COMP 512
Rice University
Spring 2015

Overview of Optimization, 3

Iterative Global Data Flow Analysis, in depth

Citation numbers refer to entries in the EaC2e bibliography.

Computing Available Expressions

The Big Picture

1. Build a control-flow graph
2. Gather the initial data (*local data*) data — **DEEXPR**(*b*) & **EXPRKILL**(*b*) — and initialize the **AVAIL** sets (*unknowns*) at each block
3. Evaluate the equation at each node, then repeat to fixed point
   - Propagates information around the graph'
   - Annotates each block with its correct and complete **AVAIL** set

$$\textbf{AVAIL}(b) = \bigcap_{x \in preds(b)} (\textbf{DEEXPR}(x) \cup (\textbf{AVAIL}(x) \cap \overline{\textbf{EXPRKILL}(x)}))$$

Most data-flow problems are solved in, essentially, the same way

Round-robin Iterative Algorithm

AVAIL(b_0) ← Ø

for i ← 1 to N

  AVAIL(b_i) ← { *all expressions* }

change ← true

while (change)

  change ← false

  for i ← 0 to N

    $\textbf{TEMP} \leftarrow \bigcap_{x \in preds(b_i)} (\textbf{DEEXPR}(x) \cup (\textbf{AVAIL}(x) \cap \overline{\textbf{EXPRKILL}(x)}))$

    if **AVAIL**(b_i) ≠ **TEMP** then

      change ← true

      AVAIL(b_i) ← **TEMP**

The round-robin solver is easier to analyze than a worklist solver.

Questions that we should ask:

- Termination: does it halt?
- Correctness: what answer does it produce?
- Speed: how quickly does it find that answer?

Data-flow Analysis

Definition

Data-flow analysis is a collection of techniques for *compile-time* reasoning about the *run-time* flow of values

- Almost always involves building a graph
  - ♦ Problems are trivial on a basic block
  - ♦ Global problems ⇒ control-flow graph (or derivative)
  - ♦ Whole program problems ⇒ call graph (or derivative)
- Usually formulated as *simultaneous equations* over *sets of values*
  - ♦ Sets attached to nodes and / or edges
  - ♦ Semilattice to describe values
  - ♦ We solved **AVAIL** with an iterative fixed-point algorithm
- Desired result is usually *meet over all paths* solution
  - ♦ "What is true on every path from the entry?"
  - ♦ "Can this happen on any path from the entry?"
  - ♦ Related to the safety of optimization (*how we use the results*)

MOP ≅ meet over all paths solution
LFP ≅ least fixed-point solution
MFP ≅ maximal fixed-point solution

Data-flow Analysis

Limitations

1. Precision – these algorithms are precise *"up to symbolic execution"*
   - ♦ Assume all paths are taken
2. Solution – cannot afford to compute **MOP** solution
   - ♦ Large class of problems where **MOP** = **MFP** = **LFP**
   - ♦ Not all problems of interest are in this class
3. Arrays – classical analysis treats them naively
   - ♦ Represent whole array with a single fact
4. Pointers – difficult (*and expensive*) to analyze
   - ♦ Imprecision rapidly adds up
   - ♦ Need to ask the right questions

The Good News:
Simple problems can carry us pretty far

Summary

For scalar values, we can quickly solve simple problems

Data-flow Analysis

Semilattice

A **semilattice** is a set L and a meet operation $\wedge$ such that, $\forall\, a, b, \& \; c \in L$:

1. $a \wedge a = a$
2. $a \wedge b = b \wedge a$
3. $a \wedge (b \wedge c) = (a \wedge b) \wedge c$

$\wedge$ imposes a **partial order** on L, $\forall\, a, b, \& \; c \in L$:

1. $a \geq b \Leftrightarrow a \wedge b = b$
2. $a > b \Leftrightarrow a \geq b$ and $a \neq b$

a and b may not be comparable, when a $\wedge$ b is neither a nor b

A semilattice has a **bottom** element, denoted $\perp$

1. $\forall\, a \in L, \perp \wedge\, a = \perp$
2. $\forall\, a \in L$, a $\geq \perp$

$\wedge$ is the operator applied to sets when two control-flow paths converge

Data-flow Analysis

How does this relate to data-flow analysis?

