Dataflow Analysis

- Lattice theoretic foundations
  - Partial ordering
  - Meet, Join, Lattice, Chain
- Round robin fixed point iteration
- Function properties
  - Monotonicity
  - Distributivity
- Justification for using fixed-point iteration on dataflow equations
- Meet Over all Paths solution

Lattice Theory

- Partial ordering □
  - Relation between pairs of elements
  - Reflexive x □ x
  - Anti-symmetric x □ y, y □ x □ x = y
  - Transitive x □ y, y □ z □ x □ z
- Poset (Set S, □)
- 0 Element 0 □ x, □ x □ S
- 1 Element 1 ≥ □ x □ S

A poset need not have 0 or 1 element.

Lattice Theory

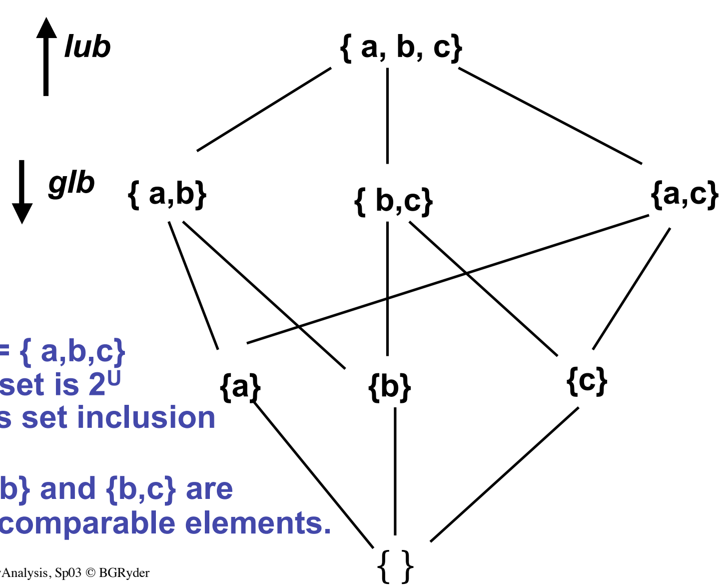
- Greatest lower bound (glb)
  l1, l2 ∈ poset S, a ∈ S is glb(l1, l2)
  if a ≤ l1 and a ≤ l2 then
  for any b ∈ S, b ≤ l1, b ≤ l2 ⇒ b ≤ a

if glb is unique it is called the meet (⊓) of l1 and l2.

- Least upper bound (lub)
  l1, l2 ∈ poset S, c ∈ S is lub(l1,l2)
  if c ≥ l1 and c ≥ l2 then
  for any d ∈ S, d ≥ l1, d ≥ l2 ⇒ c ≤ d.

if lub is unique is called the join (⊔) of l1 and l2.

Poset Example



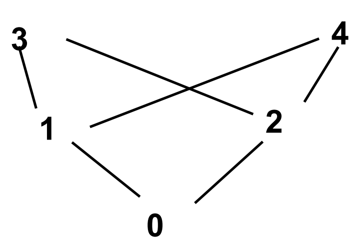
U = { a,b,c}
poset is 2^U
⊆ is set inclusion

{a,b} and {b,c} are incomparable elements.

Definition of a Lattice (L, $\sqcap$, $\sqcup$)

- L, a poset under $\sqsubseteq$ such that every pair of elements has a unique glb (meet) and lub (join).
- A lattice need not contain an 0 or 1 element.
- A finite lattice must contain an 0 and 1 element.
- Not every poset is a lattice.
- If a $\sqsubseteq$ x $\forall$ x $\in$ L, then a is 0 element of L
- If x $\sqsubseteq$ a $\forall$ x $\in$ L, then a is 1 element of L

a poset, but not a lattice



There is no lub(3,4) in this poset so it is not a lattice.

