Dataflow Frameworks, cont.

Announcements

- Quiz 1 today
  - On four classical dataflow problems
- Homework due Thursday
  - Questions?

Outline of Today's Class

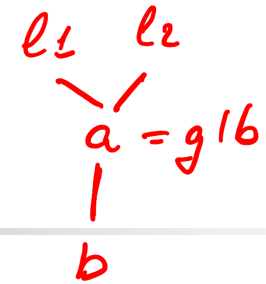
- Dataflow framework
  - Lattices
  - Transfer functions
  - Worklist algorithm
- MOP solution vs. MFP solution
- Reading:
  - Dragon Book, Chapter 9.2 and 9.3

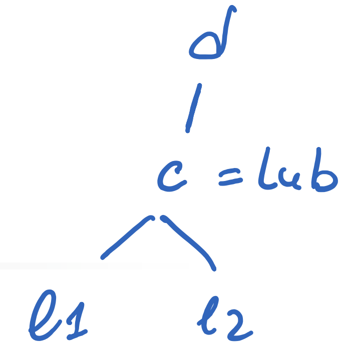
Lattice Theory

- Partial ordering (denoted by **≤** or ⊑)
  - Relation between pairs of elements
  - Reflexive **a ≤ a**
  - Anti-symmetric **a ≤ b** and **b ≤ a** ==> **a = b**
  - Transitive **a ≤ b** and **b ≤ c** ==> **a ≤ c**
- Partially ordered set (poset) (set S, ≤)
  - **0** element **0 ≤ a**, for every **a** in S
  - **1** element **a ≤ 1**, for every **a** in S

We don't necessarily need 0 or 1 element

Lattice Theory





- Greatest lower bound (glb)

  l1, l2 in poset S, **a** in poset S is the **glb(l1,l2)** iff

  1) **a ≤ l1** and **a ≤ l2**

  2) for any **b** in S, **b ≤ l1, b ≤ l2** implies **b ≤ a**

If glb exists, it is unique. Why? Called ***meet*** (denoted by Λ or ⊓) of l1 and l2.

- Least upper bound (lub)

  l1, l2 in poset S, **c** in poset S is the **lub(l1,l2)** iff

  1) **c ≥ l1** and **c ≥ l2**

  2) for any **d** in S, **d ≥ l1, d ≥ l2** implies **d ≥ c**

If lub exists, it is unique. Called ***join*** (denoted by V or ⊔) of l1 and l2.

Definition of a Lattice **(L, ∧, ∨)**

- A lattice **L** is a poset under ≤, such that every pair of elements has a **glb (meet)** and **lub (join)**

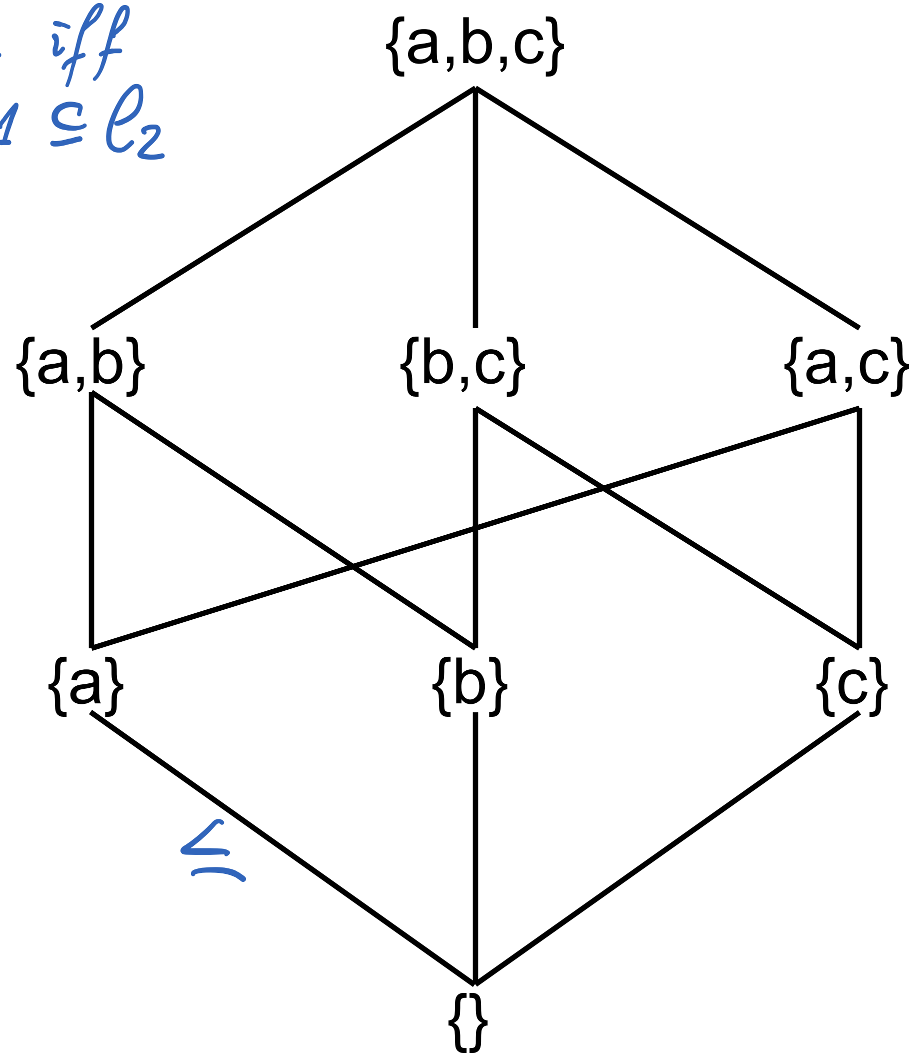
- A lattice need not contain a 0 or 1 element
- A finite lattice must contain 0 and 1 elements
- Not every poset is a lattice
- If there is element **a** such that **a ≤ x** for every **x** in **L**, then **a** is the 0 element of **L**
- If there is **a** such that **x ≤ a** for every **x** in **L**, then **a** is the 1 element of **L**

Is This Poset a Lattice

$\ell_1 \leq \ell_2$ iff $\ell_1 \subseteq \ell_2$

D = {a,b,c}
The poset is 2^D, ≤ is set inclusion

$glb(\ell_1, \ell_2) = \ell_1 \cap \ell_2$
$lub(\ell_1, \ell_2) = \ell_1 \cup \ell_2$



Thm: If $glb(l_1, l_2)$ exists then it is unique.

