# 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>forward:</td>
<td>backward:</td>
</tr>
<tr>
<td></td>
<td>( \text{out}[b] = \text{f}_b(\text{in}[b]) )</td>
<td>( \text{in}[b] = \wedge \text{out}[	ext{pred}(b)] )</td>
</tr>
<tr>
<td></td>
<td>( \text{in}[b] = \wedge \text{out}[	ext{pred}(b)] )</td>
<td>( \text{out}[b] = \wedge \text{in}[	ext{succ}(b)] )</td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
<td>( \text{f}_b(x) = \text{Gen}_b \cup (x - \text{Kill}_b) )</td>
<td>( \text{f}_b(x) = \text{Use}_b \cup (x - \text{Def}_b) )</td>
</tr>
<tr>
<td><strong>Meet Operation ((\wedge))</strong></td>
<td>(\cup)</td>
<td>(\cup)</td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
<td>( \text{out}[\text{entry}] = \emptyset )</td>
<td>( \text{in}[\text{exit}] = \emptyset )</td>
</tr>
<tr>
<td><strong>Initial interior points</strong></td>
<td>( \text{out}[b] = \emptyset )</td>
<td>( \text{in}[b] = \emptyset )</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 on 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><strong>Domain</strong></td>
<td>Sets of definitions</td>
<td></td>
</tr>
<tr>
<td><strong>Direction</strong></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] = $\land$ out[pred(b)]</td>
<td></td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
<td>$f_b(x) = Gen_b \cup (x-Kill_b)$</td>
<td></td>
</tr>
<tr>
<td><strong>Meet Operation</strong></td>
<td>$\cup$</td>
<td></td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
<td>out[entry] = $\emptyset$</td>
<td></td>
</tr>
<tr>
<td><strong>Initial interior pts</strong></td>
<td>out[b] = $\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>
Problem 2: A legal solution to (May) Reaching Def?

- Will the worklist algorithm generate this answer?
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?
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
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** \((F, V, \land)\) are defined by
  - A semi-lattice
    - domain of values \(V\)
    - meet operator \(\land: V \times V \rightarrow V\)
  - A family of transfer functions \(F: V \rightarrow V\)
Semi-lattice: Structure of the Domain of Values

- A semi-lattice $S = \langle \text{a set of values } V, \text{ a meet operator } \wedge \rangle$

- Properties of the meet operator
  - idempotent: $x \wedge x = x$
  - commutative: $x \wedge y = y \wedge x$
  - associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$

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

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

• $x \wedge y =$ 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: \( \land \)

Partial order \( \leq : \)

- 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
  – $V = \{x \mid \text{such that } x \subseteq \{d_1, d_2, d_3\}\}$
  – $\wedge = \cap$

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

– $\leq$ 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.

![Diagram of meet-semilattice](image)

- **Greatest lower bound**: $x \land y = \text{First common descendant of } x \land y$
- **Largest**: top element $T$, if $x \land T = x$ for all $x$.
- **Smallest**: bottom element $\bot$, if $x \land \bot = \bot$ 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 \(\{(y, x): x < y \text{ and } \neg \exists z \text{ s.t. } (x < z) \land (z < y)\}\)
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
- Example: Union of definitions
  - For each element
    - $\langle x_1, x_2 \rangle \preceq \langle y_1, y_2 \rangle$ iff $x_1 \preceq y_1$ and $x_2 \preceq y_2$
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 $f: V \to V$
  - Has an identity function
    - $\exists f$ such that $f(x) = x$, for all $x$.
  - Closed under composition
    - if $f_1, f_2 \in F$, $f_1 \circ f_2 \in F$
Monotonicity: 2 Equivalent Definitions

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

• Equivalently,
  a framework \((F, V, \wedge)\) is monotone iff
  \(- f(x \wedge y) \leq f(x) \wedge f(y),\)
  \(- \text{meet inputs, then apply } f\)
  \(- \leq\)
  \(- \text{apply } f \text{ individually to inputs, then meet results}\)
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:
    
    • \( f(x_1 \land x_2) = (\text{Gen } U ((x_1 U x_2) - \text{Kill})) \)
      
      \( f(x_1) \land f(x_2) = (\text{Gen } U (x_1 - \text{Kill})) U (\text{Gen } U (x_2 - \text{Kill})) \)
Distributivity

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

  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: \( f: Gen = \{d_1\} ; \text{Kill} = \{d_2\} \)
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, because any larger
  value is not a solution.
Sketch of Inductive Proof

For each IN/OUT of an interior program point:

- Invariant: new value ≤ old value in any step
- Start with T (largest value)
- Proof by induction
  - 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 $f_1, \ldots, f_m : \mathbb{F}, f_i$ is the transfer function for node $i$
    
    $$f_p = f_{n_k} \circ \cdots \circ f_{n_1}, \text{p is a path through nodes } n_1, \ldots, n_k$$

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

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

- Example

  ![Diagram](image)

  - 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 $n$:
    - $MOP(n) = \land f_{p_i}$ (boundary value), for all paths $p_i$ reaching $n$

- **Compare MOP with IDEAL**
  - $MOP$ includes more paths than IDEAL
  - $MOP = IDEAL \land Result(\text{Unexecuted-Paths})$
  - $MOP \leq IDEAL$
  - $MOP$ 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
  – $FP \leq MFP \leq MOP \leq IDEAL$
  – FP, MFP, MOP are safe
  – If framework is distributive, $FP \leq MFP = MOP \leq IDEAL$
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

  ⇒ iterative algorithm converges
  ⇒ finds maximum fixed point (MFP)
  ⇒ $FP \leq MFP \leq MOP \leq IDEAL$

  – Distributive framework
    ⇒ $FP \leq MFP = MOP \leq IDEAL$