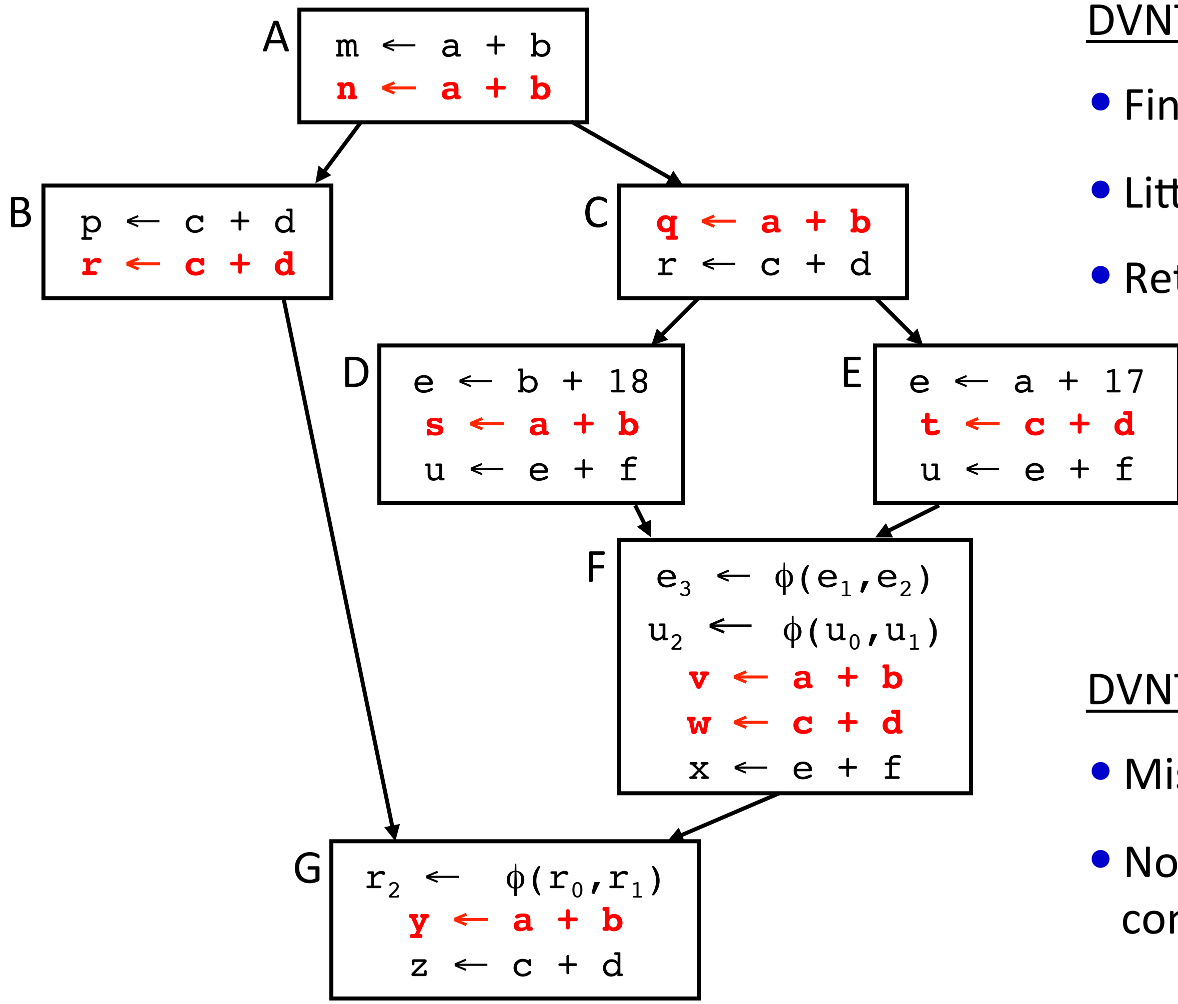
The DVNT Algorithm

- Use superlocal algorithm on extended basic blocks
  - → Retain use of scoped hash tables & SSA name space
- Start each node with table from its IDOM
  - → DVNT generalizes the superlocal algorithm
- No values flow along back edges (*i.e.,* around loops)
- Constant folding, algebraic identities as before

Larger scope leads to (*potentially*) better results

- → LVN + SVN + good start for EBBs missed by SVN

Dominator Value Numbering



DVNT advantages

- Find more redundancy
- Little additional cost
- Retains *online* character

DVNT shortcomings

- Misses some opportunities
- No loop-carried CSEs or constants

Computing Dominators

Critical first step in SSA construction and in DVNT

- A node n dominates m iff n is on every path from n_0 to m
  - → Every node dominates itself
  - → n's <u>immediate dominator</u> is its closest dominator, IDOM(n)†

$$\text{DOM}(n_0) = \{ n_0 \}$$

$$\text{DOM}(n) = \{ n \} \cup \left(\bigcap_{p \in preds(n)} \text{DOM}(p) \right)$$

Initially, DOM(n) = N, $\forall\, n \neq n_0$.
Can do better.