Suppose $\exists\ a = glb(l_1, l_2)$ and $b = glb(l_1, l_2)$ and $\boxed{a \neq b}$

By def of glb: $a \leq l_1$ $\quad b \leq l_1$

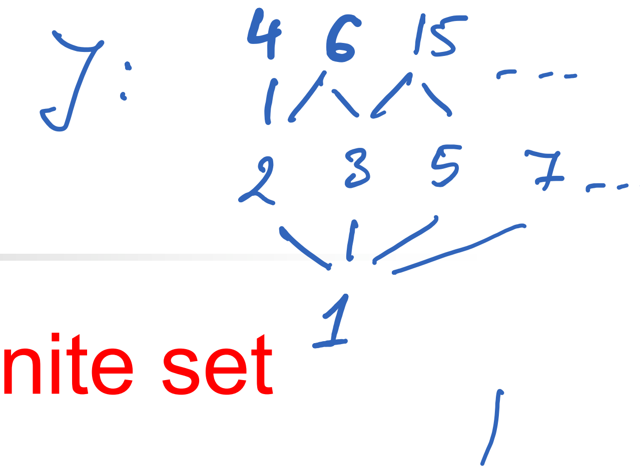
$a \leq l_2$ $\quad b \leq l_2$

Since $\boxed{a}$ is a glb of l_1 and l_2, any other lower bound $\boxed{b}$ is $b \leq a$.

Analogously, $a \leq b$.

By anti-symmetry $\boxed{a = b}$

Examples of Lattices



- H = (2^D, ∩, U) where D is a finite set
  - Partial order is subset
  - **glb(s1,s2)** denoted **s1Λs2**, is set intersection **s1∩s2**
  - **lub(s1,s2)** denoted **s1Vs2**, is set union **s1Us2**
- J = (N_1, gcd, lcm)
  - Partial order is integer division
  - **lub(n1,n2)** denoted **n1Vn2** is **lcm(n1,n2)**
  - **glb(n1,n2)** denoted **n1Λn2** is **gcd(n1,n2)**

(N_1 denotes natural numbers starting at 1)

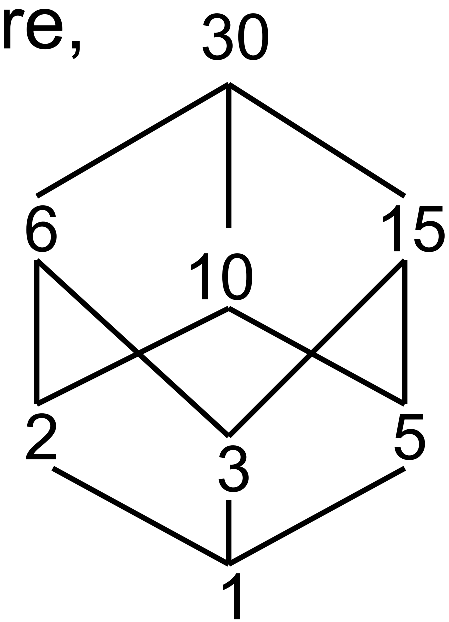
Chain

- A poset C where for every pair of elements **c1**, **c2** in C, either **c1 ≤ c2** or **c2 ≤ c1**.
  - E.g., {} ≤ {a} ≤ {a,b} ≤ {a,b,c}
  - E.g., from the lattice J as shown here,

    1 ≤ 2 ≤ 6 ≤ 30

    1 ≤ 3 ≤ 15 ≤ 30
- A lattice s.t. every ascending chain is finite, is said to satisfy the *Ascending Chain Condition*

J:



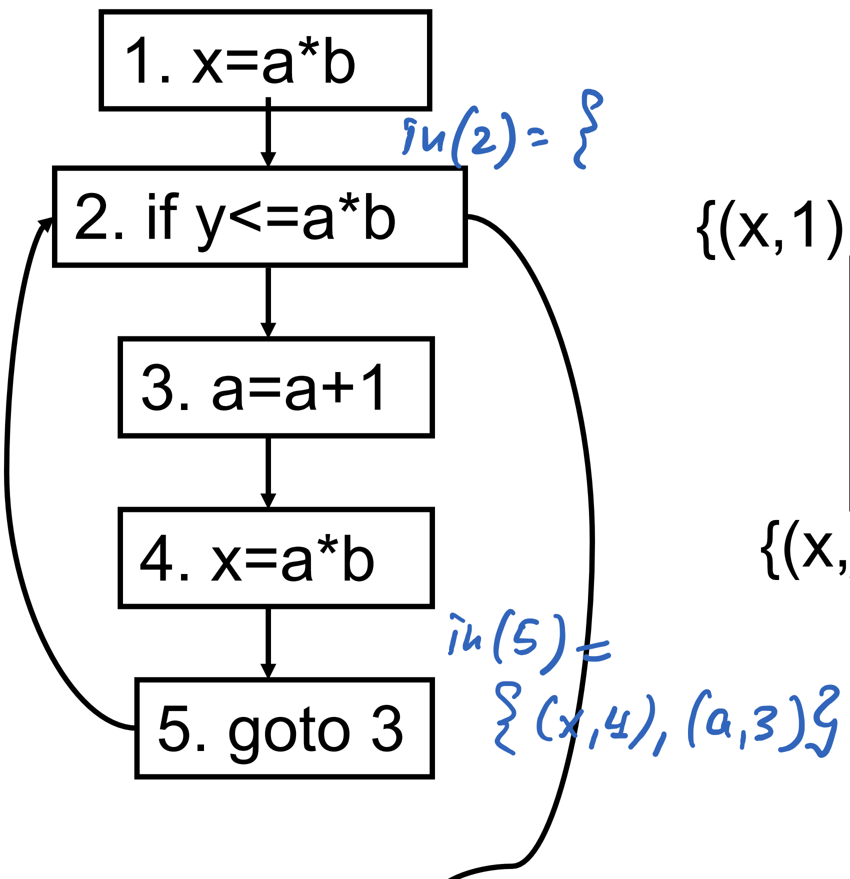
Lattices in Dataflow Analysis

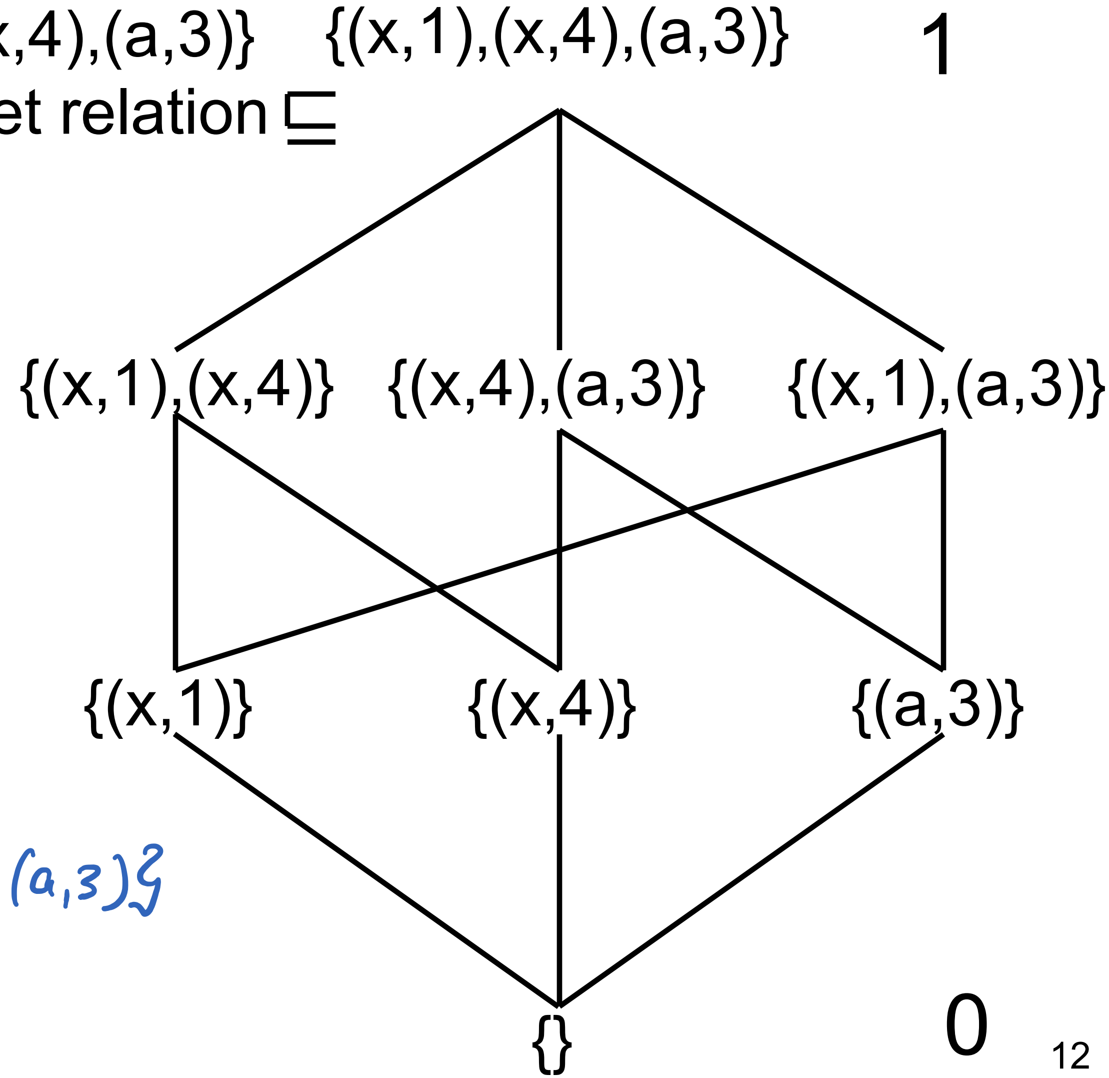
- Lattices define property space
- Lattice properties give rise to certain properties of the worklist algorithm (standard way of solving dataflow problems)

