Lecture 2 - Outline

- Theoretic foundations for dataflow analysis
  - Lattices, fixed point theorem
  - Convergence, complexity, precision, safety properties
  - References: Hecht book, Marlowe & Ryder Acta Informatica 1990 paper

Theoretical Foundations of Static Analysis

- Dataflow analysis is definable as related to abstract interpretation; we prefer to use the more operational definition of dataflow analysis
- Lattice theory and partially ordered sets
- Solution procedures for dataflow equations
  - Function properties for convergence
  - Fixed-point iteration
  - Worklist algorithm

Questions

- How do we solve these dataflow eqns?
  - How do we know that a solution exists?
  - How do we know how quickly a solution can be found?
- How do we formulate other useful dataflow problems?
  - What do we need to define to formulate a dataflow analysis?
- How do we define dataflow problems that involve method calls (interprocedural) to prevent following infeasible paths?

Answers

- Firm, mathematical foundations underlie dataflow analysis
  - Lattice theory, partially ordered sets
  - Functions with specific properties to ensure convergence
    - Fixed point theorem provides solution procedure
  - Specific fixed-point iteration procedure to find a solution
- Serves as underpinnings of all static analyses in compilation
  - But not necessary to explain all analyses using this formalism
  - Need to understand there is mathematical justification for dataflow analysis

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
- Partially ordered set (Set S, £)
- 0 Element 0 £ x, ∀ x ∈ S
- 1 Element 1 ≥ ∀ x ∈ S

A partially ordered set need not have 0 or 1 element.

Lattice Theory

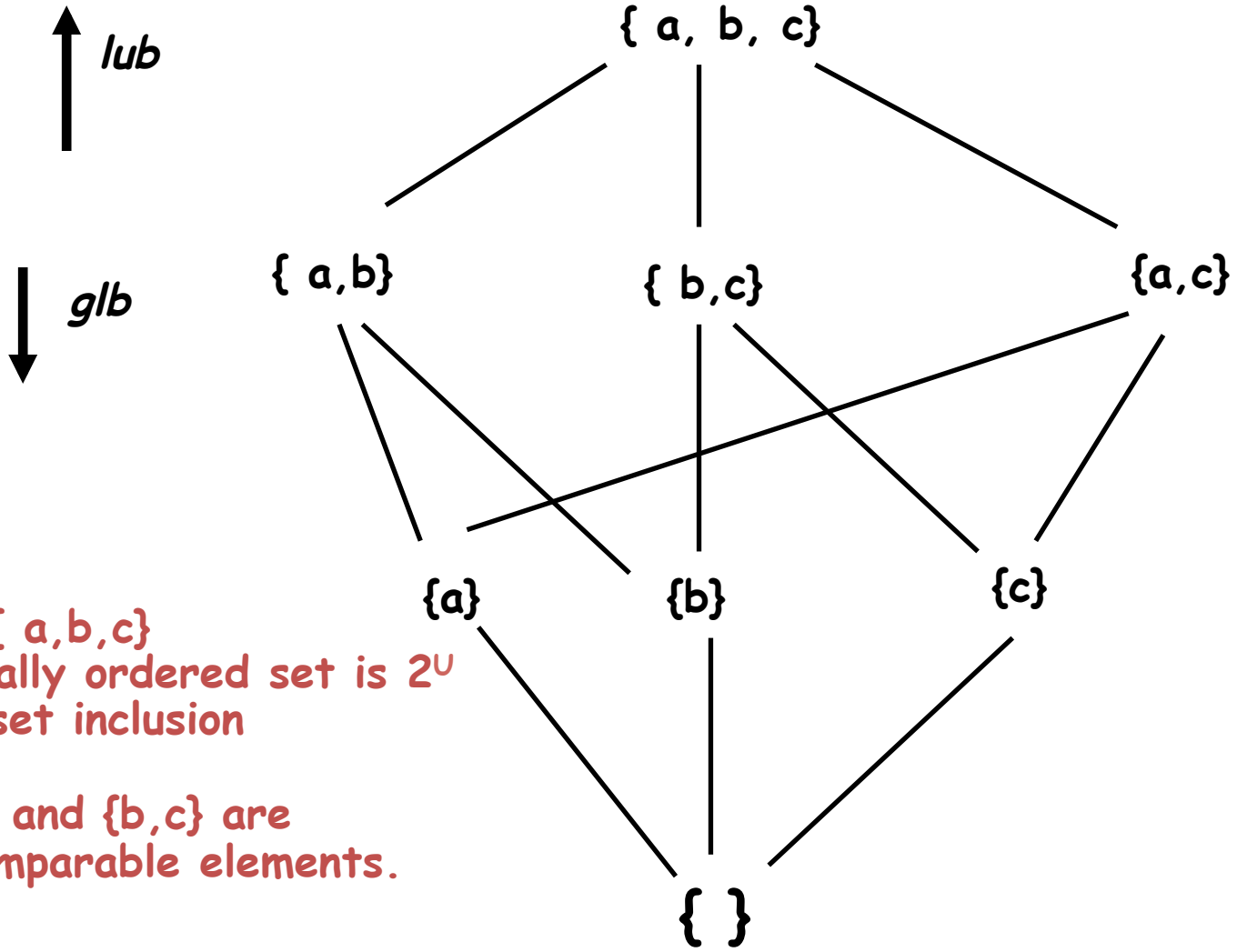
- Greatest lower bound (glb)
  a, b ∈ partially ordered set S, c ∈ S is glb(a, b)
  if c £ a and c £ b then
  for any z ∈ S, z £ a, z £ b ⇒ z £ c

