Lattices

- Lattice:
  - Set augmented with a partial order relation \( \sqsubseteq \)
  - Each subset has a LUB and a GLB
  - Can define: meet \( \sqcap \), join \( \sqcup \), top \( \top \), bottom \( \bot \)

- Use lattice in the compiler to express information about the program

- To compute information:
  - build constraints which describe how the lattice information changes
  - Effect of instructions: transfer functions
  - Effect of control flow: meet operation

Properties of Meet and Join

- The meet and join operators are:
  1. Associative \( (x \sqcap y) \sqcap z = x \sqcap (y \sqcap z) \)
  2. Commutative \( x \sqcap y = y \sqcap x \)
  3. Idempotent: \( x \sqcap x = x \)

- Property: If "\( \sqcap \)" is an associative, commutative, and idempotent operator, then the relation "\( \sqsubseteq \)" defined as \( x \sqsubseteq y \) iff \( x \sqcap y = x \) is a partial order

- Above property provides an alternative definition of a partial orders and lattices starting from the meet (join) operator

Transfer Functions

- Let \( L = \) dataflow information lattice

- Transfer function \( F_i : L \rightarrow L \) for each instruction \( I \)
  - Describes how \( I \) modifies the information in the lattice
  - If \( \text{in}[I] \) is info before \( I \) and \( \text{out}[I] \) is info after \( I \), then
    - Forward analysis: \( \text{out}[I] = F_i(\text{in}[I]) \)
    - Backward analysis: \( \text{in}[I] = F_i^{-1}(\text{out}[I]) \)

- Transfer function \( F_B : L \rightarrow L \) for each basic block \( B \)
  - Is composition of transfer functions of instructions in \( B \)
  - If \( \text{in}[B] \) is info before \( B \) and \( \text{out}[B] \) is info after \( B \), then
    - Forward analysis: \( \text{out}[B] = F_B(\text{in}[B]) \)
    - Backward analysis: \( \text{in}[B] = F_B^{-1}(\text{out}[B]) \)

Monotonicity and Distributivity

- Two important properties of transfer functions
  - Monotonicity: function \( F : L \rightarrow L \) is monotonic if \( x \sqsubseteq y \) implies \( F(x) \sqsubseteq F(y) \)
  - Distributivity: function \( F : L \rightarrow L \) is distributive if \( F(x \sqcap y) = F(x) \sqcap F(y) \)
  - Property: \( F \) is monotonic iff \( F(x \sqcap y) \sqsupseteq F(x) \sqcap F(y) \)
  - any distributive function is monotonic!

Proof of Property

- Prove that the following are equivalent:
  1. \( x \sqsubseteq y \) implies \( F(x) \sqsubseteq F(y) \), for all \( x, y \)
  2. \( F(x \sqcap y) \sqsupseteq F(x) \sqcap F(y) \), for all \( x, y \)

- Proof for "1 implies 2"
  - Need to prove that \( F(x \sqcap y) \sqsubseteq F(x) \) and \( F(x \sqcap y) \sqsubseteq F(y) \)
  - Use \( x \sqcap y \sqsubseteq x \) and \( x \sqcap y \sqsubseteq y \), and property 1

- Proof of "2 implies 1"
  - Let \( x, y \) such that \( x \sqsubseteq y \)
  - Then \( x \sqcap y = x \), so \( F(x \sqcap y) = F(x) \)
  - Use property 2 to get \( F(x) \sqsupseteq F(x) \sqcap F(y) \)
  - Hence \( F(x) \sqsubseteq F(y) \)
Control Flow
- **Meet operation** models how to combine information at split/join points in the control flow
  - If \( \text{in}(B) \) is info before \( B \) and \( \text{out}(B) \) is info after \( B \), then:
    - **Forward analysis**: \( \text{in}(B) = \sqcap\{\text{out}(B') | B' \preceq \text{pred}(B)\} \)
    - **Backward analysis**: \( \text{out}(B) = \sqcap\{\text{in}(B') | B' \succeq \text{succ}(B)\} \)
- Can alternatively use join operation \( \sqcup \) (equivalent to using the meet operation \( \sqcap \) in the reversed lattice)