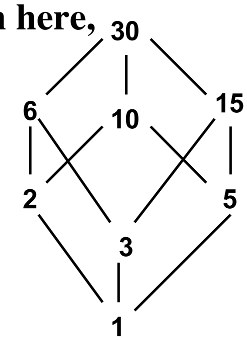
Examples of Lattices

- **H = (2^U , □, □) where U is a finite set**
  - **glb (s1, s2) is (s1 □ s2) which is s1 □ s2**
  - **lub (s1, s2) is (s1 □ s2) which is s1 □ s2**

- **J = (N_1 , gcd, lowest common multiple)**
  - **partial order relation is integer divide on N_1**
    n1 | n2 if division is even
  - **lub (n1, n2) is n1 □ n2 = lowest common multiple(n1,n2)**
  - **glb (n1,n2) is n1 □ n2 = greatest common divisor (n1,n2)**

Chain

- **A poset C where, for every pair of elements**
c1, c2 □ C, either c1 □ c2 or c2 □ c1.
  e.g., { } □ {a} □ {a,b} □ {a,b,c}
  and from the lattice as shown here,
    1 □ 2 □ 6 □ 30
    1 □ 3 □ 15 □ 30



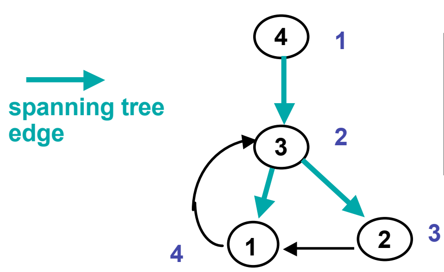
Lattices are used in dataflow analysis to argue the existence of a solution obtainable through fixed-point iteration.

Round-robin Fixed-point Iteration

- Iterates through nodes of cfg in specific node order, which preserves the partial order of the graph.
- Recall worklist algorithm(per bit) has O(me) work where |E|=e and m is number of dataflow facts in the set.
- Postorder on depth-first spanning tree defines rPostorder which is *reverse postorder*
  - Ancestor-first node ordering, good for forward dataflow problems
  - rPostorder_number =(| N |+1) - postorder_number
  - Postorder is good node ordering for backward dataflow problems

rPostorder Example

control flowgraph



Postorder numbering of depth-first leftmost traversal of cfg rPostorder# is defined as (n+1)-Postorder#

Round-robin Iteration

Form depth-first spanning tree of control flowgraph and derive rPostorder numbering for nodes

```
change = true
Initialize REACH(m) = ∅, □ m

while (change) do
{   change = false
    for (m = 2; m <= n; m++)
    { new = □ ( Reach(j) □ pres(j) □ dgen(j) )
          j = pred(m)

      if (new != REACH(m))
              then {REACH(m) = new;  change = true}
    }
}
```

Round-robin tends to push dataflow facts as deep as possible on acyclic paths in the flowgraph on each iteration

Round-robin Iteration

Form depth-first spanning tree of control flowgraph and derive rPostorder numbering for nodes

```
change = true
Initialize REACH(m) = ∅, □ m)

while (change) do
{   change = false
    for (m = 2; m <= n; m++)
    { new = □ ( Reach(j) □ pres(j) □ dgen(j) )
          j = pred(m)

      if (new != REACH(m))
              then {REACH(m) = new;  change = true}
    }
}
```

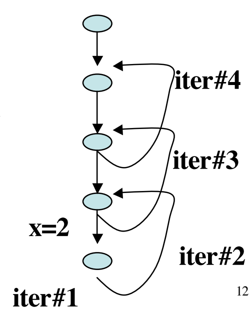
O(n)

#iters of while * cost each iteration

cost *for* statement ~ cost of all ops during lifetime of *for*

$\boxed{d = 3; d+2 = 5}$

iter#5, no change



Cost of Loop Operations

Calculate *for* statement cost over all its iterations:

1. 2(n-1) bit vector operations to calculate each node's factor new = □ (Reach(j) □ pres(j) □ dgen(j) and memoize it;
2. unions: (#preds(m)-1) for each node m means we do (e - n) unions per *for* statement for all nodes;
3. rest of *for* is constant number of operations

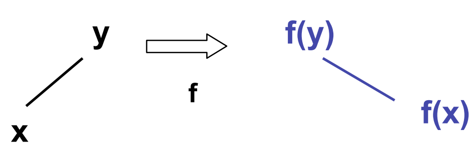
Each while loop iteration costs (e - n) + (2n-2) □ e + n;

Therefore, entire algorithm costs O(de), (□ O(dn) by assumption for flowgraphs), bit vector operations where d+2 for classical bitvector problems.

Note: this dominates O(e) of calculation of rPostorder numbers and the O(n) of initialization loop.

More Lattice Properties

- *Finite length*: if every chain in lattice is finite
- *Bounded lattice*: if lattice contains both 0 and 1 elements
- *Distributive lattice*: if □ x,y,z □ S
  x □ (y □ z) = (x □ y) □ (x □ z)
  x □ (y □ z) = (x □ y) □ (x □ z)
- (S,□) poset, f: S --> S is monotonic iff
  □ x, y □ S, x □ y □ f(x) □ f(y)
- Monotonic functions preserve domain ordering in their range values



Fixed point theorem - Why it works?

Intutition--

Given a 0 in lattice and monotonic function f, 0 □ f(0).

Apply f again and obtain

0 □ f(0) □ f(f(0)) = f^2 (0)

Continuing,

0 □ f(0) □ f^2 (0) □ f^3(0) □ ... □ f^k(0)= f^{k+1}(0) for a finite chain lattice.

This is tantamount to saying $\lim_{k □} f^k(0)$ exists and is called the *least fixed point* of f,

since f(f^k(0)) = f^k(0)

Fixed Point Theorem

Thm: f: S --> S monotonic function on poset (S, □) with a 0 element and finite length. The *least fixed point* of f is f^k (0) where

i. f^0 (x) = x,

ii. f^{i+1} (x) = f(f^i(x)), $i \geq 0$,

iii. f^k(0) = f(f^k(0)) and this is the smallest k for which this is true.

- **For any p such that f(p)=p, f^k(0) □ p.**
- **Theorem justifies the iterative algorithm for global data flow analysis for lattices & functions with right properties**
- **Dual theorem exists for 1 element and *maximal fixed point* for k such that f^k(1) = f^{k+1}(1).**

Application to DFA on CFG

- **Cartesian cross product of posets is a poset**
  - **if (S, □) is a poset, then (S x S x ... x S, □') is a poset whose partial ordering is component-wise □**

  W=(W1,W2,…,Wn) □'Y=(Y1,Y2,…,Yn) iff

  W1 □ Y1 and W2 □ Y2, …, and Wn □ Yn
- **Cartesian cross product of lattices is a lattice (with an induced partial ordering)**
- **Monotone function on cross product lattice can be built from monotone functions on each component lattice** **F(Y1, Y2,..., Yn) = (g_1 (Y1, Y2,...,Yn),..., g_n(Y1, Y2,...,Yn))**

where g_I : (L,L, ..., L) --> L and □' is component-wise □ from L

Example - Available Exprs

- **lattice is 2^{Exprs} where Exprs is set of all binary expressions in program**
- **Partial order is □ (subset inclusion) so meet is □**
- **< Exprs,Exprs,...,Exprs> is 1 element**
- **< ∅, ∅,..., ∅) is 0 element**
- **From the data flow equations for AVAIL, we know that if a set of dataflow facts X is true on entry to a flowgraph node n, then f(X) is true on each exit edge of n where**

  f(X) = epres(n) □ X □ egen(n)

f is called the *transfer function* for AVAIL

Example, Available Exprs

- Cross product lattice is
  - (2^{Exprs} , 2^{Exprs} ,..., 2^{Exprs}) with n components where n is number of nodes in the cfg and □' is □ component-wise
- Avail equation at a node can be expressed thusly,
  Avail(j) = □ { Avail(m) □ epres(m) □ egen(m) }
  m □ Pred(j)
  - AVAIL (j) is the solution at entry of node j and f(AVAIL(j)) is solution at exit of node j,

  g_j = □ f(g_m), m □ Pred(j)
  - One step of the worklist algorithm maps a potential solution for AVAIL at a node of the cfg to another potential solution for that node