if glb is unique it is called the meet (Λ) of a and b

- Least upper bound (lub)
  a, b ∈ partially ordered set S, c ∈ S is lub(a,b)
  if c ≥ a and c ≥ b then
  for any d ∈ S, d ≥ a, d ≥ b ⇒ c £ d.

if lub is unique is called the join (υ) of a and b

Partially Ordered Set Example



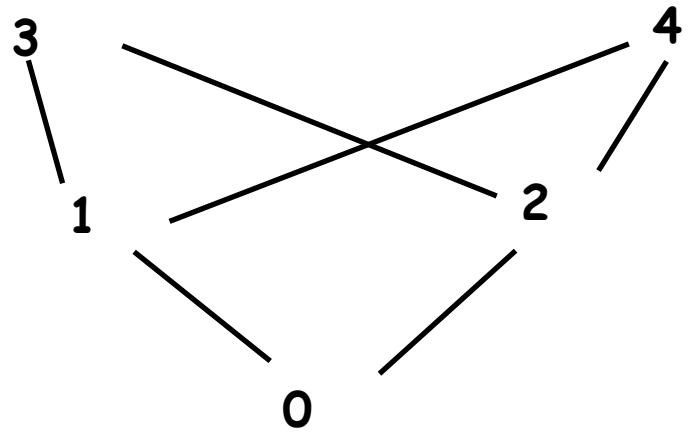
U = { a,b,c}
partially ordered set is 2^U
≤ is set inclusion

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

Definition of a Lattice (L, Λ, υ)

- L, a partially ordered set under £ 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 partially ordered set is a lattice.
- If a £ x, $\forall$ x $\in$ L, then a is 0 element of L
- If x £ a, $\forall$ x $\in$ L, then a is 1 element of L

a partially ordered set, but not a lattice



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

Examples of Lattices

- H = (2^U , ∩, ∪) where U is a finite set
  - glb (s1, s2) is (s1 Λ s2) which is s1 ∩ s2
  - lub (s1, s2) is (s1 υ 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 υ n2 = lowest common multiple(n1,n2)
  - glb (n1,n2) is n1 Λ n2 = greatest common divisor (n1,n2)

Chain

- A partially ordered set C where, for every pair of elements

c1, c2 ∈ C, either c1 £ c2 or c2 £ c1.

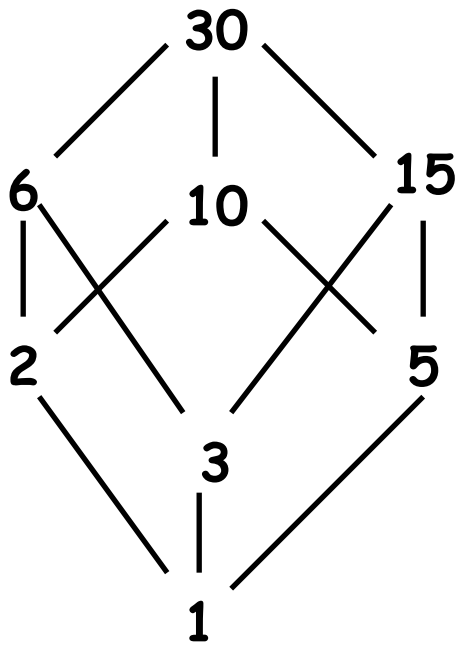
e.g., { } £ {a} £ {a,b} £ {a,b,c}

and from the lattice as shown here,

1 £ 2 £ 6 £ 30

1 £ 3 £ 15 £ 30

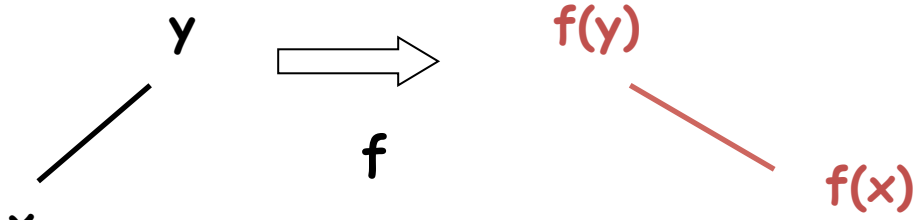
Lattices are used in dataflow analysis to argue the existence of a solution obtainable through fixed-point iteration.



***Finite length lattice*: if every chain in lattice is finite**

Functions on a Lattice

- (S,£) partially ordered set, f: S --> S is monotonic iff
  $\forall x, y \in S,\ x \pounds y \Rightarrow f(x) \pounds f(y)$
- Monotonic functions preserve domain element ordering in their range values



- Distributive functions allow function application to distribute over the meet

  $\forall x, y \in S,\ f(x) \Lambda f(y) = f(x \Lambda y)$

  Classical dataflow problems are all distributive

  Distributive implies monotone (try to prove this)

Solve by Fixed-point Iteration

Schema:

Dataflow equations defined on each CFG node

Initialize variables in equations.

Find all nodes where the equations are not yet satisfied, and assign new values to variables

Continue until all equations are satisfied.

Why this works?

Need valid choice of initial values.

Need dataflow equations to define monotone functions on the lattice of solutions

Use the fixed-point theorem to justify convergence

Classical usage – bit vector problems in compilation

Fixed-point Iteration for Reach

initialize Reach(m) = ∅;
change = true;
while (change) *do*
{ change = false;
while (∃ j ∋ Reach(j) ≠ $\bigcup_{m \in pred(j)}$ (Reach(m) ∩ pres(m) ∪ dgen(m)) *do*

{ Reach(j) = $\bigcup_{m \in pred(j)}$ (Reach(m) ∩ pres(m) ∪ dgen(m)) ;

change = true;
}
}

Advanced Dataflow Analysis

- Why fixed-point iteration works?
- Practical fixed-point algorithms
- Properties of a solution to a dataflow analysis problem
    - MOP versus MFP
- Solution safety

Fixed point theorem - Why it works?

Intuition:

Given a 0 in lattice and monotonic function f, $0 \leq f(0)$.

Apply f again and obtain

$0 \leq f(0) \leq f(f(0)) = f^2(0)$

Continuing,

$0 \leq f(0) \leq f^2(0) \leq f^3(0) \leq \ldots \leq f^k(0) = f^{k+1}(0)$ for a finite chain lattice.

This is tantamount to saying

$\lim_{k \Rightarrow \infty} f^k(0)$ exists and is called the *least fixed point* of f,

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

$k \Rightarrow \infty$

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. $f^0(x) = x$,

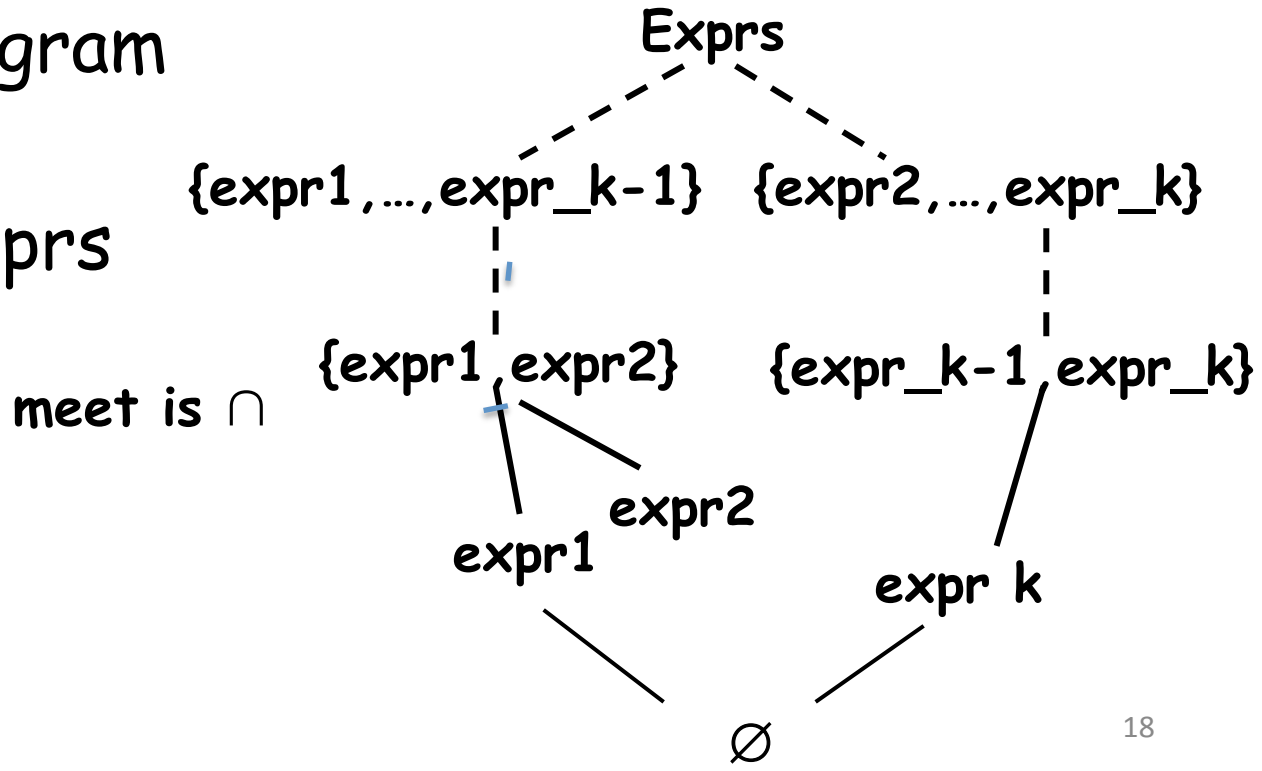
ii. $f^{i+1}(x) = f(f^i(x))$, $i \geq 0$,

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 $f^k(1) = f^{k+1}(1)$.