Monotonicity of Meet
- **Meet operation** is also monotonic over \( \mathbb{L} \times \mathbb{L} \):
  \[ x_1 \sqsubseteq y_1 \text{ and } x_2 \sqsubseteq y_2 \implies (x_1 \sqcap x_2) \sqsubseteq (y_1 \sqcap y_2) \]
- **Proof**:
  - Any lower bound of \( \{x_1, x_2\} \) is also a lower bound of \( \{y_1, y_2\} \), because \( x_1 \sqsubseteq y_1 \) and \( x_2 \sqsubseteq y_2 \)
  - \( x_1 \sqcap x_2 \) is a lower bound of \( \{x_1, x_2\} \)
  - So \( x_1 \sqcap x_2 \) is a lower bound of \( \{y_1, y_2\} \)
  - But \( y_1 \sqcap y_2 \) is the greatest lower bound of \( \{y_1, y_2\} \)
  - Hence \( (x_1 \sqcap x_2) \sqsubseteq (y_1 \sqcap y_2) \)

Forward Dataflow Analysis
- **Control flow graph** \( G \) with entry (start) node \( B_s \)
- **Lattice** \( (\mathbb{L}, \sqsubseteq) \) represents information about program
  - Meet operator \( \sqcap \), top element \( \top \)
- **Monotonic transfer functions**
  - Transfer function \( F: \mathbb{L} \to \mathbb{L} \) for each instruction \( I \)
  - Can derive transfer functions \( F_B \) for basic blocks
- **Goal**: compute the information at each program point, given the information at entry of \( B_s \) is \( X_0 \)
- Require the solution to \( \text{out}(B) = F_B(\text{in}(B)), \) for all \( B \)
- Require the solution to \( \text{in}(B) = \sqcap\{\text{out}(B') | B' \preceq \text{pred}(B)\}, \) for all \( B \)
- Require the solution to \( \text{out}(B) = \sqcap\{\text{in}(B') | B' \succeq \text{succ}(B)\}, \) for all \( B \)

Backward Dataflow Analysis
- **Control flow graph** \( G \) with exit node \( B_e \)
- **Lattice** \( (\mathbb{L}, \sqsubseteq) \) represents information about program
  - Meet operator \( \sqcap \), top element \( \top \)
- **Monotonic transfer functions**
  - Transfer function \( F: \mathbb{L} \to \mathbb{L} \) for each instruction \( I \)
  - Can derive transfer functions \( F_B \) for basic blocks
- **Goal**: compute the information at each program point, given the information at exit of \( B_e \) is \( X_0 \)
- Require the solution to \( \text{in}(B) = F_B(\text{out}(B)), \) for all \( B \)
- Require the solution to \( \text{out}(B) = \sqcap\{\text{in}(B') | B' \preceq \text{pred}(B)\}, \) for all \( B \)
- Require the solution to \( \text{out}(B) = \sqcap\{\text{in}(B') | B' \succeq \text{succ}(B)\}, \) for all \( B \)

Dataflow Equations
- The constraints are called dataflow equations:
  \[
  \begin{align*}
  \text{out}(B) &= F_B(\text{in}(B)), \text{ for all } B \\
  \text{in}(B) &= \sqcap\{\text{out}(B') | B' \preceq \text{pred}(B)\}, \text{ for all } B \\
  \text{in}(B_s) &= X_0
  \end{align*}
  \]
- Solve equations: use an iterative algorithm
  - Initialize \( \text{in}(B) = X_0 \)
  - Initialize everything else to \( \top \)
  - Repeatedly apply rules
  - Stop when reach a fixed point