Dataflow Lattices: *Reach*

D = all definitions:{(x,1),(x,4),(a,3)}

Poset is 2^D, ≤ is the subset relation ⊑

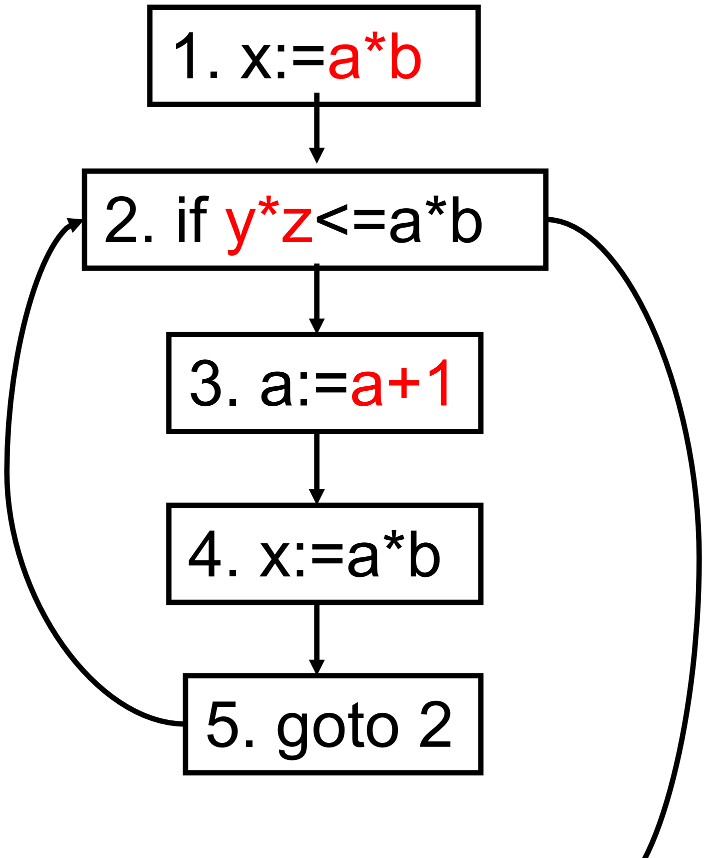


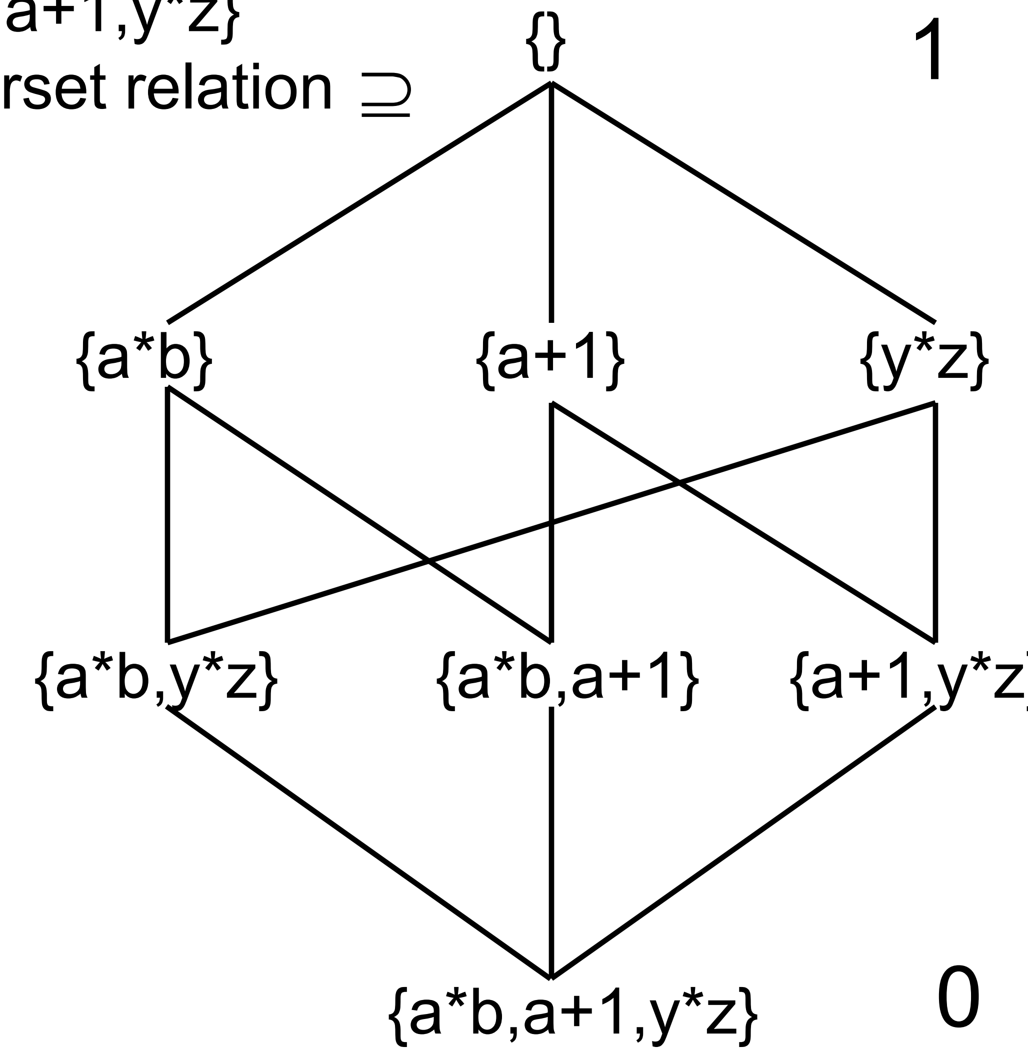


Dataflow Lattices: *Avail*

D = all expressions: {a*b,a+1,y*z}

Poset is 2^D, ≤ is the superset relation ⊇





Property Space

- **Property space must be:**
  1. A lattice **L, ≤**
  2. **L** satisfies the *Ascending Chain Condition*

     Requires that all ascending chains are finite

Property Space

- **Merge operator V** must be the join of **L**
- In dataflow, **L** is often the lattice of the subsets over a finite set of dataflow facts **D**
  - Choose universal set **D** (e.g., all definitions)
  - Choose ordering operation **≤**. Since merge operator must be the join of **L**, a *may* problem sets **≤** to **subset** and a *must* problem sets **≤** to **superset**

Example: *Reach* Lattice

- Property space is the lattice of the subsets
  - **D** is the set of all definitions in program
  - **≤** is the **subset** operation
    - Thus, join is set union, as needed for *Reach*, which is a *may* problem
  - Lattice has a **0** being **{}**, and a **1** being **D**
  - Lattice satisfies the *Ascending Chain Condition*

Example: *Avail* Lattice

- Property space is the lattice of the subsets
  - **D** is the set of all expressions in the program
  - **≤** is **superset**
    - Thus, join is set intersection, as needed for *Avail*, which is a *must* problem
  - Lattice has a **0** being **D,** and a **1** being **{}**
  - Lattice satisfies *Ascending Chain Condition*

(Monotone) Dataflow Framework

- A problem fits into the dataflow framework if
  - problem's <u>property space</u> is a lattice **L, ≤** that satisfies the *Ascending Chain Condition*
  - problem's merge operator is the join of **L**

  and
  - its <u>transfer function space</u> **F: L→ L** is monotone
- Thus, we can make use of a generic solution procedure, known as the worklist algorithm (also the maximal fixpoint algorithm or the fixpoint iteration algorithm)

Outline of Today's Class

- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm
- MOP solution vs. MFP solution
- Reading:
  - Dragon Book, Chapter 9.2 and 9.3

Transfer Functions

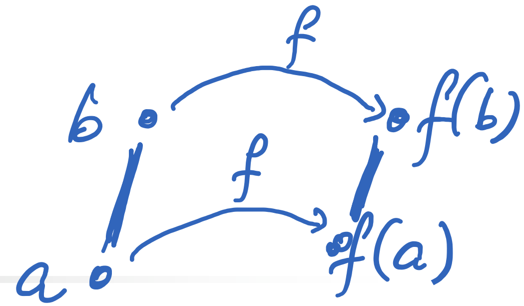
- **The transfer functions: f: L→ L**. Formally, function space **F** is such that
  1. **F** contains all $\mathbf{f_j}$
  2. **F** contains the identity function **id(x) = x**
  3. **F** is closed under composition
  4. Each **f** must be **monotone**

$$out(j) = (in(j) - kill(j)) \cup gen(j)$$

$$out(j) = f_j(in(j))$$

$$y = f_j(x)$$

Monotonicity Property



- **F: L→ L** is monotone if and only if:

