Lecture 3

Foundation of Data Flow Analysis

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

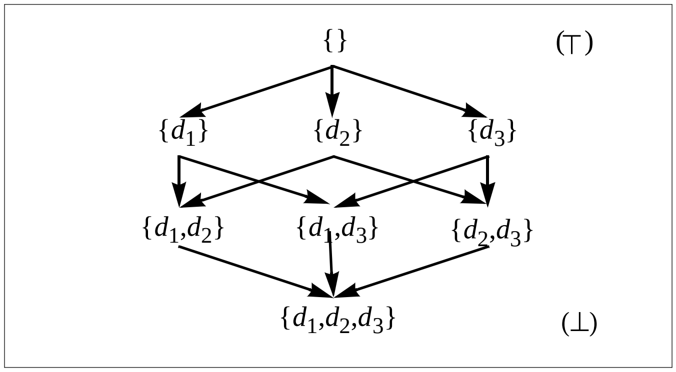
- **A semi-lattice S = < a set of values *V*, a meet operator $\wedge$ >**

- **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 | such that $x \subseteq \{d_1, d_2, d_3\}$}, $\wedge = \cup$



- $x \wedge y$ = first common descendant of *x* & *y* important
- Define top element $\top$, such that $x \wedge \top = x$
- Define bottom element $\perp$, such that $x \wedge \perp = \perp$
- 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 $x \wedge y = x$

$$y \xrightarrow{\text{path}} x \quad \equiv \quad (x \wedge y = x) \quad \equiv \quad (x \leq y)$$

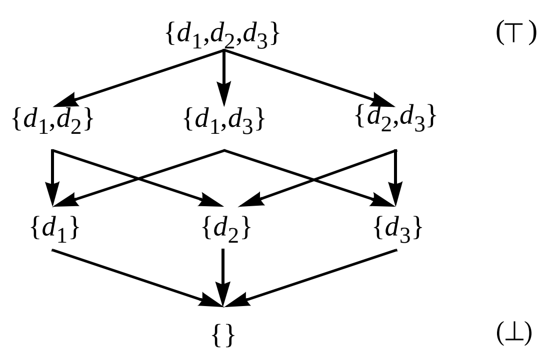
- **Properties of meet operator guarantee that $\leq$ is a partial order**
  - Reflexive: $x \leq x$
  - Antisymmetric: if $x \leq y$ and $y \leq x$ then $x = y$
  - Transitive: if $x \leq y$ and $y \leq z$ then $x \leq z$
- $(x < y) \equiv (x \leq y) \wedge (x \neq y)$
- **A semi-lattice diagram:**
  - Set of nodes: set of values
  - Set of edges $\{(y, x)$: $x < y$ and $\neg\exists z$ 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: $x \wedge x = x$
    - commutative: $x \wedge y = y \wedge x$
    - associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
  - 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

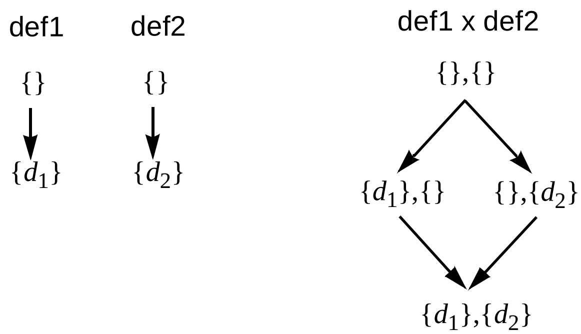
- Semi-lattice
  - V = {x | such that x $\subseteq$ { d_1, d_2, d_3}}
  - $\wedge = \cap$



  - $\leq$ 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



  - $<x_1, x_2> \le <y_1, y_2>$ iff $x_1 \le y_1$ and $x_2 \le y_2$

Descending Chain

- **Definition**
  - The **height** of a lattice is the largest number of > relations that will fit in a descending chain.

$$x_0 > x_1 > \dots$$

- **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 $f_1, f_2 \in F$, $f_1 \bullet f_2 \in F$

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
  - $f(x \wedge y) \leq f(x) \wedge f(y)$,
  - meet inputs, then apply f
    $\leq$
    apply f individually to inputs, then meet results

Example

- **Reaching definitions: $f(x) = Gen \cup (x - Kill)$, $\wedge = \cup$**
  - Definition 1:
    - Let $x_1 \leq x_2$,

      $f(x_1)$: $Gen \cup (x_1 - Kill)$

      $f(x_2)$: $Gen \cup (x_2 - Kill)$
  - Definition 2:
    - $f(x_1 \wedge x_2) = (Gen \cup ((x_1 \cup x_2) - Kill))$

      $f(x_1) \wedge f(x_2) = (Gen \cup (x_1 - Kill)) \cup (Gen \cup (x_2 - Kill))$

Important Note

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

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

III. Properties of Iterative Algorithm

- **Given:**
  - $\wedge$ 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

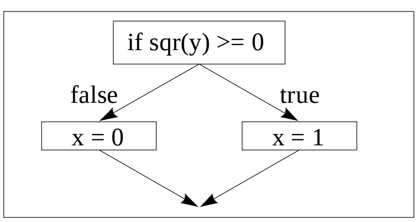
IV. What Does the Solution Mean?

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

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

    f_p = identify function, if p is an empty path

  - For each node n: $\wedge f_{p_i}$ (boundary value),
    for <u>all possibly executed paths</u> 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*:

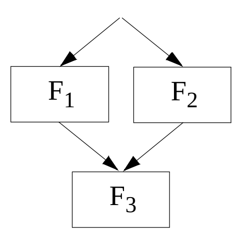
    MOP(n) = $\wedge\ f_{p_i}$ (boundary value), for all paths p_i reaching n

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

- **Data Flow Solution $\leq$ MOP $\leq$ 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 $\leq$ 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

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

  - distributive framework
    $\Rightarrow$ FP $\leq$ MFP $=$ MOP $\leq$ IDEAL