- Choose a semilattice to represent the facts
- Attach a meaning to each $a \in L$

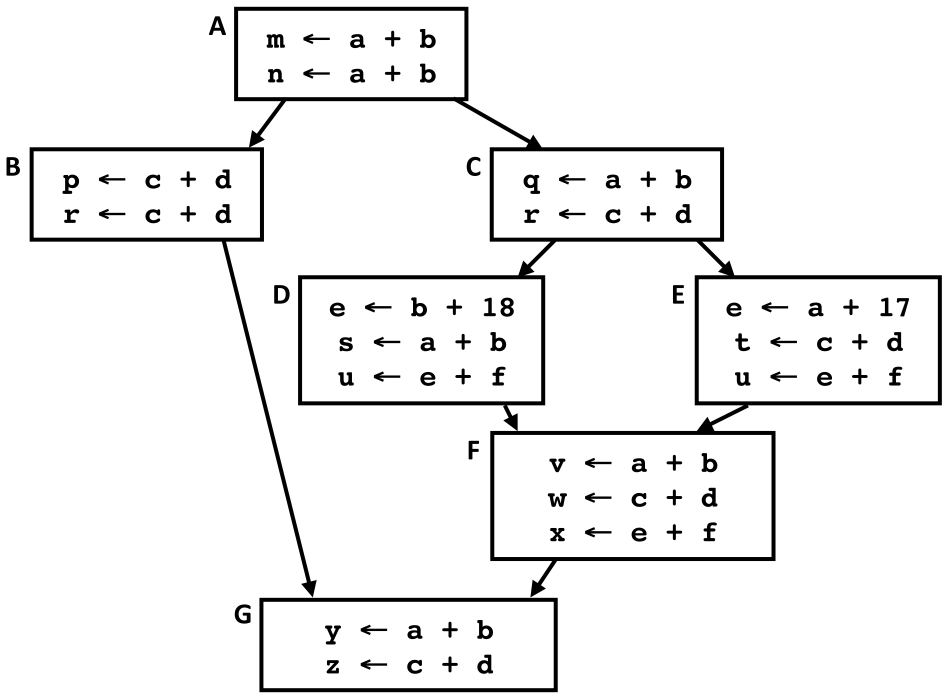
  Each $a \in L$ is a distinct set of known facts
- With each node n, associate a function $f_n : L \rightarrow L$

  f_n models behavior of code in block corresponding to n
- Let F be the set of all functions that the code might generate

Example — AVAIL

- Semilattice is $(2^E, \wedge)$, where E is the set of all expressions & $\wedge$ is $\cap$
  - ♦ Set are bigger than $|variables|$, $\perp$ is $\emptyset$
- For a node n, f_n has the form $f_n(x) = a_n \cup (x \cap b_n)$
  - ♦ Where a_n is **DEEXPR**(n) and b_n is not(**EXPRKILL**(n))