Computing DOM

- These simultaneous set equations define a simple problem in data-flow analysis
- Equations have a unique fixed point solution
- An iterative fixed-point algorithm will solve them quickly

†IDOM(n) ≠ n, unless n is n_0, by convention.

Round-robin Iterative Algorithm

$\text{DOM}(b_0) \leftarrow \emptyset$

for i $\leftarrow$ 1 to N
    $\text{DOM}(b_i) \leftarrow \{$ *all nodes in graph* $\}$

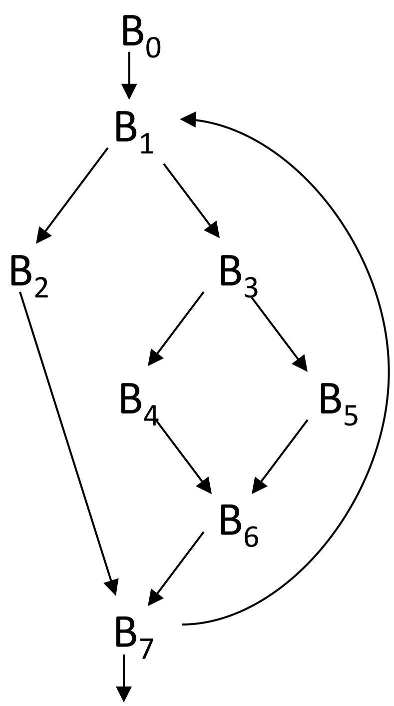
change $\leftarrow$ true

while (change)
    change $\leftarrow$ false
    for i $\leftarrow$ 0 to N
        $\text{TEMP} \leftarrow \{ i \} \cup (\cap_{x \in pred(b)} \text{DOM}(x))$
    if $\text{DOM}(b_i) \neq \text{TEMP}$ then
        change $\leftarrow$ true
        $\text{DOM}(b_i) \leftarrow \text{TEMP}$

Termination

- Makes sweeps over the nodes
- Halts when some sweep produces no change

Example



Flow Graph

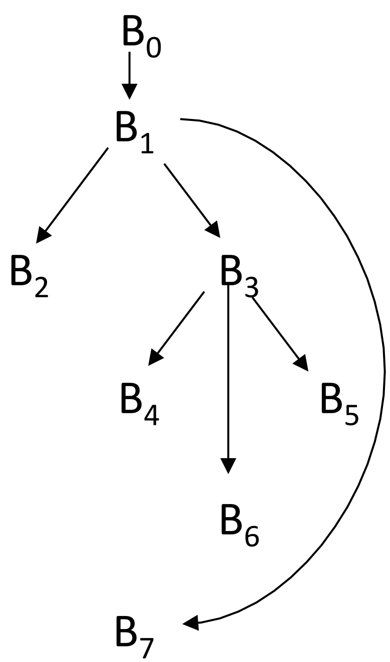
Progress of iterative solution for Dom

Iter-	Dom(*n*)							
ation	0	1	2	3	4	5	6	7
0	0	*N*	*N*	*N*	*N*	*N*	*N*	*N*
1	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7
2	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7

Results of iterative solution for Dom

	0	1	2	3	4	5	6	7
Dom	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7
IDom	0	0	1	1	3	3	3	1

Example



Dominance Tree

Progress of iterative solution for DOM

Iter-ation	DOM(*n*) 0	1	2	3	4	5	6	7
0	0	*N*	*N*	*N*	*N*	*N*	*N*	*N*
1	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7
2	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7

Results of iterative solution for DOM

	0	1	2	3	4	5	6	7
DOM	0	0,1	0,1,2	0,1,3	0,1,3,4	0,1,3,5	0,1,3,6	0,1,7
IDOM	0	0	1	1	3	3	3	1

There are asymptotically faster algorithms.

With the right data structures, the iterative algorithm can be made extremely fast.

See Cooper, Harvey, & Kennedy, on the web site, or algorithm in Chapter 9 of EaC2e.

Aside on Data-Flow Analysis

The iterative DOM calculation is an example of data-flow analysis

- Data-flow analysis is a collection of techniques for *compile-time reasoning about the run-time flow of values*
- Data-flow analysis almost always operates on a graph
  - → Problems are trivial in a basic block
  - → Global problems use the control-flow graph (or derivative)
  - → Interprocedural problems use call graph (or derivative)
- Data-flow problems are formulated as simultaneous equations
  - → Sets attached to nodes and edges
  - → One solution technique is the iterative algorithm
- Desired result is usually *meet over all paths (MOP) solution*
  - → *"What is true on every path from the entry node?"*
  - → *"Can this event happen on any path from the entry?"*

Related to safety

Aside on Data-Flow Analysis

Why did the iterative algorithm work?

Termination

- The DOM sets are initialized to the (finite) set of nodes
- The DOM sets shrink monotonically
- The algorithm reaches a *fixed point* where they stop changing

Correctness

- We <u>can</u> prove that the fixed point solution is also the MOP
- That proof is beyond today's lecture, but we'll revisit it

Efficiency

- The round-robin algorithm is <u>*not*</u> particularly efficient
- Order in which we visit nodes is important for efficient solutions