Algorithm
\[
\begin{align*}
\text{in}(B_e) &= X_0 \\
\text{out}(B) &= \top, \text{ for all } B \\
\text{Repeat} \\
&\quad \text{For each basic block } B = B_s \\
&\quad \quad \text{in}(B) = \sqcap\{\text{out}(B') | B' \preceq \text{pred}(B)\} \\
&\quad \text{For each basic block } B \\
&\quad \quad \text{out}(B) = F_B(\text{in}(B)) \\
\text{Until no change}
\end{align*}
\]
Efficiency

- Algorithm is inefficient
  - Effects of basic blocks re-evaluated even if the input information has not changed
- Better: re-evaluate blocks only when necessary
- Use a worklist algorithm
  - Keep list of blocks to evaluate
  - Initialize list to the set of all basic blocks
  - If out[B] changes after evaluating out[B] = F_B(in[B]), then add all successors of B to the list

Worklist Algorithm

\[ \text{in}(B) = X_B \]
\[ \text{out}(B) = T, \text{ for all } B \]
\[ \text{worklist} = \text{set of all basic blocks } B \]

Repeat
- Remove a node B from the worklist
- \[ \text{in}(B) = \bigcap \{ \text{out}(B') \mid B' \text{ pred}(B) \} \]
- \[ \text{out}(B) = F_B(\text{in}(B)) \]
- If out(B) has changed, then
  - worklist \[ = \text{worklist} \cup \text{succ}(B) \]
Until worklist \[ = \varnothing \]

Correctness

- Initial algorithm is correct
  - If dataflow information does not change in the last iteration, then it satisfies the equations
- Worklist algorithm is correct
  - Maintains the invariant that
    \[ \text{in}(B) = \bigcap \{ \text{out}(B') \mid B' \text{ pred}(B) \} \]
    \[ \text{out}(B) = F_B(\text{in}(B)) \]
  - for all the blocks B not in the worklist
  - At the end, worklist is empty

Termination

- Do these algorithms terminate?
- Key observation: at each iteration, information decreases in the lattice
  \[ \text{in}_{k+1}(B) \subseteq \text{in}_k(B) \text{ and } \text{out}_{k+1}(B) \supseteq \text{out}_k(B) \]
  where \( \text{in}(B) \) is info before \( B \) at iteration \( k \) and \( \text{out}(B) \) is info after \( B \) at iteration \( k \)
- Proof by induction:
  - Induction basis: true, because we start with top element, which is greater than everything
  - Induction step: use monotonicity of transfer functions and meet operation
- Information forms a chain: \( \text{in}_k(B) \supseteq \text{in}_i(B) \supseteq \text{in}_j(B) \ldots \)

Chains in Lattices

- A chain in a lattice \( L \) is a totally ordered subset \( S \) of \( L \):
  \[ x \equiv y \text{ or } y \equiv x \text{ for any } x, y \in S \]
- In other words:
  Elements in a totally ordered subset \( S \) can be indexed to form an ascending sequence:
  \[ X_1 \equiv X_2 \equiv X_3 \equiv \ldots \]
  or they can be indexed to form a descending sequence:
  \[ X_1 \equiv X_2 \equiv X_3 \equiv \ldots \]
- Height of a lattice = size of its largest chain
- Lattice with finite height: only has finite chains

Termination

- In the iterative algorithm, for each block \( B \):
  \( (\text{in}(B), \text{in}_i(B), \ldots) \)
  is a chain in the lattice, because transfer functions and meet operation are monotonic
- If lattice has finite height then these sets are finite, i.e. there is a number \( k \) such that \( \text{in}(B) = \text{in}_k(B) \), for all \( i \geq k \) and all \( B \)
- If \( \text{in}(B) = \text{in}_k(B) \) then also \( \text{out}(B) = \text{out}_k(B) \)
- Hence algorithm terminates in at most \( k \) iterations
- To summarize: dataflow analysis terminates if
  1. Transfer functions are monotonic
  2. Lattice has finite height
Multiple Solutions

- The iterative algorithm computes a solution of the system of dataflow equations.
- ... is the solution unique?
- **No, dataflow equations may have multiple solutions!**

