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 \( T \), 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

Transfer Functions

- Let \( L = \text{dataflow information lattice} \)

- 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: \( \text{in}[I] = F_i(\text{out}[I]) \)

- 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: \( \text{in}[B] = F_B(\text{out}[B]) \)

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 \( F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y) \)
  - 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. \( F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y) \), 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 \( F(x) \sqcap F(y) \)
  - 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: \( \text{in}[B] = \sqcap \{ \text{out}[B'] | B' \in \text{pred}(B) \} \)
    Backward analysis: \( \text{out}[B] = \sqcup \{ \text{in}[B'] | B' \in \text{succ}(B) \} \)

- 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 $L \times L$:
  $$x_1 \sqcap y_1 \text{ and } x_2 \sqcap y_2 \implies (x_1 \sqcap x_2) \sqsubseteq (y_1 \sqcap y_2)$$

- **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 $B_0$
- **Lattice $(L, \sqsubseteq)$** represents information about program
  - Meet operator $\sqcap$, top element $\top$
- **Monotonic transfer functions**
  - Transfer function $F_I : L \rightarrow L$ 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 $B_0$ is $X_0$

- **Require the** $\text{out}(B) = F_B(\text{in}(B))$, for all $B$

- **solution to**
  - $\text{in}(B) = \bigcap \{ \text{out}(B') \mid B' \text{ succ}(B) \}$, for all $B$
  - **satisfy:** $\text{out}(B_0) = X_0$

---

Backward Dataflow Analysis

- **Control flow graph $G$** with exit node $B_n$
- **Lattice $(L, \sqsubseteq)$** represents information about program
  - Meet operator $\sqcap$, top element $\top$
- **Monotonic transfer functions**
  - Transfer function $F_I : L \rightarrow L$ 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 $B_n$ is $X_n$

- **Require the** $\text{in}(B) = F_B(\text{out}(B))$, for all $B$

- **solution to**
  - $\text{out}(B) = \bigcap \{ \text{in}(B') \mid B' \text{ prec}(B) \}$, for all $B$
  - **satisfy:** $\text{in}(B_n) = X_n$

---

Dataflow Equations

- The constraints are called **dataflow equations**:
  - $\text{out}(B) = F_B(\text{in}(B))$, for all $B$
  - $\text{in}(B) = \bigcap \{ \text{out}(B') \mid B' \text{ prec}(B) \}$, for all $B$
  - $\text{in}(B_0) = X_0$

- **Solve equations:** use an iterative algorithm
  - Initialize $\text{in}(B_0) = X_0$
  - Initialize everything else to $\top$
  - Repeatedly apply rules
  - Stop when reach a fixed point

---

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 of list of blocks to evaluate
  - Initialize list to the set of all basic blocks
  - If out[B] changes after evaluating out[B] = $F_B(\text{in}[B])$, then add all successors of B to the list

---

Algorithm

$\text{in}(B_0) = X_0$

$\text{out}(B) = \top$, for all $B$

Repeat

For each basic block $B = B_n$

$\text{in}(B) = \bigcap \{ \text{out}(B') \mid B' \text{ prec}(B) \}$

For each basic block $B$

$\text{out}(B) = F_B(\text{in}(B))$

Until no change
Worklist Algorithm

\[ \text{in}(B) = X_0 \]
\[ \text{out}(B) = T, \text{ for all } B \]
\[ \text{worklist} = \text{set of all basic blocks } B \]

Repeat
- Remove a node \( B \) from the worklist
- \( \text{in}(B) = \cap \{ \text{out}(B') \mid B' \subseteq \text{pred}(B) \} \)
- \( \text{out}(B) = F_0(\text{in}(B)) \)
  - if \( \text{out}(B) \) has changed, then
    - \( \text{worklist} = \text{worklist} \cup \text{succ}(B) \)

Until \( \text{worklist} = \emptyset \)

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
    \[ \text{in}(B) = \cap \{ \text{out}(B') \mid B' \subseteq \text{pred}(B) \} \]
    \[ \text{out}(B) = F_0(\text{in}(B)) \]
  - 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:
  \[ \text{in}_{k+1}(B) \subseteq \text{in}_k(B) \text{ and } \text{out}_{k+1}(B) \subseteq \text{out}_k(B) \]
  where \( \text{in}_k(B) \) is info before \( B \) at iteration \( k \) and \( \text{out}_k(B) \) 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: \( \text{in}_0(B) \supseteq \text{in}_k(B) \supseteq \text{in}_0(B) \) ...

Chains in Lattices

- A chain in a lattice \( L \) is a totally ordered subset \( S \) of \( L \):
  \[ x \subseteq y \text{ or } y \subseteq x \text{ for any } x, y \in S \]
- In other words:
  Elements in a totally ordered subset \( S \) can be indexed to form an ascending sequence:
  \[ x_1 \sqsubseteq x_2 \sqsubseteq x_3 \sqsubseteq ... \]
  or they can be indexed to form a descending sequence:
  \[ x_1 \sqsupseteq x_2 \sqsupseteq x_3 \sqsupseteq ... \]
- 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 \( B \):
  \( \text{in}_i(B), \text{in}_{i+1}(B), ... \)
  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 \( \text{in}_k(B) = \text{in}_{k+1}(B) \), for all \( k \) and all \( B \)
- If \( \text{in}_k(B) = \text{in}_{k+1}(B) \) then also \( \text{out}_k(B) = \text{out}_{k+1}(B) \)
- 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

  \[
  \begin{align*}
  y &= 1 \\
  x &= y \\
  \end{align*}
  \]

  \[
  \begin{align*}
  11 &= 12-(y) \\
  12 &= 13 \cup 13 \\
  13 &= (14-\{x\}) \cup \{y\} \\
  14 &= \{x\} \\
  \end{align*}
  \]

  Solution 1: \( 11=\{x\}, 12=\{y\}, 13=\{y\}, 14=\{x\} \)
  Solution 2: \( 11=\{x\}, 12=\{x\}, 13=\{y\}, 14=\{x\} \)
### 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
- **Example**:
  - Solution 2 includes x in live set I1, 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 ⊑ 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 ⊑ 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 B0
  - For each path π ∈ (B0, B1, ..., Bn) from entry to block Bn:
    \[ in(\pi) = F_{\text{top}}(B_n) \]
  - Compute solution as
    \[ in(B_n) = \bigcap \{ in(\pi) \mid \text{all paths } \pi \text{ from } B_0 \text{ to } B_n \} \]
- This solution is the **Meet Over Paths solution (MOP)**

### MFP versus MOP

- **Precision**: can prove that MOP solution is always more precise than MFP
  \[ MFP \not\subseteq MOP \]
- **Why not use MOP?**
  - MOP is intractable in practice
  1. Exponential number of paths: for a program consisting of a sequence of \textit{N} if statement, 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
  \[ MFP = MOP \]
- 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:
  - FP ⊆ MFP ⊆ MOP ⊆ IDEAL
- MFP = MOP if distributive transfer functions
- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP