Lecture 3
Foundation of Data Flow Analysis

I  Semi-lattice (set of values, meet operator)
II  Transfer functions
III  Correctness, precision and convergence
IV  Meaning of data flow solution

Reading: Chapter 9.3
Recapping Lecture 2: Data Flow Framework

<table>
<thead>
<tr>
<th></th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>Sets of definitions</td>
<td>Sets of variables</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td><strong>forward:</strong></td>
<td><strong>backward:</strong></td>
</tr>
<tr>
<td></td>
<td><code>out[b] = f_b(in[b])</code></td>
<td><code>in[b] = f_b(out[b])</code></td>
</tr>
<tr>
<td></td>
<td><code>in[b] = \land out[pred(b)]</code></td>
<td><code>out[b] = \land in[succ(b)]</code></td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
<td><code>f_b(x) = Gen_b \cup (x - Kill_b)</code></td>
<td><code>f_b(x) = Use_b \cup (x - Def_b)</code></td>
</tr>
<tr>
<td><strong>Meet Operation (\land)</strong></td>
<td>\cup</td>
<td>\cup</td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
<td><code>out[entry] = \emptyset</code></td>
<td><code>in[exit] = \emptyset</code></td>
</tr>
<tr>
<td><strong>Initial interior points</strong></td>
<td><code>out[b] = \emptyset</code></td>
<td><code>in[b] = \emptyset</code></td>
</tr>
</tbody>
</table>
Thought Problem 1. “Must-Reach” Definitions

• A definition D (a = b+c) **must** reach point P iff
  – D appears at least once along all paths leading to P
  – a is not redefined along any path after last appearance of D and before P

• How do we formulate the data flow algorithm for this problem?

<table>
<thead>
<tr>
<th></th>
<th>MAY Reach</th>
<th>MUST Reach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Sets of definitions</td>
<td></td>
</tr>
<tr>
<td>Direction</td>
<td>forward:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>out[b] = f_b(in[b])</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in[b] = ( \bigwedge ) out[pred(b)]</td>
<td></td>
</tr>
<tr>
<td>Transfer function</td>
<td>( f_b(x) = Gen_b \cup (x - Kill_b) )</td>
<td></td>
</tr>
<tr>
<td>Meet Operation (( \bigwedge ))</td>
<td>( \cup )</td>
<td></td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[entry] = ( \emptyset )</td>
<td></td>
</tr>
<tr>
<td>Initial interior pts</td>
<td>out[b] = ( \emptyset )</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1: Example

• Quiz 1: Does the precise answer satisfy the equations?
• Quiz 2: Are there alternative solutions that satisfy the equations?
• Quiz 3: Are the alternative solutions incorrect?
• Quiz 4: How do you get the precise answer?
• Quiz 5: What is a wrong answer?
Problem 2: A legal solution to **(May)** Reaching Def?

- Quiz 1: Is this a solution to the equation?
- Quiz 2: Will the worklist algorithm generate this answer?
- Quiz 3: Is this solution safe or incorrect?
Summary

• There may be multiple (fixed-point) solutions to a set of equations
  – Non-solutions are incorrect
  – All solutions to the equations are considered safe, but maybe imprecise
  – Initialization of the interior points gives different solutions
• There is a direction to the solutions
• Answer is precise only when the initialization of interior points is
  – U for must-reach
  – {} for may-reach
Problem 3. What are the algorithm properties?

- Correctness

- Precision: how good is the answer?

- Convergence: will the analysis terminate?

- Speed: how long does it take?
I. Purpose of a Framework

• **Purpose 1**
  – Prove properties of entire family of problems once and for all
    • Will the program converge?
    • What does the solution to the set of equations mean?

• **Purpose 2:**
  – Aid in software engineering: re-use code
The Data-Flow Framework

- **Data-flow problems** \((F, V, \wedge)\) **are defined by**
  - A semi-lattice
    - domain of values \(V\)
    - meet operator \(\wedge: V \times V \rightarrow V\)
  - A family of transfer functions \(F: V \rightarrow V\)
Lecture Outline

I  Semi-lattice (set of values, meet operator)
II  Transfer functions
III  Correctness, precision and convergence
IV  Meaning of data flow solution
Semi-lattice: Structure of the Domain of Values

• A semi-lattice $S = \langle$ a set of values $\lor$, a meet operator $\land$ $\rangle$

• Properties of the meet operator
  – idempotent: $x \land x = x$
  – commutative: $x \land y = y \land x$
  – associative: $x \land (y \land z) = (x \land y) \land z$

• Examples of meet operators?
• Non-examples?
Example of a Semi-Lattice Diagram

• \((V, \wedge) : V = \{x \mid \text{such that } x \subseteq \{d_1,d_2,d_3\}\}, \wedge = U\)

\[
\begin{array}{c}
\{\}
\end{array}
\begin{array}{ccc}
\{d_1\} & \{d_2\} & \{d_3\} \\
\{d_1,d_2\} & \{d_1,d_3\} & \{d_2,d_3\} \\
\{d_1,d_2,d_3\} & & \\
\end{array}
\begin{array}{c}
(T)
\end{array}
\]

• \(x \wedge y = \text{first common descendant of } x \& y\)  

• A meet semi-lattice is bounded if there exists a top element \(T\), such that \(x \wedge T = x\) for all \(x\).

• A bottom element \(\bot\) exists, if \(x \wedge \bot = \bot\) for all \(x\).
A Meet Operator Defines a Partial Order

- **Partial order of a meet semi-lattice**
  \[ \leq : x \leq y \text{ if and only if } x \land y = x \]

- **Meet operator**: \( U \)

- **Properties of meet operator guarantee that \( \leq \) is a partial order**
  - Reflexive: \( x \leq x \)
  - Antisymmetric: if \( x \leq y \) and \( y \leq x \) then \( x = y \)
  - Transitive: if \( x \leq y \) and \( y \leq z \) then \( x \leq z \)
Another Example

• Semi-lattice
  – \( V = \{ x \mid \text{such that } x \subseteq \{d_1, d_2, d_3\} \} \)
  – \( \wedge = \cap \)

\[
\begin{array}{c}
\{d_1, d_2, d_3\} \\
\{d_1, d_2\} & \{d_1, d_3\} & \{d_2, d_3\} \\
\{d_1\} & \{d_2\} & \{d_3\} \\
\{\}\ \\
\end{array}
\]

– \( \leq \) is
Meet Semi-Lattices vs Partially Ordered Sets

- A **meet-semilattice** is a partially ordered set which has a **meet** (or **greatest lower bound**) for any nonempty finite subset.

- Greatest lower bound: \( x \wedge y = \text{First common descendant of } x \& y \)
- Largest: top element \( T \), if \( x \wedge T = x \) for all \( x \).
- Smallest: bottom element \( \perp \), if \( x \wedge \perp = \perp \) for all \( x \).
Drawing a Semi-Lattice Diagram

- \((x < y) \equiv (x \leq y) \land (x \neq y)\)

- **A semi-lattice diagram:**
  - Set of nodes: set of values
  - Set of edges \(\{(y, x) : x < y \land \neg \exists z \text{ s.t. } (x < z) \land (z < y)\}\)
Summary

Three ways to define a semi-lattice:

• Set of values + meet operator
  – idempotent: \( x \land x = x \)
  – commutative: \( x \land y = y \land x \)
  – associative: \( x \land (y \land z) = (x \land y) \land z \)

• Set of values
  + partial order with a greatest lower bound for any nonempty subset
    – Reflexive: \( x \leq x \)
    – Antisymmetric: if \( x \leq y \) and \( y \leq x \) then \( x = y \)
    – Transitive: if \( x \leq y \) and \( y \leq z \) then \( x \leq z \)

• A semi-lattice diagram
One Element at a Time

- A semi-lattice for data flow problems can get quite large: $2^n$ elements for $n$ var/definition
- A useful technique:
  - define semi-lattice for 1 element
  - product of semi-lattices for all elements
  - $<x_1, x_2> \leq <y_1, y_2>$ iff $x_1 \leq y_1$ and $x_2 \leq y_2$
- **Example**: Union of definitions
  - For each element
    
    \[
    \begin{array}{ccc}
    \text{def1} & \text{def2} & \text{def1} \times \text{def2} \\
    \{\} & \{\} & \{\},{} \\
    \{d_1\} & \{d_2\} & \{d_1\},{} \\
    \{\} & \{d_2\} & \{d_1\},\{d_2\} \\
    \{d_1\},\{d_2\} & \{\},\{d_2\} & \\
    \end{array}
    \]
Descending Chain

• Definition
  – The **height** of a lattice is the largest number of > relations that will fit in a descending chain.
  \[ x_0 > x_1 > \ldots \]

• Height of values in reaching definitions?

• Important property: finite descending chains
II. Transfer Functions

- A family of transfer functions $F$
- Basic Properties $f : V \rightarrow V$
  - Has an identity function
    - $\exists f$ such that $f(x) = x$, for all $x$.
  - Closed under composition
    - if $f_1, f_2 \in F$, $f_1 \cdot f_2 \in F$
Monotonicity: 2 Equivalent Definitions

• A framework \((F, V, \land)\) is monotone iff
  – \(x \leq y\) implies \(f(x) \leq f(y)\)

• Equivalently,
  a framework \((F, V, \land)\) is monotone iff
  – \(f(x \land y) \leq f(x) \land f(y)\),
  – meet inputs, then apply \(f\)
    \leq
  apply \(f\) individually to inputs, then meet results
Example

• Reaching definitions: \( f(x) = \text{Gen } U \, (x - \text{Kill}), \ \land = U \)
  
  – Definition 1:
    
    • Let \( x_1 \leq x_2 \),
    
    \[ f(x_1): \text{Gen } U \, (x_1 - \text{Kill}) \]
    \[ f(x_2): \text{Gen } U \, (x_2 - \text{Kill}) \]

  – Definition 2:
    
    • \( f(x_1 \land x_2) = (\text{Gen } U \, ((x_1 \lor x_2) - \text{Kill})) \)
    
    \[ f(x_1) \land f(x_2) = (\text{Gen } U \, (x_1 - \text{Kill}) \lor \text{Gen } U \, (x_2 - \text{Kill})) \]
Distributivity

- A framework \((F, V, \land)\) is distributive if and only if
  \[ f(x \land y) = f(x) \land f(y), \]

  meet input, then apply \(f\) is equal to
  apply the transfer function individually then merge result
Important Note

- Monotone framework **does not mean** that $f(x) \leq x$
  - e.g. Reaching definition for two definitions in program
  - suppose: $f: \text{Gen} = \{d_1\} ; \text{Kill} = \{d_2\}$
  - Quiz: What are the inputs and outputs of $f$?
III. Properties of Iterative Algorithm

• Given
  A monotone data flow framework
  With finite descending chains

• The iterative algorithm where all interior points are initialized to $T$
  – Converges
  – To the Maximum Fixed Point (MFP) solution of equations
Key Concept

• The answer is a set of values for all basic block boundaries:
  \{ \text{in}[b], \text{out}[b] \mid b \text{ in the program} \}

• We need to prove the invariant:
  – Values assigned to the same \text{in}[b] or \text{out}[b]
    cannot increase in each iteration of the algorithm

• The algorithm converges
  if the semilattice has finite descending chains

• Given an initialization of $T$,
  the answer is the MFP (Maximum Fixed Point),
  because any larger value is not a solution.
Sketch of Inductive Proof

The boundary is set to the correct answer and never changed

For each IN/OUT of an interior program point:

- Invariant: new value ≤ old value in any step
- Initialize interior points with T (largest value)
- Proof by induction
  - Base case:
    1st transfer function or meet operator: new value ≤ old value (T)
  - Meet operation:
    - Assume new inputs ≤ old inputs, new output ≤ old output
  - Transfer function (in a monotone framework):
    - Assume new inputs ≤ old inputs, new output ≤ old output
IV. What Does the Solution Mean?

• IDEAL data flow solution
  – Let $f_1, ..., f_m : \in F$, $f_i$ is the transfer function for node $i$

    $f_p = f_{n_k} \ldots f_{n_1}$, $p$ is a path through nodes $n_1, ..., n_k$

    $f_p = \text{identify function, if } p \text{ is an empty path}$

  – For each node $n$: $\bigwedge f_{p_i}$ (boundary value), for all possibly executed paths $p_i$ reaching $n$

  – Example

- Determining all possibly executed paths is undecidable
Meet-Over-Paths MOP

• **Err in the conservative direction**

• **Meet-Over-Paths MOP**
  – Assume every edge is traversed
  – For each node n:
    - $\text{MOP}(n) = \bigwedge f_{p_i}$ (boundary value), for all paths $p_i$ reaching $n$

• **Compare MOP with IDEAL**
  – MOP includes more paths than IDEAL
  – $\text{MOP} = \text{IDEAL} \land \text{Result(Unexecuted-Paths)}$
  – $\text{MOP} \leq \text{IDEAL}$
  – MOP is a “smaller” solution, more conservative, safe

• **MOP $\leq$ IDEAL**
  – Goal: as close to MOP from below as possible
Solving Data Flow Equations

• What is the difference between MOP and MFP of data flow equations?

• Therefore
  – FP ≤ MFP ≤ MOP ≤ IDEAL
  – FP, MFP, MOP are safe
  – If framework is distributive, FP ≤ MFP = MOP ≤ IDEAL
Summary

• **A data flow framework**
  - Semi-lattice
    • set of values (top)
    • meet operator
    • finite descending chains?
  - Transfer functions
    • summarizes each basic block
    • boundary conditions

• **Properties of data flow framework:**
  - Monotone framework and finite descending chains
    \[ \Rightarrow \text{iterative algorithm converges} \]
    \[ \Rightarrow \text{finds maximum fixed point (MFP)} \]
    \[ \Rightarrow \text{FP} \leq \text{MFP} \leq \text{MOP} \leq \text{IDEAL} \]

  - Distributive framework
    \[ \Rightarrow \text{FP} \leq \text{MFP} = \text{MOP} \leq \text{IDEAL} \]