  (1) **a,b** in **L**, **f** in **F** then **a ≤ b** $\Longrightarrow$ **f(a) ≤ f(b)**

  or (equivalently):

  (2) **x,y** in **L**, **f** in **F** then **f(x) V f(y) ≤ f(x V y)**

- Theorem: Definitions (1) and (2) are equivalent.
  - Show that (1) implies (2)
  - Show that (2) implies (1)

Monotonicity Property

- Show that (1) implies (2)

$a \le b \Rightarrow f(a) \le f(b)$

We need to show x, y $f(x) \vee f(y) \le f(x \vee y)$

$x \le x \vee y \Rightarrow f(x) \le f(x \vee y)$ (by (1))

$y \le x \vee y \Rightarrow f(y) \le f(x \vee y)$ (by (1))

$f(x \vee y)$ is an upper bound of $f(x)$ and $f(y)$

Therefore $f(x \vee y) \ge f(x) \vee f(y)$

the least upper bound

Distributivity Property

- **F: L → L** is distributive if and only if **x,y** in **L**, **f** in **F** then **f(x) V f(y) = f(x V y)**

- A distributive function is also monotone but not the other way around
  - Distributivity is a very nice property!

i1 i2

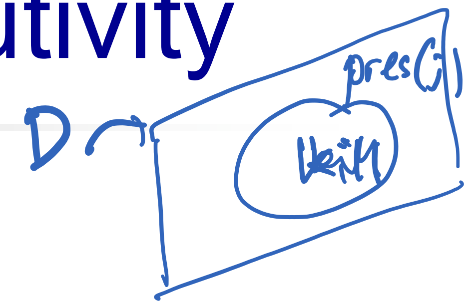
out(i1) out(i2)

j

$in(j) = out(i1) \cup out(i2)$

$f_j(in(j)) = f_j(out(i1) \cup out(i2)) = f_j(out(i1)) \cup f_j(out(i2))$

Monotonicity and Distributivity



- Is classical *Reach* distributive?
  - Yes

$out(j) = (in(j) - kill(j)) \cup gen(j)$

$out(j) = (in(j) \cap pres(j)) \cup gen(j)$

  - To show distributivity:

For each **j:** $\overbrace{((X_1 \cup X_2) \cap pres(j)) \cup gen(j)}^{f_j(X_1 \cup X_2)} =$

$\underbrace{((X_1 \cap pres(j)) \cup gen(j))}_{f_j(X_1)} \cup \underbrace{((X_2 \cap pres(j)) \cup gen(j))}_{f_j(X_2)}$

$((X_1 \cup X_2) \cap pres(j)) \cup gen(j) =$

$((X_1 \cap pres(j)) \cup (X_2 \cap pres(j))) \cup gen(j) =$

$((X_1 \cap pres(j)) \cup gen(j)) \cup ((X_2 \cap pres(j)) \cup gen(j))$

Monotone Dataflow Framework

- A problem fits into the dataflow framework if
  - problem's <u>property space</u> is a lattice **L, ≤** that satisfies the *Ascending Chain Condition*
  - problem's merge operator is the join of **L**

  and
  - its <u>transfer function</u> space **F: L→ L** is monotone
- Thus, we can make use of a generic solution procedure, known as the worklist algorithm.

Worklist Algorithm for Forward Dataflow Problems

/* Initialize to initial values; 1 is entry node of CFG */

in(1) = InitialValue; out(1) = f_1(in(1))

for m = 2 to n do in(m) = **0**; out(m) = f_m(**0**)

W = {2,…,n} /* put every node but 1 on the worklist */

while W ≠ Ø do {

    remove j from W

    in(j) = **V** { out(i) | i is predecessor of j }

    out(j) = f_j(in(j))

    if out(j) changed then

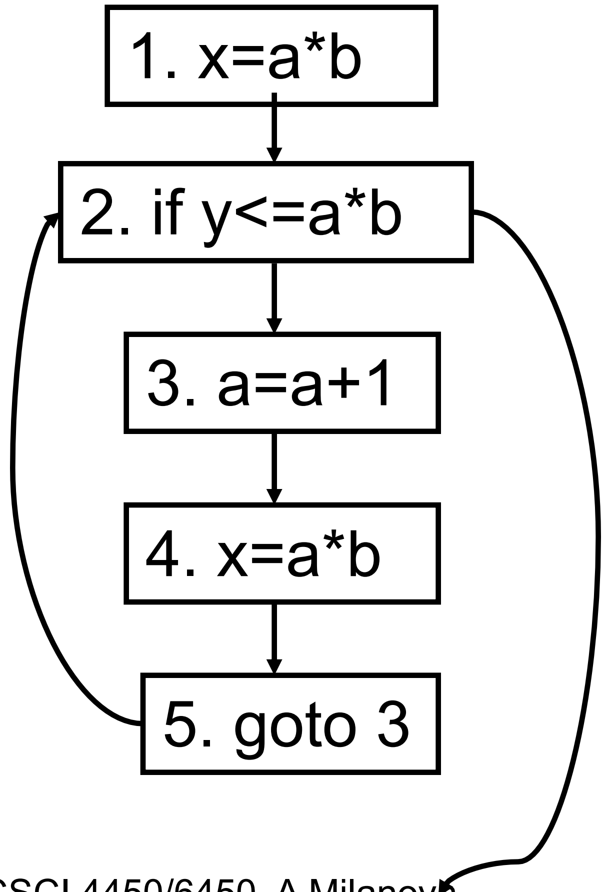
        W = W U { k | k is successor of j }

}

Worklist Algorithm on *Reach*

D = all definitions:{(x,1),(x,4),(a,3)}

Poset is 2^D, ≤ is the subset relation $\sqsubseteq$



Termination Argument

- Theorem: the algorithm terminates. Why?

- Sketch of argument:

  A node k is placed on worklist only if the out(j) of a predecessor j changes. Monotonicity of **f** guarantees $in^n(j) \leq in^{n+1}(j)$ and $out^n(j) \leq out^{n+1}(j)$. (Here $in^n(j)$, $out^n(j)$ are the sets at iteration n.)

  in and out sets are elements of **L** and **L** satisfies the *Ascending Chain Condition*; thus, there is only a finite number of times each out(j) changes.

Correctness Argument

- Theorem: Worklist algorithm computes a solution that satisfies the dataflow equations. Why?

- Sketch of argument:

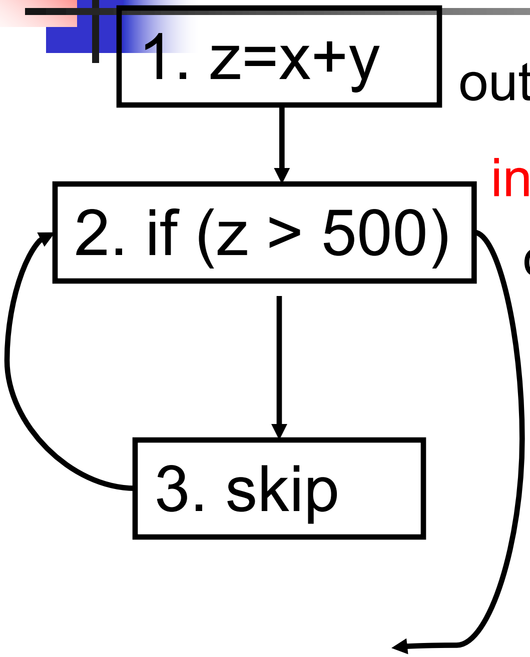
  Suppose either (1) $Vout(i) \neq in(j)$ or (2) $out(j) \neq f_j(in(j))$

  For (1) to hold we must have "grown" out(i) in some iteration and not added successor j to worklist; this is impossible.

Precision Argument

- Theorem: Worklist algorithm computes the least solution of the dataflow equations.
  - Historically, solution computed by worklist algorithm is called the **maximal fixpoint solution (MFP solution)**
  - For every node j, worklist algorithm computes a solution MFP(j) = (in(j),out(j)), such that for every solution (in'(j),out'(j)) of the dataflow equations we have in(j) ≤ in'(j) and out(j) ≤ out'(j)

Example



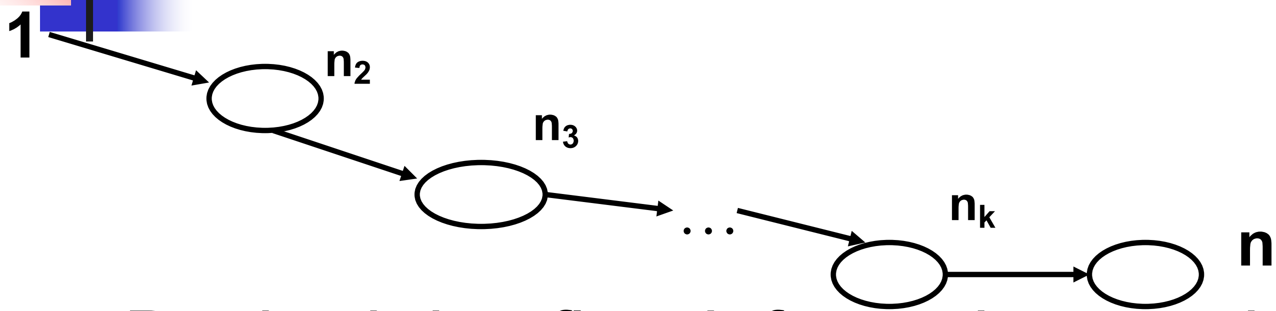
	Solution1	Solution2
$in_{Avail}(1) = \varnothing$	$\varnothing$	$\varnothing$
$out_{Avail}(1) = (in_{Avail}(1) - E_z) \cup \{x+y\}$	{x+y}	{x+y}
$in_{Avail}(2) = out_{Avail}(1) \lor out_{Avail}(3)$	{x+y}	$\varnothing$
$out_{Avail}(2) = in_{Avail}(2)$		
$in_{Avail}(3) = out_{Avail}(2)$	{x+y}	$\varnothing$
$out_{Avail}(3) = in_{Avail}(3)$		

Equivalent to: $in_{Avail}(2) = \{x+y\}$ **V** $in_{Avail}(2)$ and recall that **V** is ∩ (i.e., set intersection).

Outline of Today's Class

- Dataflow frameworks
  - Lattices
  - Transfer functions
  - Worklist algorithm
- MOP solution vs. MFP solution
- Reading:
  - Dragon Book, Chapter 9.2 and 9.3

Meet Over All Paths (**MOP**)



- Desired dataflow information at **n** is obtained by traversing ALL PATHS from 1 (entry node) to **n**.

- For every path **p=(**1, $\mathbf{n_2}$, $\mathbf{n_3}$ **..., $\mathbf{n_k}$)** we compute $\mathbf{f_{n_k}(\ldots f_{n_2}(f_1(}\text{InitialValue}\mathbf{)))}$
- The MOP at *entry* of **n** is **V** $\mathbf{f_{n_k}(\ldots f_{n_2}(f_1(}\text{InitialValue}\mathbf{)))}$

  over all paths **p** from **1** to **n**

MOP vs. MFP

- MOP is an abstraction of the best solution computable with dataflow analysis
  - It is a common assumption in dataflow analysis that *all program paths are executable*
- MFP is the solution computed by the worklist algorithm

MOP vs. MFP

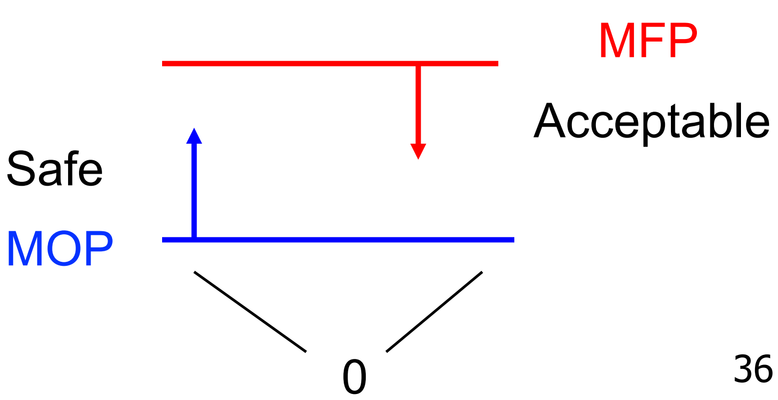
- For *distributive* problems MFP = MOP!

- Unfortunately, for *monotone* problems this is not true. But we still have a safe solution: it is a theorem that for monotone problems, MFP ≥ MOP

Safety of a Dataflow Solution

- A safe (also, correct or sound) solution X overestimates the "best" possible dataflow solution, i.e., $X \geq MOP$

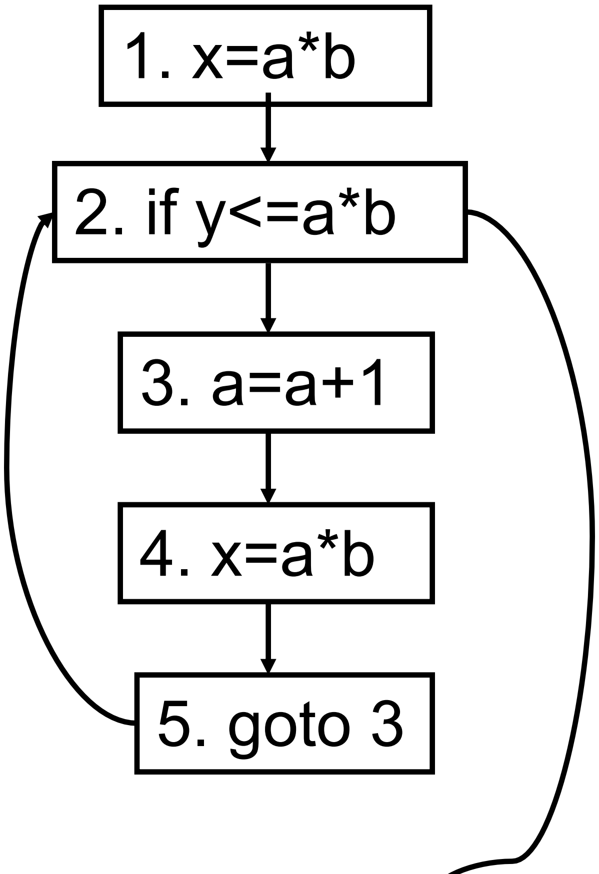
- Historically, an acceptable solution X is one that is better than what we can do with the MFP, i.e., $X \leq MFP$