Concrete Example: Available Expressions



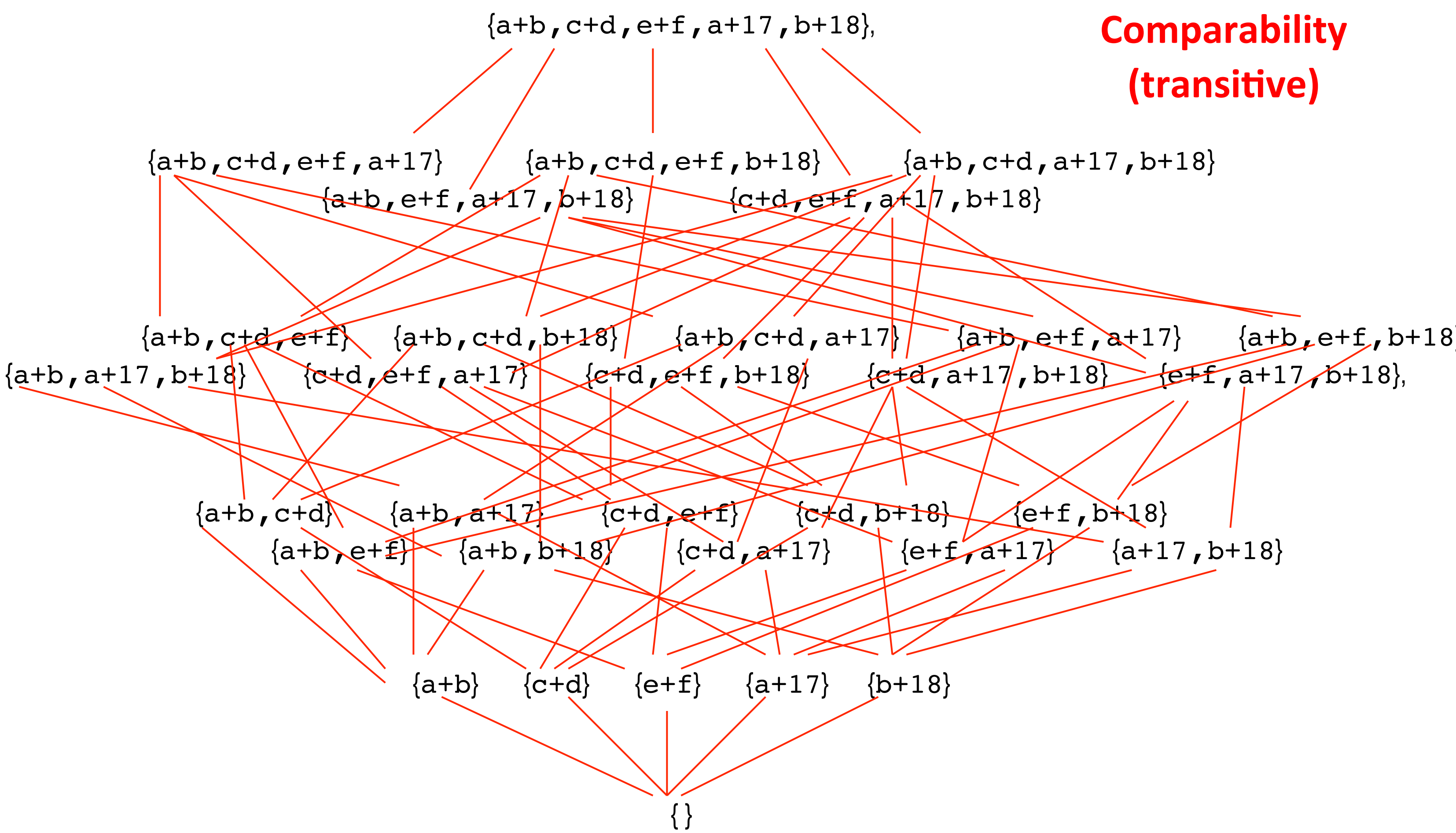
***E* = {a+b, c+d, e+f, a+17, b+18}**

2^E is the set of all subsets of *E*

2^E = [{a+b,c+d,e+f,a+17,b+18},
{a+b,c+d,e+f,a+17},
{a+b,c+d,e+f,b+18},
{a+b,c+d,a+17,b+18},
{a+b,e+f,a+17,b+18},
{c+d,e+f,a+17,b+18}, {a+b,c+d,e+f},
{a+b,c+d,b+18}, {a+b,c+d,a+17},
{a+b,e+f,a+17}, {a+b,e+f,b+18},
{a+b,a+17,b+18}, {c+d,e+f,a+17},
{c+d,e+f,b+18}, {c+d,a+17,b+18},
{e+f,a+17,b+18}, {a+b,c+d},
{a+b,e+f}, {a+b,a+17}, {a+b,b+18},
{c+d,e+f}, {c+d,a+17}, {c+d,b+18},
{e+f,a+17}, {e+f,b+18}, {a+17,b+18},{a
+b}, {c+d}, {e+f}, {a+17}, {b+18}, {}]