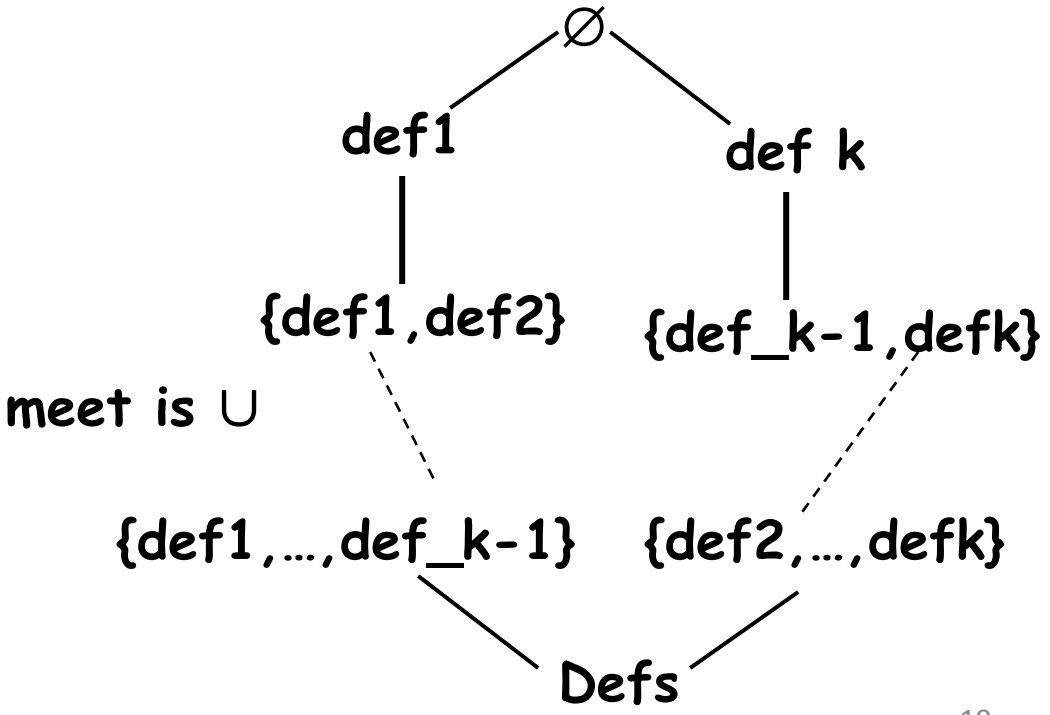
AVAIL

- AVAIL meet operation is set intersection with partial order subset inclusion
  - Why? recall that the 0 element α is such that
    $\alpha \leq x, \forall x$ which means α is a subset of x!
- Exprs = {<node, binary expression>}, all Exprs in program
- 0 element $\varnothing$
- 1 element is Exprs



Reaching Definitions

- REACH meet operation is set union with partial order is $\subseteq$ superset inclusion
  - Why? recall that the 0 element α is such that $\alpha \leq x, \forall x$ which means α is a superset of x!
- Defs = {<node,var>}, all defs in program
- 0 element Defs
- 1 element is $\emptyset$



Worklist Algorithm: A Practical Version of Fixed-point Iteration

$$Reach(j) = \bigcup_{m \in Pred(j)} \{ Reach(m) \cap pres(m) \cup dgen(m) \}$$

Initialize all CFG nodes to ∅.
Put all nodes on the worklist W.
Loop: Do until W is empty{
    remove a node from the worklist W;
    calculate right-hand-side of above eqn;
    compare result with Reach(j)
      if result is different, {update Reach(j) and
        put descendent nodes of j on worklist W}
    }
 //when terminates have correct reaching definitions
   solution at each node

Monotone Dataflow Frameworks

- Formalism for expressing and categorizing data flow problems (Kildall, POPL 1973) <G, L, F, M>
  - G, flowgraph <N, E, ρ>
  - L, (semi-)lattice with meet Λ
    - usually assume L has a 0 and 1 element
    - finite chains
  - F, function space, $\forall f \in F$, f: L --> L
    - Contains identity function
    - Closed under composition $\forall f, g \in F, f \circ g \in F$
    - Closed under point-wise meet, if $h(x) = f(x) \Lambda g(x)$ then $h \in F$
  - M : E --> F, maps an edge to a corresponding transfer function that describes dataflow effect of traversing that edge