**Example:** live variables

Equations:
1. $11 = 12 - (y)$
2. $13 = (14 - (x)) U (y)$
3. $12 = 11 U 13$
4. $14 = ()$

**Solution 1:** $11 = (); 12 = (y); 13 = (y); 14 = ()$

**Solution 2:** $11 = (x); 12 = (x,y); 13 = (y); 14 = ()$

---

Safety

- **Solution for live variable analysis:**
  - Sets of live variables must include each variable whose values will further be used in some execution.
  - ... may also include variables never used in any execution!
- The analysis is **safe** if it takes into account all possible executions of the program.
  - ... may also characterize cases which never occur in any execution of the program.
  - Say that the analysis is a conservative approximation of all executions.

**In example**
- Solution 2 includes $x$ in live set $11$, which is not used later.
- However, analysis is conservative.

---

Safety and Precision

- **Safety:** dataflow equations guarantee a safe solution to the analysis problem.
- **Precision:** a solution to an analysis problem is more precise if it is less conservative.

**Live variables analysis problem:**
- Solution is more precise if the sets of live variables are smaller.
- Solution which reports that all variables are live at each point is safe, but is the least precise solution.

- **In the lattice framework:** $S1$ is less precise than $S2$ if the result in $S1$ at each program point is less than the corresponding result in $S2$ at the same point.

- Use notation $S1 \sqsubseteq S2$ if solution $S1$ is less precise than $S2$.

---

Maximal Fixed Point Solution

- **Property:** among all the solutions to the system of dataflow equations, the iterative solution is the most precise.
- **Intuition:**
  - We start with the top element at each program point (i.e., most precise information).
  - Then refine the information at each iteration to satisfy the dataflow equations.
  - Final result will be the closest to the top.

- Iterative solution for dataflow equations is called Maximal Fixed Point solution (MFP).
- For any solution $FP$ of the dataflow equations: $FP \subseteq MFP$.

---

Meet Over Paths Solution

- Is MFP the best solution to the analysis problem?

- **Another approach:** consider a lattice framework, but use a different way to compute the solution.
  - Let $G$ be the control flow graph with start block $B_0$.
  - For each path $P = [B_0, B_1, ..., B_n]$ from entry to block $B_n$:
    
    $in(p) = F_{B_n+1} \ldots (F_{B_1}(F_{B_0}(in(X))))$

  - Compute solution as
    
    $in(B_n) = \cap \{ in(p) \ | \ all \ paths \ p_i \ from \ B_0 \ to \ B_n \}$

- This solution is the **Meet Over Paths solution (MOP)**.

---

MFP versus MOP

- **Precision:** can prove that MOP solution is always more precise than MFP.
  
  $MFP \subseteq MOP$

- **Why not use MOP?**
  
  MOP is intractable in practice.

  1. Exponential number of paths: for a program consisting of a sequence of $N$ if statements, there will $2^N$ paths in the control flow graph.
  2. Infinite number of paths: for loops in the CFG.
Importance of Distributivity

- **Property:** if transfer functions are distributive, then the solution to the dataflow equations is identical to the meet-over-paths solution
  
  \[ \text{MFP} = \text{MOP} \]

- For distributive transfer functions, can compute the intractable MOP solution using the iterative fixed-point algorithm

Better Than MOP?

- Is MOP the best solution to the analysis problem?
- MOP computes solution for all path in the CFG
- There may be paths which will never occur in any execution
- So MOP is conservative
- **IDEAL** = solution which takes into account only paths which occur in some execution
- This is the best solution
- ... but it is undecidable

Summary

- **Dataflow analysis**
  - sets up system of equations
  - iteratively computes MFP
  - Terminates because transfer functions are monotonic and lattice has finite height
- Other possible solutions: FP, MOP, IDEAL
- All are safe solutions, but some are more precise:
  
  \[ \text{FP} \subseteq \text{MFP} \subseteq \text{MOP} \subseteq \text{IDEAL} \]
- **MFP = MOP if distributive transfer functions**
- **MOP and IDEAL are intractable**
- **Compilers use dataflow analysis and MFP**