Given a poset \((S, \leq)\), and two elements \(a \in S\) and \(b \in S\), then the:

- **least upper bound (lub)** is an element \(c\) such that \(a \leq c, b \leq c, \) and \(\forall d \in S . (a \leq d \land b \leq d) \Rightarrow c \leq d\)
- **greatest lower bound (glb)** is an element \(c\) such that \(c \leq a, c \leq b, \) and \(\forall d \in S . (d \leq a \land d \leq b) \Rightarrow d \leq c\)

**lub and glb don’t always exists:**

A lattice is a tuple \((S, \sqsubseteq, \sqcap, \sqcup, \sqcap, \sqcup)\) such that:

- \((S, \sqsubseteq)\) is a poset
- \(\forall a \in S . \sqcap \sqsubseteq a\)
- \(\forall a \in S . a \sqcap \sqcup\)
- Every two elements from \(S\) have a lub and a glb
- \(\sqcup\) is the least upper bound operator, called a join
- \(\sqcap\) is the greatest lower bound operator, called a meet

**Examples of lattices**

- **Powerset lattice**

"Lub and glb"
Examples of lattices

- Booleans expressions

\[
\begin{align*}
\top & \leq a \leq \bot \\
\top & \leq b \leq \bot \\
\top & \leq c \leq \bot \\
\top & \leq d \leq \bot \\
\end{align*}
\]

Back to our example

let m: map from edge to computed value at edge
let worklist: work list of nodes

for each edge e in CFG do
\[ m(e) := \delta \]

for each node n do
\[ \text{worklist.add}(n) \]

while (worklist.empty.not) do
\[ \text{let } n := \text{worklist.remove_any}; \]
\[ \text{let info_in := } m(\text{n.incoming_edges}); \]
\[ \text{let info_out := } F(\text{n}, \text{info_in}); \]
\[ \text{for } i := 0 .. \text{info_out.length do} \]
\[ \text{let new_info := } m(\text{n.outgoing_edges[i]}); \]
\[ \text{if } (m(\text{n.outgoing_edges[i]}) \neq \text{new_info}) \]
\[ m(\text{n.outgoing_edges[i]}) := \text{new_info}; \]
\[ \text{worklist.add}(\text{n.outgoing_edges[i].dst}); \]

End of background material
Back to our example

• We formalize our domain with a powerset lattice
• What should be top and what should be bottom?

Direction of lattice

• Unfortunately:
  – dataflow analysis community has picked one direction
  – abstract interpretation community has picked the other
• We will work with the abstract interpretation direction
• Bottom is the most precise (optimistic) answer, Top the most imprecise (conservative)

Worklist algorithm using lattices

```plaintext
let m: map from edge to computed value at edge
let worklist: work list of nodes

for each edge e in CFG do
    m(e) := \top

for each node n do
    worklist.add(n)

while (worklist.empty.not) do
    let n := worklist.remove_any;
    let info_in := m(n.incoming_edges);
    let info_out := F(n, info_in);
    for i := 0 .. info_out.length do
        let new_info := m(n.outgoing_edges[i])
        if (m(n.outgoing_edges[i])  new_info)
            m(n.outgoing_edges[i]) := new_info;
        worklist.add(n.outgoing_edges[i].dst);
```

Back to our example

• We formalize our domain with a powerset lattice
• What should be top and what should be bottom?

Direction of lattice

• Always safe to go up in the lattice
• Can always set the result to \top
• Hard to go down in the lattice
• So ... Bottom will be the empty set in reaching defs
Termination of this algorithm?

- For reaching definitions, it terminates...
- Why?
  - lattice is finite
  - Can we loosen this requirement?
  - Yes, we only require the lattice to have a finite height
  - Height of a lattice: length of the longest ascending or descending chain
  - Height of lattice \((2^S, \subseteq) = \)

Even more formal

- To reason more formally about termination and precision, we re-express our worklist algorithm mathematically
  - We will use fixed points to formalize our algorithm

Fixed points

- Recall, we are computing \(m\), a map from edges to dataflow information
- Define a global flow function \(F\) as follows: \(F\) takes a map \(m\) as a parameter and returns a new map \(m'\), in which individual local flow functions have been applied
  
  \[
  \text{Fixed points}
  \]

  \[
  \begin{align*}
  \text{while } & (\text{worklist}.\text{empty}.\text{not}) \text{ do} \\
  & \text{let } n := \text{worklist}.\text{remove}.\text{any} \\
  & \text{let } \text{info}_\text{in} := m(n.\text{incoming}.\text{edges}) \\
  & \text{let } \text{info}_\text{out} := F(n, \text{info}_\text{in}) \\
  & \text{for } i := 0 \ldots \text{info}_\text{out}.\text{length} \text{ do} \\
  & \quad \text{let } \text{new}_\text{info} := m(n.\text{outgoing}.\text{edges}[i]) \cup \\
  & \quad \quad \text{info}_\text{out}[i] \\
  & \quad \text{if } (m(n.\text{outgoing}.\text{edges}[i]) \neq \text{new}_\text{info}) \\
  & \quad \quad m(n.\text{outgoing}.\text{edges}[i]) := \text{new}_\text{info} \\
  & \quad \text{worklist}.\text{add}(n.\text{outgoing}.\text{edges}[i].\text{dst})
  \end{align*}
\]
- It would be nice to get rid of it, if there is a property of the flow functions that would allow us to do so

- We want to find a fixed point of \(F\), that is to say a map \(m\) such that \(m = F(m)\)
- Approach to doing this?
  - Define \(\lambda\), which is \(\bot\) lifted to be a map:
    \[
    \lambda = \lambda_e \perp
    \]
  - Compute \(F(\lambda)\), then \(F(F(\lambda))\), then \(F(F(F(\lambda)))\), ... until the result doesn’t change anymore

Termination

- Still, it’s annoying to have to perform a join in the worklist algorithm

- It would be nice to get rid of it, if there is a property of the flow functions that would allow us to do so
Fixed points

- Formally:
  \[ S_{\text{fin}} = \bigcup_{i=0}^{\infty} F^i(\mathcal{I}) \quad \alpha \subseteq F(\mathcal{I}) \]

- We would like the sequence \( F(\mathcal{I}) \) for \( i = 0, 1, 2 \) ... to be increasing, so we can get rid of the outer join
  
  Require that \( F \) be monotonic: 
  \[ \alpha \subseteq \mathcal{I} \Rightarrow F(\alpha) \subseteq F(\mathcal{I}) \]

Back to termination

- So if \( F \) is monotonic, we have what we want: finite height \( \Rightarrow \) termination, without the outer join
- Also, if the local flow functions are monotonic, then global flow function \( F \) is monotonic