Function Properties that Guarantee a Solution

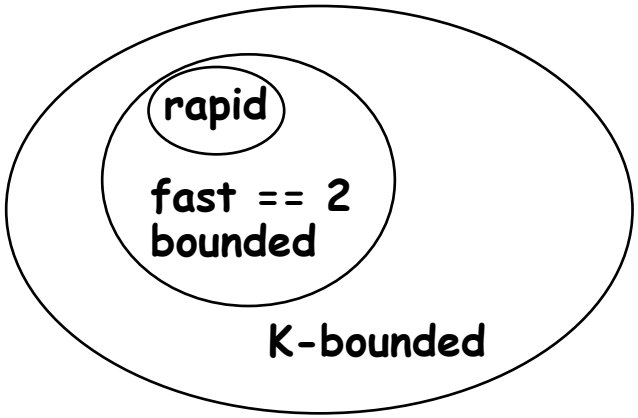
- *Monotonicity*
  - Defined as $x \pounds y \Rightarrow f(x) \leq f(y)$.
  - Equivalent formulation of definition
    $f(x \wedge y) \pounds f(x) \wedge f(y)$
- *Distributivity*
  - If $f(x \wedge y) = f(x) \wedge f(y)$ then f *distributive*
  - Distributivity implies monotonicity
  - Four classical bitvector problems are distributive

Function Properties that Specify Speed of Convergence

K-bounded: all contributions to MFP solution occur prior to Kth iteration

Fast: 1 pass around a cycle is enough to summarize its contribution to the dataflow solution (e.g., reflexive transitive closure is fast but not rapid)

Rapid: contribution of a cycle is independent of value at entry node; 1 pass around the cycle is enough. All classical bitvector problems are rapid



Meet Over all Paths Solution (MOP)

- Imagine full knowledge of all possible executions of a program and therefore all its execution paths
- Dataflow analysis summarizes what can happen with respect to certain program properties – over all possible program paths
- Challenge: Deciding if a static execution path is actually feasible is Turing complete.
    - So we assume all paths in our compile-time program representation are executable
    - Some analyses have been developed to refine this assumption

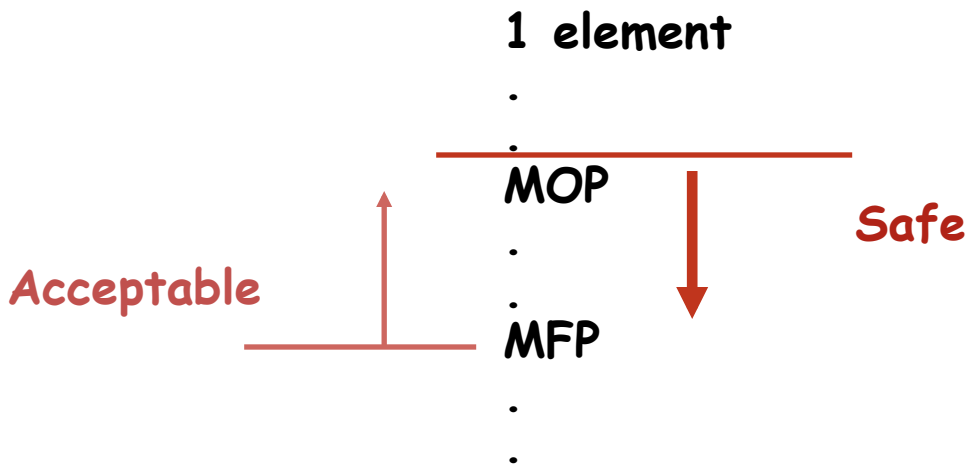
MOP vs MFP

- If *distributive* functions define the dataflow problem, to obtain dataflow solution at node n, can gather information on paths (e.g., P1, P2) simultaneously without loss of precision.
  - e.g., $f_{P1}(0)$, $f_{P2}(0)$ needn't be calculated explicitly
- However, Kam and Ullman showed that this is not true for all *monotone* functions; Kam, Ullman, 1976,1977
- Therefore, MFP only approximates MOP for general monotone functions that are not distributive.

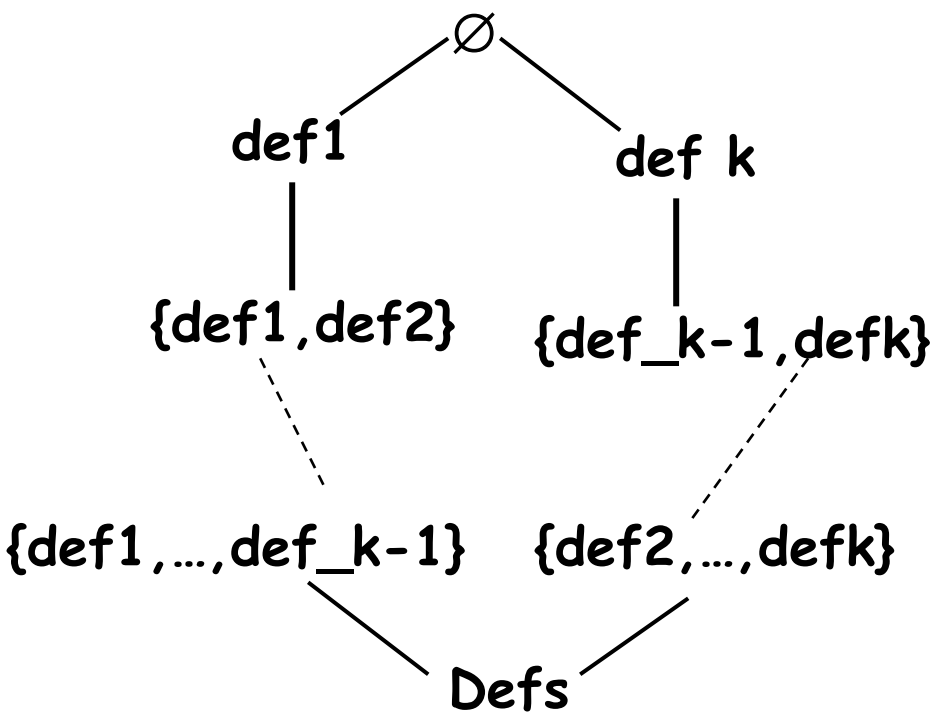
Safety of Dataflow Solution

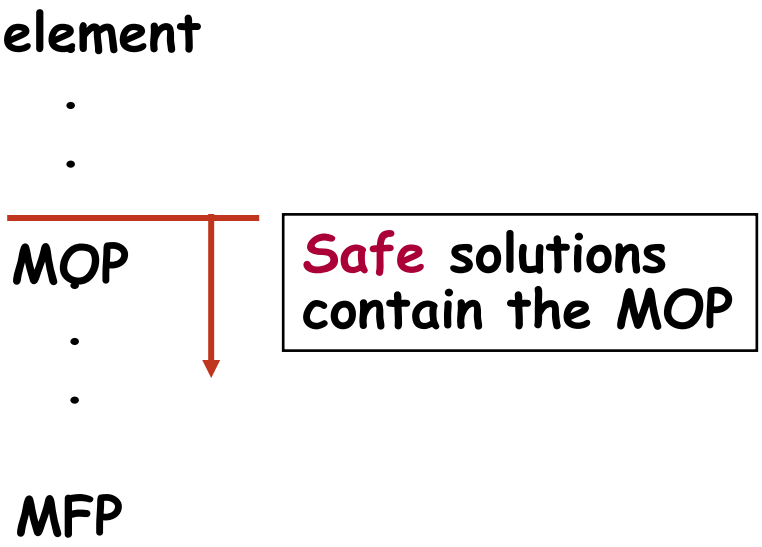
- Safe solution underestimates the actual dataflow solution; x £ MOP is an approximate solution
- Acceptable solution is one that contains a fixed point of the function, y ≥ z where z is any fixed point.
- If they exist, MOP is largest safe solution and MFP is smallest acceptable solution.
- Between MFP and MOP

are interesting solutions.