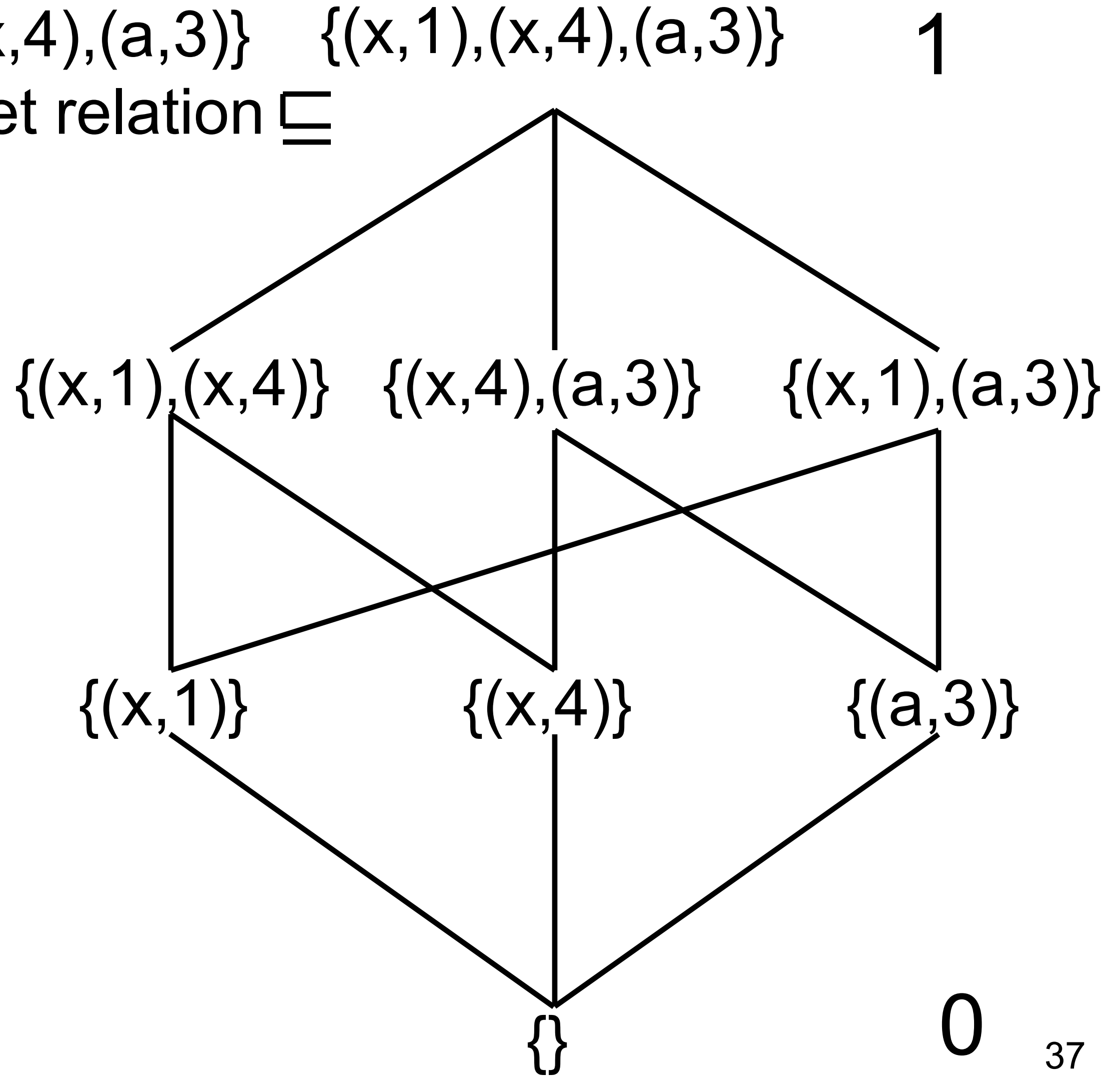


Safe Solutions: *Reach*

U = all definitions:{(x,1),(x,4),(a,3)}

Poset is 2^U, ≤ is the subset relation $\sqsubseteq$

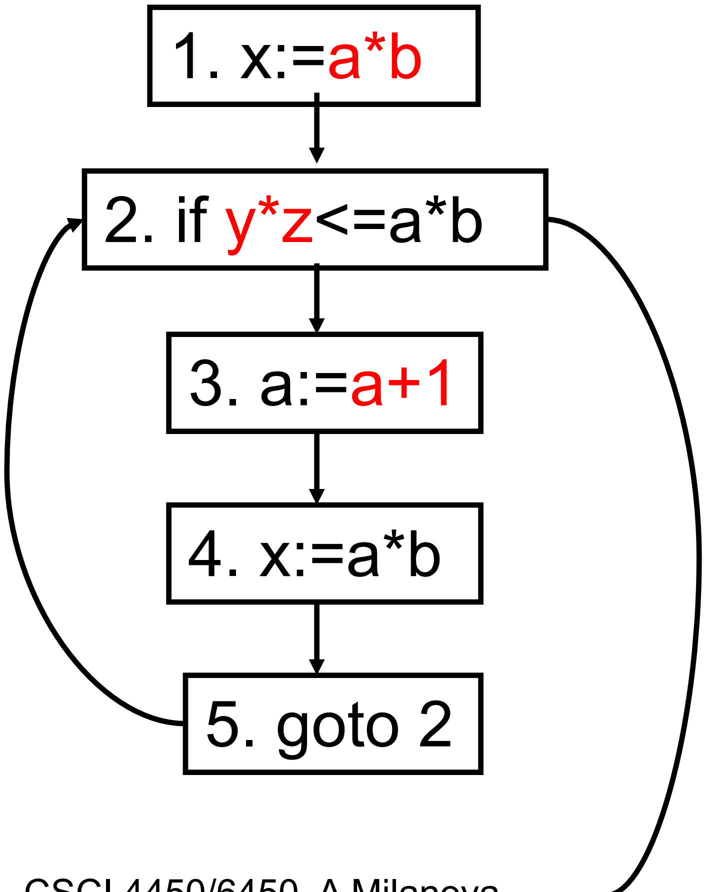


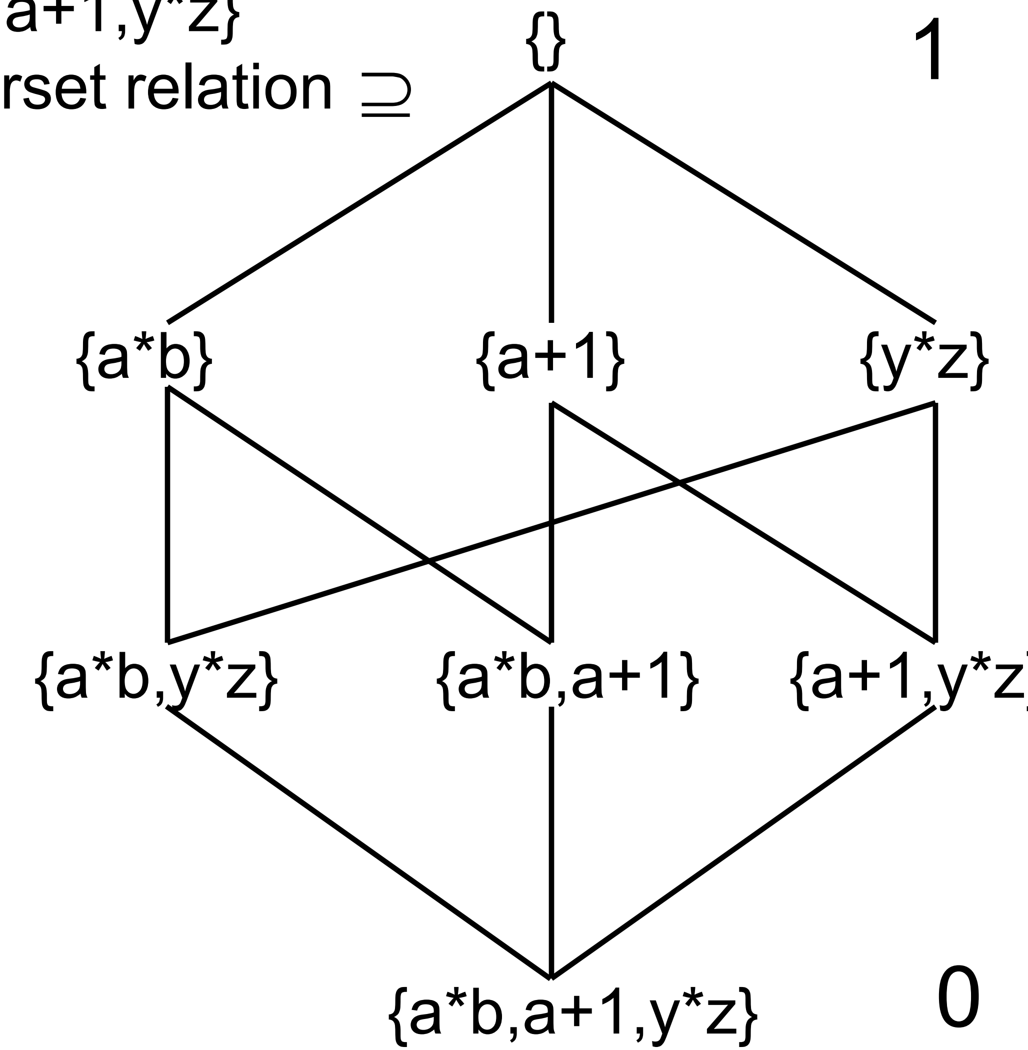


Safe Solutions: *Avail*

U = all expressions: {a*b,a+1,y*z}

Poset is 2^U, $\leq$ is the superset relation $\supseteq$





Precision of a Dataflow Solution

- Precise solution is one that is “close” to MOP
  - A precise solution contains few spurious dataflow facts (spurious facts is what we call noise)
  - Unfortunately, for most problems even the MOP (an approximation itself!) is undecidable

- MOP ≤ X ≤ Y: X is more precise than Y
  - Usually, we can compare two solutions X and Y
  - But, for most problems, we have no way of knowing the “ground truth”

Next class: real analyses

- Next time: non-distributive analyses
  - Constant propagation
  - Pointer analysis