Lattices

- **Lattice:**
  - Set augmented with a partial order relation \(\sqsubseteq\)
  - Each subset has a LUB and a GLB
  - Can define: meet \(\cap\), 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 \(L\) = 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 \(F(x \sqcap y) \sqsubseteq F(x) \sqcap F(y)\)
    - Use \(x \sqcap y \sqsubseteq x, x \sqcap y \sqsubseteq y\), 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 \(F(x) \sqsubseteq F(x) \sqcap F(y)\)
    - 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:** \(\text{out}[B] = \sqcap \{\text{in}[B'] | B' \in \text{succ}(B)\}\)

- 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 \( L \times L \):
  \[
  x_1 \sqcap y_1 \sqsubseteq x_2 \sqsubseteq y_2 \quad \text{implies} \quad (x_1 \sqcap x_2) \sqsubseteq (y_1 \sqcap y_2)
  \]

- **Proof:**
  - Any lower bound of \( \langle x_1, x_2 \rangle \) is also a lower bound of \( \langle y_1, y_2 \rangle \), because \( x_1 \sqsubseteq y_1 \) and \( x_2 \sqsubseteq y_2 \)
  - \( x_1 \sqcap x_2 \) is a lower bound of \( \langle x_1, x_2 \rangle \)
  - So \( x_1 \sqcap x_2 \) is a lower bound of \( \langle y_1, y_2 \rangle \)
  - But \( y_1 \sqcap y_2 \) is the greatest lower bound of \( \langle y_1, y_2 \rangle \)
  - 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** \( (L, \sqsubseteq) \) 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 \( B_s \) is \( X_0 \)
- Require the \( \text{out}[B] = F_B(\text{in}[B]) \), for all \( B \)
  - Solve equations: use an iterative algorithm
    - Initialize \( \text{in}[B_s] = X_0 \)
    - Initialize everything else to \( \top \)
    - Repeat:
      - For each basic block \( B = B_s \)
        - \( \text{in}[B] = \sqcap \langle \text{out}[B'] | B' \sqsubseteq \text{pred}(B) \rangle \)
        - For each basic block \( B \)
          - \( \text{out}[B] = F_B(\text{in}[B]) \)
    - Stop when reach a fixed point

Backward Dataflow Analysis

- **Control flow graph** \( G \) with exit node \( B_e \)
- **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 \( B_e \) is \( X_0 \)
- Require the \( \text{in}[B] = F_B(\text{out}[B]) \), for all \( B \)
  - Solve equations: use an iterative algorithm
    - Initialize \( \text{out}[B_s] = X_0 \)
    - Initialize everything else to \( \top \)
    - Repeat:
      - For each basic block \( B = B_s \)
        - \( \text{out}[B] = \sqcup \langle \text{in}[B'] | B' \sqsubseteq \text{succ}(B) \rangle \), for all \( B \)
        - For each basic block \( B \)
          - \( \text{in}[B] = F_B(\text{out}[B]) \)
    - Stop when reach a fixed point

Dataflow Equations

- The constraints are called **dataflow equations**:
  - \( \text{out}[B] = F_B(\text{in}[B]) \), for all \( B \)
  - \( \text{in}[B] = \sqcap \langle \text{out}[B'] | B' \sqsubseteq \text{pred}(B) \rangle \), for all \( B \)
  - \( \text{out}[B_s] = X_0 \)

Algorithm

- **in**[**B_s**] = \( X_0 \)
  - **out**[**B**] = \( \top \), for all \( B \)

Repeat

For each basic block \( B = B_s \)
  - \( \text{in}[B] = \sqcap \langle \text{out}[B'] | B' \sqsubseteq \text{pred}(B) \rangle \)
  - For each basic block \( B \)
    - \( \text{out}[B] = F_B(\text{in}[B]) \)

Until no change

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 \( \text{out}[B] \) changes after evaluating \( \text{out}[B] = F_B(\text{in}[B]) \), then add all successors of \( B \) to the list
Worklist Algorithm

\textbf{in}[B]_0 = X_0
\textbf{out}[B] = T$, for all $B$
worklist = set of all basic blocks $B$
\Repeat
\begin{itemize}
  \item Remove a node $B$ from the worklist
  \item $\textbf{in}[B] = \cap \textbf{out}[B'] | B' \in \text{pred}(B)$
  \item $\textbf{out}[B] = \mathcal{F}_{in}(B)$
  \item if out$(B)$ has changed, then
  \item worklist = worklist $\cup$ succ$(B)$
\EndRepeat
\Until worklist = $\emptyset$