Concrete Example: Available Expressions

The Lattice



Concrete Example: Available Expressions

The Lattice

Effect of meet operator

{a+b,c+d,e+f,a+17,b+18},

{a+b,c+d,e+f,a+17} {a+b,c+d,e+f,b+18} {a+b,c+d,a+17,b+18}
{a+b,e+f,a+17,b+18} {c+d,e+f,a+17,b+18}

{a+b,c+d,e+f} {a+b,c+d,b+18} {a+b,c+d,a+17} {a+b,e+f,a+17} {a+b,e+f,b+18}
{a+b,a+17,b+18} {c+d,e+f,a+17} {c+d,e+f,b+18} {c+d,a+17,b+18} {e+f,a+17,b+18},

{a+b,c+d} {a+b,a+17} {c+d,e+f} {c+d,b+18} {e+f,b+18}
{a+b,e+f} {a+b,b+18} {c+d,a+17} {e+f,a+17} {a+17,b+18}

{a+b} {c+d} {e+f} {a+17} {b+18}

{ }

*

Round-robin Iterative Algorithm

$\textbf{AVAIL}(b_0) \leftarrow \emptyset$

for i $\leftarrow$ 1 to N

  $\textbf{AVAIL}(b_i) \leftarrow \{$ *all expressions* $\}$

change $\leftarrow$ true

while (change)

  change $\leftarrow$ false

  for i $\leftarrow$ 0 to N

    $\textbf{TEMP} \leftarrow \bigcap_{x \in preds(b)} (\textbf{DEEXPR}(x) \cup (\textbf{AVAIL}(x) \cap \overline{\textbf{EXPRKILL}(x)}))$

    if $\textbf{AVAIL}(b_i) \neq \textbf{TEMP}$ then

      change $\leftarrow$ true

      $\textbf{AVAIL}(b_i) \leftarrow \textbf{TEMP}$

Inner loop, or one "sweep"

Termination

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

Iterative Data-flow Analysis

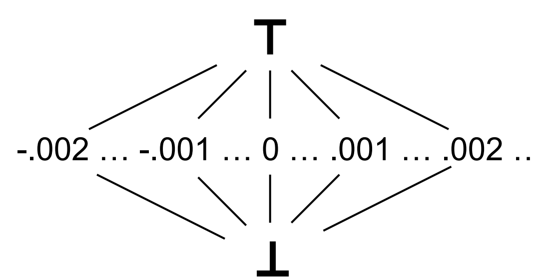
Termination

- If every $f_n \in F$ is **monotone**, *i.e.*, $x \leq y \Rightarrow f(x) \leq f(y)$, and
- If the lattice is **bounded**, *i.e.*, **every descending chain is finite**
  - Chain is sequence $x_1, x_2, \ldots, x_n$ where $x_i \in L$, $1 \leq i \leq n$
  - $x_i > x_{i+1}$, $1 \leq i < n \Rightarrow$ chain is descending

Then

- The set at each node can only change a *finite* number of times
- The iterative algorithm *must halt* on an instance of the problem ♦

- Any finite semilattice is bounded
- Some infinite semilattices are bounded



Real constants

Iterative Data-flow Analysis

Correctness (*What does it compute?*)

- If every $f_n \in F$ is **monotone**, *i.e.*, $x \leq y \Rightarrow f(x) \leq f(y)$, and
- If the semilattice is **bounded**, *i.e.*, **every descending chain is finite**
  - Chain is sequence $x_1, x_2, \ldots, x_n$ where $x_i \in L$, $1 \leq i \leq n$
  - $x_i > x_{i+1}$, $1 \leq i < n \Rightarrow$ chain is descending

Given a **bounded semilattice** S and a **monotone function space** F

- $\exists\, k$ such that $f^k(\bot) = f^j(\bot)\ \forall\, j > k$
- $f^k(\bot)$ is called the least fixed-point of f over S — pessimism
- If L has a $\top$, then $\exists\, k$ such that $f^k(\top) = f^j(\top)\ \forall\, j > k$ and

  $f^k(\top)$ is called the maximal fixed-point of f over S — optimism

Iterative Data-flow Analysis

Correctness

- If every $f_n \in F$ is **monotone**, *i.e.,* $f(x \wedge y) \leq f(x) \wedge f(y)$, and
- If the lattice is **bounded**, *i.e.,* **every descending chain is finite**
  - ♦ Chain is sequence $x_1, x_2, \ldots, x_n$ where $x_i \in L$, $1 \leq i \leq n$
  - ♦ $x_i > x_{i+1}$, $1 \leq i < n \Rightarrow$ chain is descending

Then

- The round-robin algorithm computes a least fixed-point (LFP)
- The uniqueness of the solution depends on other properties of *F*

- Unique solution ⇒ it finds the one we want
- Multiple solutions ⇒ we want to know which solution it finds
  - ♦ Specific solution may depend on order in which algorithm visits the nodes …

Iterative Data-flow Analysis

MOP ≅ meet over all paths solution
LFP ≅ least fixed-point solution
MFP ≅ maximal fixed-point solution

Correctness

- Does the iterative algorithm compute the desired answer?

Admissible Function Spaces

1. $\forall f \in F, \forall x,y \in L, f(x \wedge y) = f(x) \wedge f(y)$
2. $\exists f_i \in F$ such that $\forall x \in L, f_i(x) = x$
3. $f,g \in F \; \exists h \in F$ such that $h(x) = f(g(x))$
4. $\forall x \in L, \exists$ a finite subset $H \subseteq F$ such that $x = \wedge_{f \in H} f(\perp)$

Not distributive ⇒ fixed point solution may not be unique

If *F* meets these four conditions, then an instance of the problem will have a unique fixed point solution (*instance ⇒ graph + initial values*)

⇒ **LFP** = **MFP** = **MOP**

⇒ order of evaluation does not matter

*

Iterative Data-flow Analysis

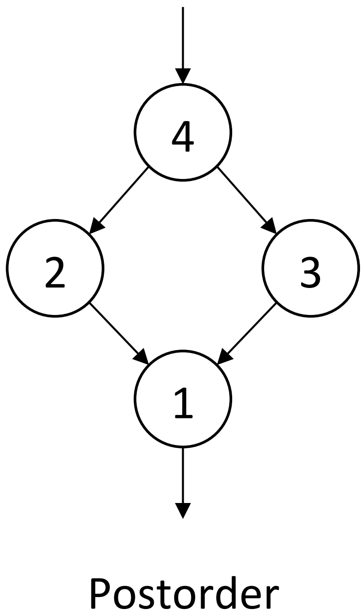
If a data-flow framework meets those admissibility conditions then it has a unique fixed-point solution

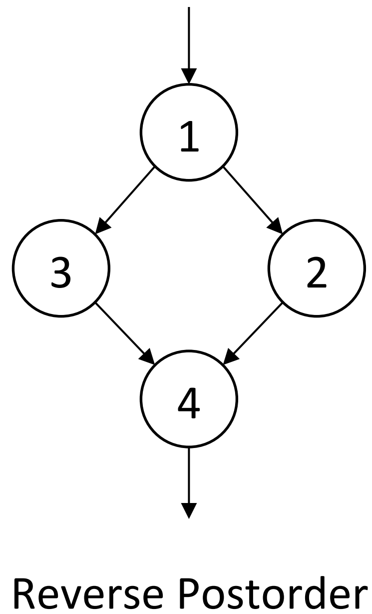
- The iterative algorithm finds the (best) answer
- The solution **does not** depend on order of computation
- Algorithm can choose an order that converges quickly

Intuition

- Choose an order so that changes propagate as far as possible on each major iteration, or "sweep" over the graph
  - ♦ Process a node's predecessors before the node
- Cycles pose problems, of course
  - ♦ Ignore back edges when computing the order?

