Lecture 21: More About Dataflow Analysis

13 Mar 02

Transfer Functions

- Let \( L = \text{dataflow information lattice} \)
  - Transfer function \( F_i : L \to L \) 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 \( F_B : L \to L \) 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 \( F : L \to L \) is monotonic if \( x \leq y \) implies \( F(x) \leq F(y) \)
  - Distributivity: function \( F : L \to L \) is distributive if \( F(x \cap y) = F(x) \cap F(y) \)
  - Property: \( F \) is monotonic iff \( F(x \cap y) \leq F(x) \cap F(y) \)
    - any distributive function is monotonic

Proof of Property

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

Proof for "1 implies 2."
- Need to prove that \( F(x \cap y) \leq F(x) \cap F(y) \)
- Use property 2 to get \( F(x \cap y) \leq F(x) \cap F(y) \)
- Hence \( F(x \cap y) \leq F(x) \cap F(y) \)

Proof of "2 implies 1."
- Let \( x, y \) such that \( x \leq y \)
  - Then \( x \cap y = x \), so \( F(x \cap y) = F(x) \)
  - Use property 2 to get \( F(x \cap y) \leq F(x) \cap F(y) \)
  - Hence \( F(x \cap y) \leq F(x) \cap F(y) \)

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: \( \text{in}[B] = \bigcap \{ \text{out}[B] \mid B < \text{pred}(B) \} \)
    Backward analysis: \( \text{out}[B] = \bigcap \{ \text{in}[B] \mid B < \text{succ}(B) \} \)
  - 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 $L \times L$:
  
  $$x_1 \sqsubseteq y_1 \text{ and } x_2 \sqsubseteq y_2 \Rightarrow (x_1 \sqcap x_2) \sqsubseteq (y_1 \sqcap y_2)$$

- **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 operation $\sqcap$, top element $T$
- 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 $B_i$ is $X_0$

Backward Dataflow Analysis

- Control flow graph $G$ with exit node $B_f$
- Lattice $(L, \sqsubseteq)$ represents information about program
  
  - Meet operator $\sqcap$, top element $T$
- 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 $B_i$ is $X_0$

Dataflow Equations

- The constraints are called dataflow equations:
  
  $$\text{out}(B) = F_B(\text{in}(B)), \text{ for all } B$$
  
  $$\text{in}(B) = \Gamma(\{\text{out}(B') | B' \sqsubseteq \text{pred}(B)\}), \text{ for all } B$$
  
  $$\text{in}[B_f] = X_0$$

- Solve equations: use an iterative algorithm
  
  - Initialize $\text{in}[B_f] = X_0$
  - Initialize everything else to $T$
  - Repeatedly apply rules
  - Stop when reach a fixed point

Algorithm

$$\text{in}[B_f] = X_0$$

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

Repeat

- For each basic block $B \neq B_f$
  
  $$\text{in}[B] = \Gamma(\{\text{out}(B') | B' \sqsubseteq \text{pred}(B)\})$$
  
  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 out[B] changes after evaluating out[B] = $F_B(\text{in}[B])$, then add all successors of $B$ to the list
### Worklist Algorithm

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

**Repeat**
- Remove a node \( B \) from the worklist
  - \( \text{in}[B] = \text{dom}(\text{out}[B]) \)
  - \( \text{out}[B] = \text{dom}(\text{in}[B]) \)
- If \( \text{out}[B] \) has changed, then
  - worklist = worklist \( \cup \) Succ(\( B \))

**Until** 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
    - \( \text{in}[B] = \text{dom}(\text{out}[B]) \)
    - \( \text{out}[B] = \text{dom}(\text{in}[B]) \)
  - 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
  - \( \text{in}_{\text{old}}[B] = \text{in}[B] \) and \( \text{out}_{\text{old}}[B] = \text{out}[B] \)
  - 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: \( \text{in}_1[B] \supseteq \text{in}_2[B] \supseteq \text{in}_3[B] \ldots \)

---

### Chains in Lattices

- A chain in a lattice \( L \) is a totally ordered subset \( S \) of \( L \):
  - \( x \sqsubseteq y \) or \( y \sqsubseteq x \) for any \( x, y \in S \)

- In other words:
  - Elements in a totally ordered subset \( S \) can be indexed to form an ascending sequence:
    - \( x_1 \sqsubseteq x_2 \sqsubseteq \ldots \)
  - or they can be indexed to form a descending sequence:
    - \( x_1 \supseteq x_2 \supseteq \ldots \)

- Height of a lattice = size of its largest chain

- Lattice with finite height: only has finite chains

---

### 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:
    - \( I_1 = I_2 = y = y \)
    - \( I_2 = I_1 \cup I_3 \)
    - \( I_4 = \{ y \} \)

- Solution 1: \( I_1 = \{ y \}, I_2 = \{ y \}, I_3 = \{ y \}, I_4 = \{ y \} \)

- Solution 2: \( I_1 = \{ y \}, I_2 = \{ x, y \}, I_3 = \{ y \}, I_4 = \{ \} \)
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 S1, which is not used later
  - However, analysis is conservative

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 \pi \in [B_0, B_1, ..., B_n] from entry to block B
    \in [B_i] = T_{\text{exit}} (\pi_i (\text{top}(B_0)))
  - Compute solution as
    \in [B_i] = \cap \{ \in [B_i] \} all paths \pi 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
  \text{MFP} \subseteq \text{MOP}
- Why not use MOP?
  - MOP is intractable in practice
    1. Exponential number of paths: for a program consisting of a sequence of if statement, there will be 2^N paths in the control flow graph
    2. Infinite number of paths: for loops in the C/G

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:
  - FP ⊆ MFP ⊆ MOP ⊆ IDEAL
- MFP = MOP if distributive transfer functions
- MOP and IDEAL are intractable
- Compilers use dataflow analysis and MFP