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** \((F, V, \wedge)\) are defined by
  - A semilattice
    - domain of values \((V)\)
    - meet operator \((\wedge)\)
  - A family of transfer functions \((F : V \rightarrow V)\)
Semi-lattice: Structure of the Domain of Values

- A semi-lattice $S = < V, \wedge >$

- 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

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

\[
\begin{array}{c}
\{\}
\end{array}
\quad
\begin{array}{c}
(\top)
\end{array}
\]

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

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

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

- \(x \wedge y = \text{first common descendant of } x \& y\)
- Define top element \(\top\), such that \(x \wedge \top = x\)
- Define bottom element \(\bot\), such that \(x \wedge \bot = \bot\)
- Semi-lattice diagram : picture of a partial order!
A Meet Operator Defines a Partial Order (vice versa)

• Definition of partial order $\leq$: $x \leq y$ if and only if $x \wedge y = x$

  $\begin{array}{c}
  \text{path} \\
  \downarrow \\
  x \\
  \equiv \\
  (x \wedge y = x) \\
  \equiv \\
  (x \leq y)
  \end{array}$

• 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 $x \leq z$

• $(x < y) \equiv (x \leq y) \wedge (x \neq y)$

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

• Example:
  • Meet operator: $\cup$ Partial order $\leq$:
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
  - 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
  - No cycles
  - \( \top \) is on top of everything
  - \( \bot \) is at the bottom of everything
Another Example

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

\[
\begin{align*}
\{d_1,d_2,d_3\} & \quad (\top) \\
\{d_1,d_2\} & \quad \{d_1,d_3\} & \quad \{d_2,d_3\} \\
\{d_1\} & \quad \{d_2\} & \quad \{d_3\} \\
\{\} & \quad (\bot)
\end{align*}
\]

• \( \leq \) is
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
    - def1
      - $\{\}$
      - $\{d_1\}$
    - def2
      - $\{\}$
      - $\{d_2\}$
    - def1 x def2
      - $\{\},\{\}$
      - $\{d_1\},\{\}$
      - $\{\},\{d_2\}$
      - $\{d_1\},\{d_2\}$

- $<x_1, x_2> \leq <y_1, y_2>$ iff $x_1 \leq y_1$ and $x_2 \leq y_2$
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 $f : V \rightarrow V$
  
  - 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**: \( f(x) = \text{Gen} \cup (x - \text{Kill}), \land = \cup \)

  - Definition 1:
    
    - Let \( x_1 \leq x_2 \),
    
    \[
    f(x_1): \text{Gen} \cup (x_1 - \text{Kill})
    \]
    
    \[
    f(x_2): \text{Gen} \cup (x_2 - \text{Kill})
    \]

  - Definition 2:
    
    - \( f(x_1 \land x_2) = (\text{Gen} \cup ((x_1 \cup x_2) - \text{Kill})) \)
    
    \[
    f(x_1) \land f(x_2) = (\text{Gen} \cup (x_1 - \text{Kill})) \cup (\text{Gen} \cup (x_2 - \text{Kill}))
    \]
Important Note

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

- Given:
  - ∧ and monotone data flow framework
  - Finite descending chain
  - \( \Rightarrow \) Converges

- Initialization of interior points to T
  - \( \Rightarrow \) Maximum Fixed Point (MFP) solution of equations
**Behavior of iterative algorithm (intuitive)**

For each IN/OUT of an interior program point:
- Its value cannot go up (new value ≤ old value) during algorithm
- Start with T (largest value)
- Proof by induction
  - Apply 1st transfer function / meet operator ≤ old value (T)
  - Inputs to “meet” change (get smaller)
    - since inputs get smaller, new output ≤ old output
  - Inputs to transfer functions change (get smaller)
    - monotonicity of transfer function:
      since input gets smaller, new output ≤ old output
- Algorithm iterates until equations are satisfied
- Values do not come down unless some constraints drive them down.
- Therefore, finds the largest solution that satisfies the equations
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 $i$
    \[
    f_p = f_{n_k} \circ ... \circ f_{n_1}, \quad p \text{ is a path through nodes } n_1, ..., n_k
    \]
  - $f_p$ = identify function, if $p$ is an empty path
  - For each node $n$: $\land f_{p_i}$ (boundary value), for all possibly executed paths $p_i$ reaching $n$
  - Example

\[
\begin{array}{c}
\text{if } \text{sqr}(y) \geq 0 \\
\text{false} & \text{true}
\end{array}
\]

\[
\begin{array}{c}
x = 0 \\
x = 0 \\
x = 1
\end{array}
\]

- 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 $n$:

  $$\text{MOP}(n) = \bigwedge f_{p_i} \text{ (boundary value), for all paths } p_i \text{ reaching } n$$

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

- Data Flow Solution $\leq \text{MOP} \leq \text{IDEAL}$
  - 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
    \[ \Rightarrow \text{FP} \leq \text{MFP} = \text{MOP} \leq \text{IDEAL} \]