Reaching Definitions





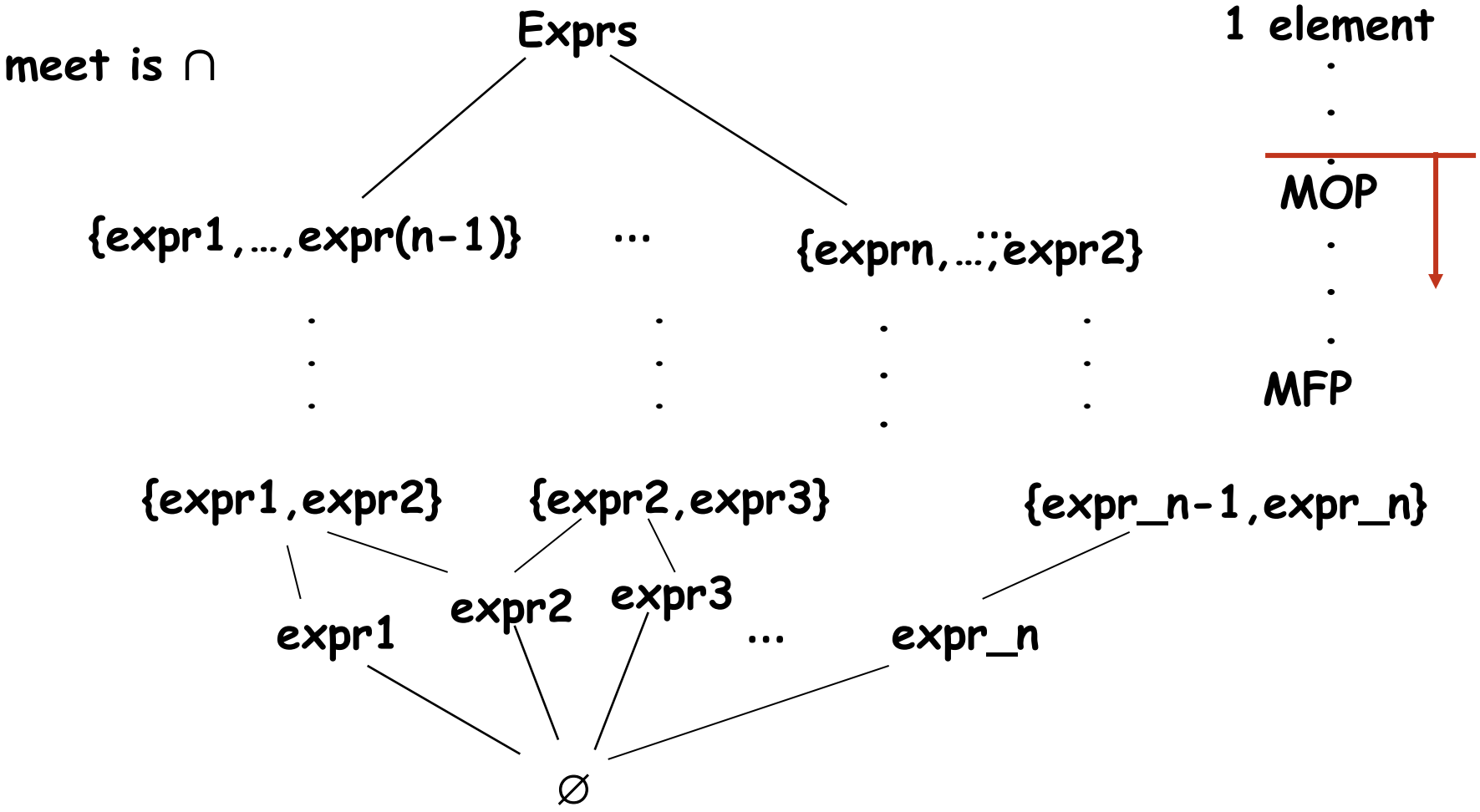
REACH: it is safe to err by saying a definition reaches when it DOES NOT REACH.

E.g., this may inhibit dead code elimination transformations

But since REACH functions are distributive, MOP=MFP here

Available Expressions

Safe solutions contain the MOP



AVAIL: it is safe to err by saying an expression is NOT AVAILABLE when it might be.
This may inhibit *common subexpression elimination* transformations
Since AVAIL functions are distributive, MOP=MFP here