Example, Available Exprs

- Can you show g_i monotone?

  g_i : (2^{Exprs} , 2^{Exprs} ,..., 2^{Exprs}) --> 2^{Exprs}
- Then this induces the monotonicity of F,

  F = (g_1,..., g_n)
- Application of dual of fixed point theorem here to find the maximal fixed point. Iterate down from the 1 element.
  - Initialize □ to ∅, all other cfg nodes to Exprs.

Sketch of Validity Proof of Iterative Algorithm for DFA

- **Let Redef ($Y_1^i \ldots Y_n^i$) be result after i steps of the worklist algorithm (per node) for DFA.**

  Redef($Y_1^i \ldots Y_n^i$) = ($Y_1^i \ldots Y_{k-1}^i$, g_k ($Y_1^i \ldots Y_n^i$),$Y_{k+1}, \ldots Y_n$)
- **In the next iteration (i+1st),only 1 component of Y= ($Y_1^i \ldots Y_n^i$) changes and this component is chosen non-deterministically.**
- **Recall we have a function defined on the flowgraph: $F(Y_1,\ldots,Y_n) = (g_1(Y_1,\ldots,Y_n),\ldots, g_n(Y_1,\ldots,Y_n))$ where g_j : (L,L, ..., L) --> L is defined by the dataflow problem**

Sketch of Proof, cont.

- **Assume minimal fixed point of F is $F^m(0)$.**
- **Must show iterative algorithm halts at a fixed point of Redef function, Redefmin, and that this fixed point is meaningful to the data flow problem under solution, namely Redefmin = $F^m(0)$.**
- **First show iterates of Redef form a chain.**

  0 □’ Y^1 □’ Y^2 □’ Y^3 □’ ... □’ Y^k

Therefore, Redefmin = Redef^k(0) for some smallest k, by finiteness of chains on the lattice.

Sketch of Proof, cont.

Second, let $F^m(0)$ be the minimum fixed point of the data flow function F on the cross product lattice. Show $F^m(0)$ □’ Redefmin.

$F^m(0)$ □’ Redefmin because Redefmin is also a fixed point of F, but not necessarily its minimal fixed point.

Third, show Redefmin □’ $F^m(0)$ in two steps

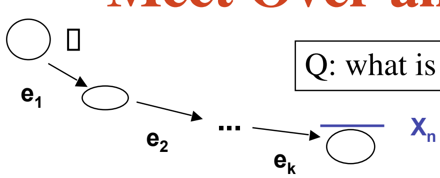
(1) $Redef^k(0)$ □’ $F^m(0)$ for all k and

(2) Redefmin = $Redef^k(0)$ for large enough k,

so for k large enough, Redefmin □’ $F^m(0)$

Therefore, Redefmin= $F^m(0)$ and the fixed point iterative procedure does find the minimal fixed point on the lattice

Meet Over all Paths solution



Q: what is relation between MFP and MOP?

- Maximal data flow information desired is obtained by traversing ALL PATHS from □ to n.
- Consider X_n = □ $f_Q(0)$, for Q a path in the flowgraph, $Q = e_1, e_2, \ldots e_k$ so that

$$X_n = \underset{Q = e_k \ldots e_1}{□} f_{e_k} \circ f_{e_{k-1}} \circ \ldots \circ f_{e_2} \circ f_{e_1}(0)$$

MOP is best summary of data flow facts possible to compute at compile-time.