Ordering the Nodes to Maximize Propagation





N+1 - *postorder number*

- Reverse postorder visits predecessors before visiting a node
- Use reverse preorder for backward problems
  - ♦ Reverse postorder on reverse CFG is not reverse preorder [EaC2e, exercise 9.4(b)]

Iterative Data-flow Analysis

Sets stabilize in two passes around a loop

Speed

- For a problem with an admissible function space & a bounded semilattice,
- If the functions all meet the rapid condition, *i.e.,*

$$\forall f, g \in F,\ \forall\ x \in L,\ f(g(\perp)) \geq g(\perp) \wedge f(x) \wedge x$$

then, a round-robin, reverse-postorder iterative algorithm will halt in d(G)+3 passes over a graph G

Each pass does O(E) meets & O(N) other operations

d(G) is the **loop-connectedness** of the graph with respect to a *DFST*

- Maximal number of back edges in an acyclic path
- Several studies suggest that, in practice, d(G) is small (<*3*)
- For most CFGs, d(G) is independent of the specific *DFST*

Iterative Data-flow analysis

What does all this mean?

- Reverse postorder
  - ♦ Easily computed order that increases propagation per pass
- Round-robin iterative algorithm
  - ♦ Visit all the nodes in a consistent order (RPO)
  - ♦ Do it again until the sets stop changing
- Rapid condition
  - ♦ Most classic global data-flow problems meet this condition

These conditions are easily met

- ♦ Admissible framework, rapid function space
- ♦ Round-robin, reverse-postorder, iterative algorithm

⇒ The analysis runs in (*effectively*) linear time

Iterative Data-Flow Analysis

Almost all of the classic global data-flow problems are admissible and rapid

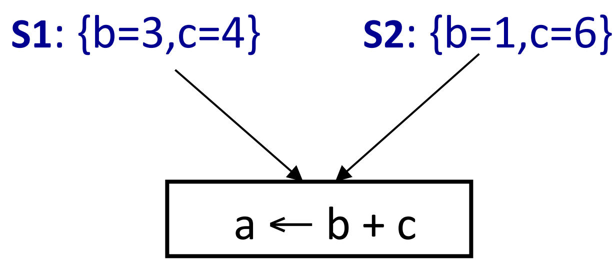
- Equations have form and properties similar to **AVAIL**
  - ♦ Live variables, reaching definitions, reachable uses
  - ♦ Some, such as dominance, have simpler equations
- Iterative algorithm will generate the correct answer quickly

The iterative algorithm is your "desert island" data-flow algorithm

- One algorithm for almost all problems
- Easy to formulate, easy to implement, easy to understand

Some problems are not admissible

Global constant propagation



- Function " **f** " models block's effects
- f(S1) = {a=7,b=3,c=4}
- f(S2) = {a=7,b=1,c=6}
- f(S1∧S2) = Ø

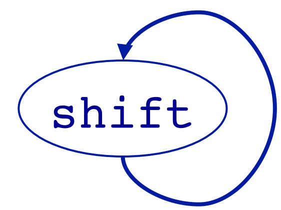
- First condition in admissibility
  $\forall f \in F, \forall x,y \in L, f(x \wedge y) = f(x) \wedge f(y)$
- Constant propagation is not admissible
  - ♦ Kam & Ullman time bound does not hold
  - ♦ There are tight time bounds, however, based on lattice height
  - ♦ Require a variable-by-variable formulation …
- Fixed point is not unique (*no guarantee that* **LFP = MFP = MOP**)

Some admissible problems are not rapid

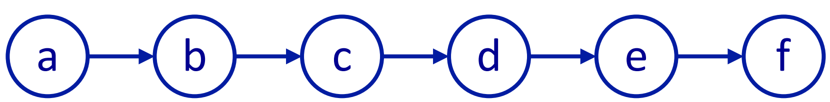
Interprocedural May Modify Sets

```
shift(a,b,c,d,e,f)
{
  local t;
  …
  call shift(t,a,b,c,d,e);
  f = 1;
  …
}
```

- Assume call-by-reference
- Compute the set of variables (in shift) that can be modified by a call to shift
- How long does it take?



- Iterations proportional to number of parameters
  - ♦ Not a function of the call graph
  - ♦ Can make example arbitrarily bad
- Proportional to length of chain of bindings…



Nothing to do with d(G)