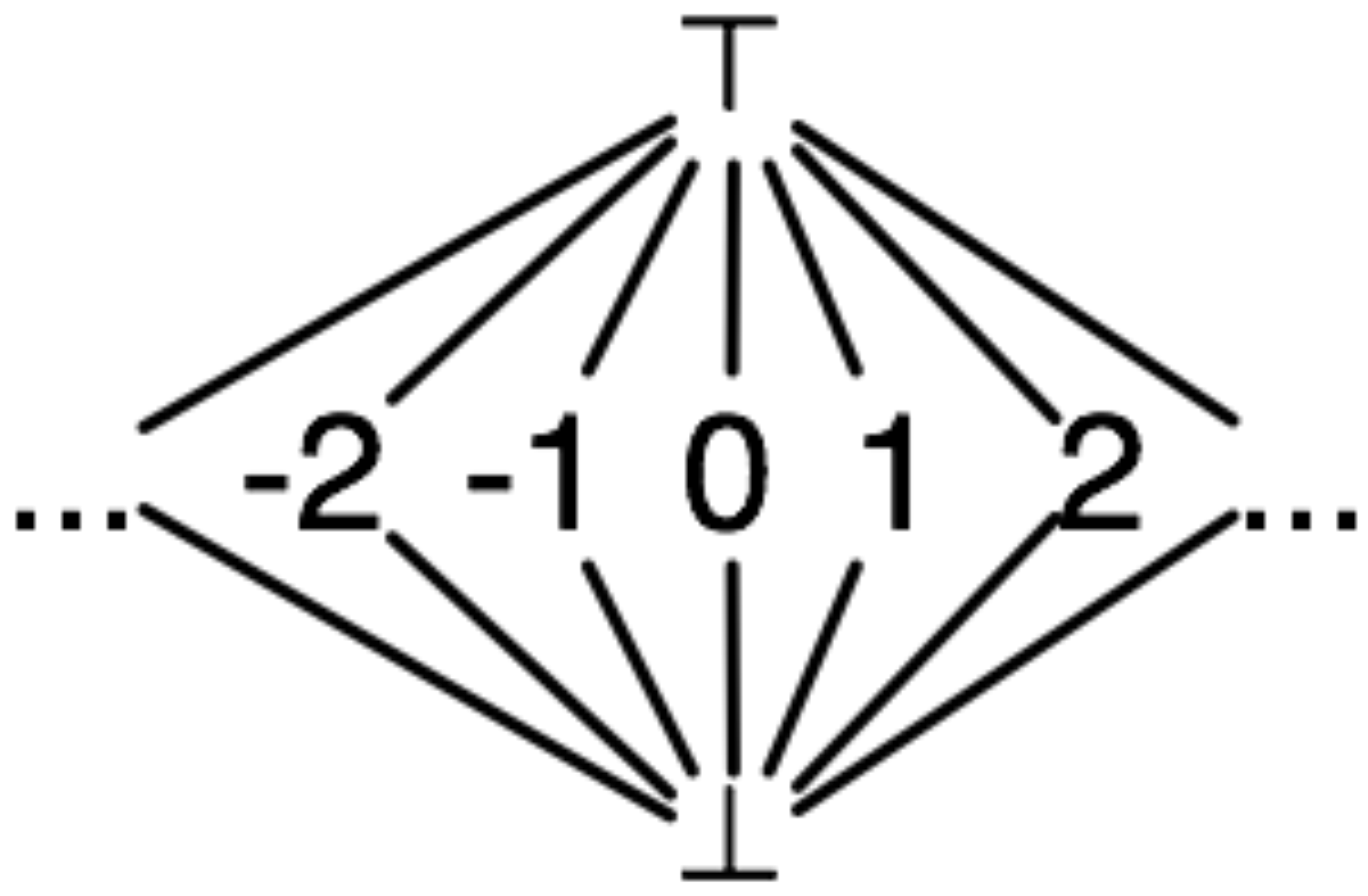
Lattices

Lattices

- A bounded *lattice* is a partially ordered set with a $\perp$ and $\top$, with two special functions for any pair of points *x* and *y* in the lattice:
  - A *join*: $x \sqcup y$ is the least element that is greater than x and y (also called the *least upper bound*)
  - A *meet*: $x \sqcap y$ is the greatest element that is less than x and y (also called the *greatest lower bound*)
- Are $\sqcup$ and $\sqcap$ monotonic?
  - Yes! (proof?)
  - If $x \sqsubseteq x'$, then both $x \sqcup y$ and $x' \sqcup y$ are fixpoints, as $x \sqcup y$ is the least fixpoint, $x \sqcup y \sqsubseteq x' \sqcup y$

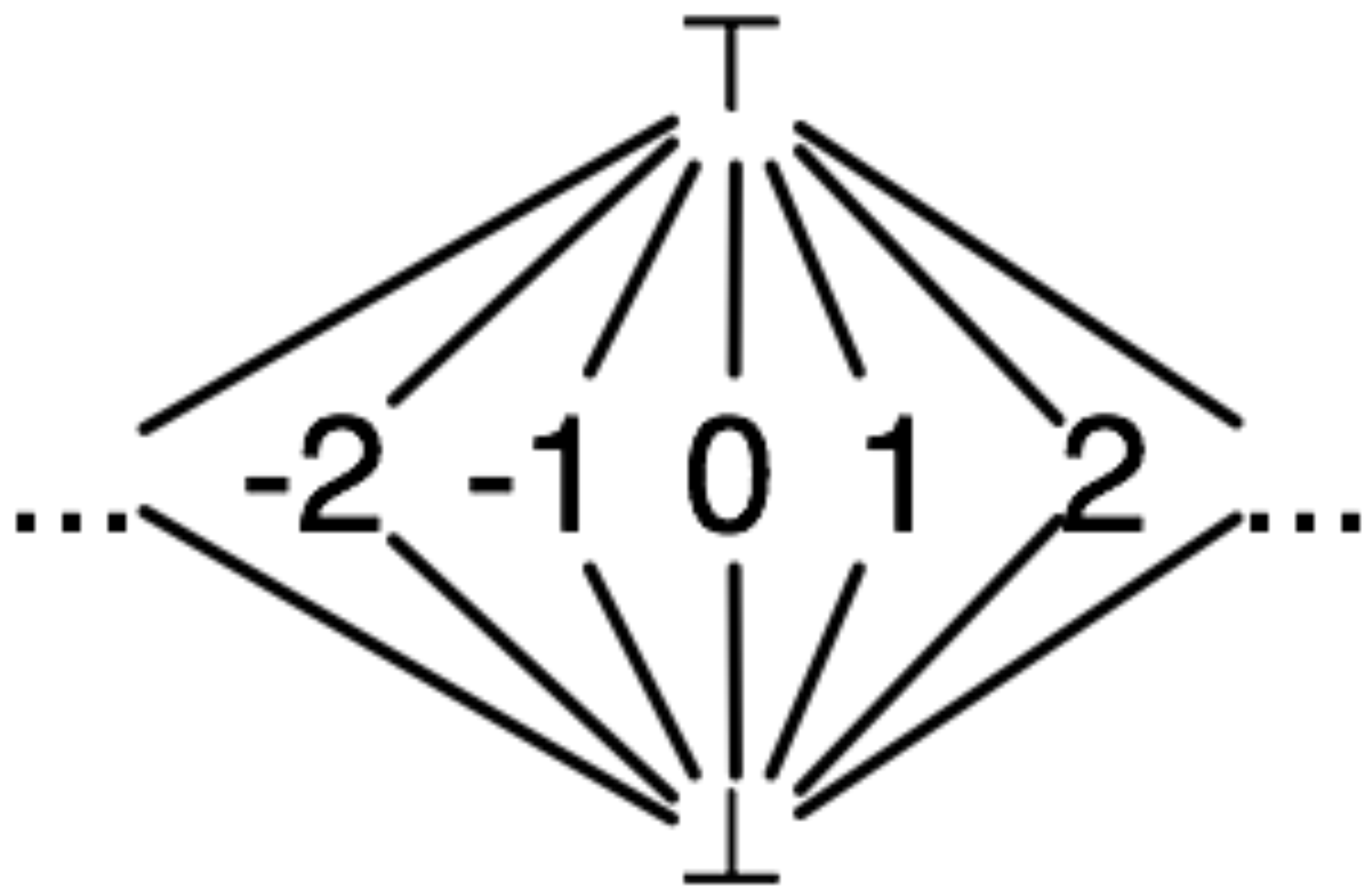
Lattice for constant propagation

- Three “levels”
  - Top (definitely not constant)
  - Middle (any specific constant)
  - Bottom (no information)
- Join in lattice: $x \sqcup \top = \top$ or $x \sqcup y = \top$



More about lattices

- Bounded lattices with a finite number of elements are a special case of domains with ⊤
- Systems of monotonic functions (including ⊔ and ⊓) will have fixpoints
- But some lattices are infinite! (example: the lattice for constant propagation)
  - It turns out that you can show a monotonic function will have a least fixpoint for any lattice (or domain) of *finite height*
  - Finite height: any totally ordered subset of domain (this is called a *chain*) must be finite
  - Why does this work?
    - ⊥, f(⊥), f(f(⊥)), f(f(f(⊥))) ... is totally ordered



Solving system of equations

- Consider

  a = f(a, b, c)

  b = g(a, b, c)

  c = h(a, b, c)

- Obvious iterative solution: evaluate every function at every step

  a = ⊥   f(⊥,⊥,⊥) f(f(⊥,⊥,⊥), g(⊥,⊥,⊥), h(⊥,⊥,⊥)) ...

  b = ⊥   g(⊥,⊥,⊥) g(f(⊥,⊥,⊥), g(⊥,⊥,⊥), h(⊥,⊥,⊥)) ...

  c = ⊥   h(⊥,⊥,⊥) h(f(⊥,⊥,⊥), g(⊥,⊥,⊥), h(⊥,⊥,⊥)) ...

Worklist algorithm

- Obvious point: only necessary to re-evaluate functions whose “important” inputs have changed
- Worklist algorithm
  - Initialize worklist with all equations
  - Initialize solution vector S to all ⊥
  - While worklist not empty
    - Get equation from worklist
    - Re-evaluate equation based on S, update entry corresponding to lhs in S
    - Put all equations which use this lhs on their rhs in the worklist
- Claim: this is basically how constant propagation works!

Constant propagation as fixpoint

- Functions map a vector of variable values <x, y, z> to another vector of variable values
  - Program statements: eval(e, V_{in})
    - These are called *transfer functions*
    - Need to make sure this is monotonic
  - Branches
    - Propagates input state vector to output – trivially monotonic
  - Merges
    - Use join or meet to combine multiple input variables – monotonic by definition

Mapping worklist algorithm

- Careful: the “variables” in constant propagation are not the individual variable values in a state vector. Each variable (from a fixpoint perspective) is an entire state vector – there are as many variables as there are edges in the CFG
- Initialize all “variables” (state vectors) to $< \perp, \perp, \perp >$
- Executing a statement uses one (or more) input state vectors, produces an output state vector
- Running worklist algorithm for finding fix point is the same as running symbolic execution until state vectors converge!

next: more dataflow analysis