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 semi-lattice
    - domain of values \(V\)
    - meet operator \(\wedge: V \times V \rightarrow V\)
  - A family of transfer functions \(F: V \rightarrow V\)
Semi-lattice: Structure of the Domain of Values

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

• 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

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

- \(x \wedge y = \text{first common descendant of } x \& y\)
- 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 \(x\).
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**: \( x \land y = \text{First common descendant of } x \land y \)
- **Largest**: top element \( T \), if \( x \land T = x \) for all \( x \).
- **Smallest**: bottom element \( \bot \), if \( x \land \bot = \bot \) for all \( x \).
A Meet Operator Defines a Partial Order

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

- **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 \( x \leq z \)
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
Another Example

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

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

- \( \preceq \) is
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

\[
\begin{align*}
def1 & \quad \text{def2} \\
\{\} & \quad \{\} \\
\{d_1\} & \quad \{d_2\}
\end{align*}
\]

\[
\begin{align*}
def1 \times \text{def2} \\
\{\},\{\} \\
\{d_1\},\{\} & \quad \{\},\{d_2\} \\
\{d_1\},\{d_2\} \\
\{d_1\},\{d_2\}
\end{align*}
\]

- $<x_1, x_2> \preceq <y_1, y_2>$ 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: $f(x) = \text{Gen } U (x - \text{Kill}), \land = U$

  - 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:
    - $f(x_1 \land x_2) = (\text{Gen } U ((x_1 U x_2) - \text{Kill}))$
      - $f(x_1) \land f(x_2) = (\text{Gen } U (x_1 - \text{Kill}) \lor \text{Gen } U (x_2 - \text{Kill}))$
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: \( f: \text{Gen} = \{d_1\} \); \( \text{Kill} = \{d_2\} \)
III. Properties of Iterative Algorithm

• **Given:**
  ∧ and monotone data flow framework
  Finite descending chain
  ⇒ Converges

• **Initialization of interior points to T**
  ⇒ 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 $i$
    
    $$f_p = f_{n_k} \cdots f_{n_1}, \text{ p is a path through nodes } n_1, ..., n_k$$

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

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

```
if sqr(y) >= 0
    if true
        x = 1
    else
        x = 0
```

- 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(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
  ⇒ \( FP \leq MFP = MOP \leq IDEAL \)