Correctness

\begin{itemize}
  \item Initial algorithm is correct
    \begin{itemize}
      \item If dataflow information does not change in the last iteration, then it satisfies the equations
    \end{itemize}
  \item Worklist algorithm is correct
    \begin{itemize}
      \item Maintains the invariant that
        \begin{itemize}
          \item $\textbf{in}[B] = \cap \textbf{out}[B'] | B' \in \text{pred}(B)$
          \item $\textbf{out}[B] = \mathcal{F}_{in}(B)$
        \end{itemize}
      \item for all the blocks $B$ not in the worklist
      \item At the end, worklist is empty
    \end{itemize}
\end{itemize}

Termination

\begin{itemize}
  \item Do these algorithms terminate?
  \item Key observation: at each iteration, information decreases in the lattice:
    \begin{itemize}
      \item $\textbf{in}_k[B] \subseteq \textbf{in}[B]$ and $\textbf{out}_k[B] \subseteq \textbf{out}[B]$
    \end{itemize}
    where $\textbf{in}[B]$ is info before $B$ at iteration $k$ and $\textbf{out}[B]$ is info after $B$ at iteration $k$
  \item Proof by induction:
    \begin{itemize}
      \item Induction basis: true, because we start with top element, which is greater than everything
      \item Induction step: use monotonicity of transfer functions and meet operation
    \end{itemize}
  \item Information forms a chain: $\textbf{in}_1[B] \subseteq \textbf{in}_2[B] \subseteq \textbf{in}_3[B] ...$
\end{itemize}

Chains in Lattices

\begin{itemize}
  \item A chain in a lattice $L$ is a totally ordered subset $S$ of $L$:
    \begin{itemize}
      \item $x \subseteq y$ or $y \subseteq x$ for any $x, y \subseteq S$
    \end{itemize}
  \item In other words:
    \begin{itemize}
      \item Elements in a totally ordered subset $S$ can be indexed to form an ascending sequence:
        \begin{itemize}
          \item $x_1 \subseteq x_2 \subseteq x_3 \subseteq ...$
        \end{itemize}
      \item or they can be indexed to form a descending sequence:
        \begin{itemize}
          \item $x_1 \supseteq x_2 \supseteq x_3 \supseteq ...$
        \end{itemize}
    \end{itemize}
  \item Height of a lattice = size of its largest chain
  \item Lattice with finite height: only has finite chains
\end{itemize}

Multiple Solutions

\begin{itemize}
  \item The iterative algorithm computes a solution of the system of dataflow equations
  \item ... is the solution unique?
  \item No, dataflow equations may have multiple solutions!
  \item Example: live variables
    \begin{itemize}
      \item $y = 1$
      \item $x = y$
    \end{itemize}
    \begin{itemize}
      \item Equations:
        \begin{itemize}
          \item $I_1 = I_2 - \{y\}$
          \item $I_3 = (I_4 - \{x\}) \cup \{y\}$
          \item $I_2 = I_1 \cup I_3$
          \item $I_4 = \{x\}$
        \end{itemize}
    \end{itemize}
    \begin{itemize}
      \item Solution 1: $I_1 = \{y\}, I_2 = \{y\}, I_3 = \{y\}, I_4 = \{x\}$
      \item Solution 2: $I_1 = \{x\}, I_2 = \{x\}, I_3 = \{y\}, I_4 = \{x\}$
    \end{itemize}
\end{itemize}
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 I_1, 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: S_1 is less precise than S_2 if the result in S_1 at each program point is less than the corresponding result in S_2 at the same point
  - Use notation S_1 \preceq S_2 if solution S_1 is less precise than S_2

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 \preceq 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:
    \text{in}[B_n] = \text{in}[B_0] \cap \{ \text{in}[p] | all paths p from B_0 to B_n \}
  - Compute solution as
    \text{in}[B_0] = \bigcap \{ \text{in}[p] | all paths p 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
  \text{MFP} \preceq \text{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 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
  \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
- \( \text{IDEAL} = \) solution which takes into account only paths which occur in some execution
- This is the best solution
- … but it is undecidable

\[
\begin{align*}
\text{if (c)} & \\
\quad x &= 1 & x &= 2 \\
\text{if (c)} & \\
\quad y &= y + 2 & y &= x + 1
\end{align*}
\]

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