

Recapping Lecture 2: Data Flow Framework

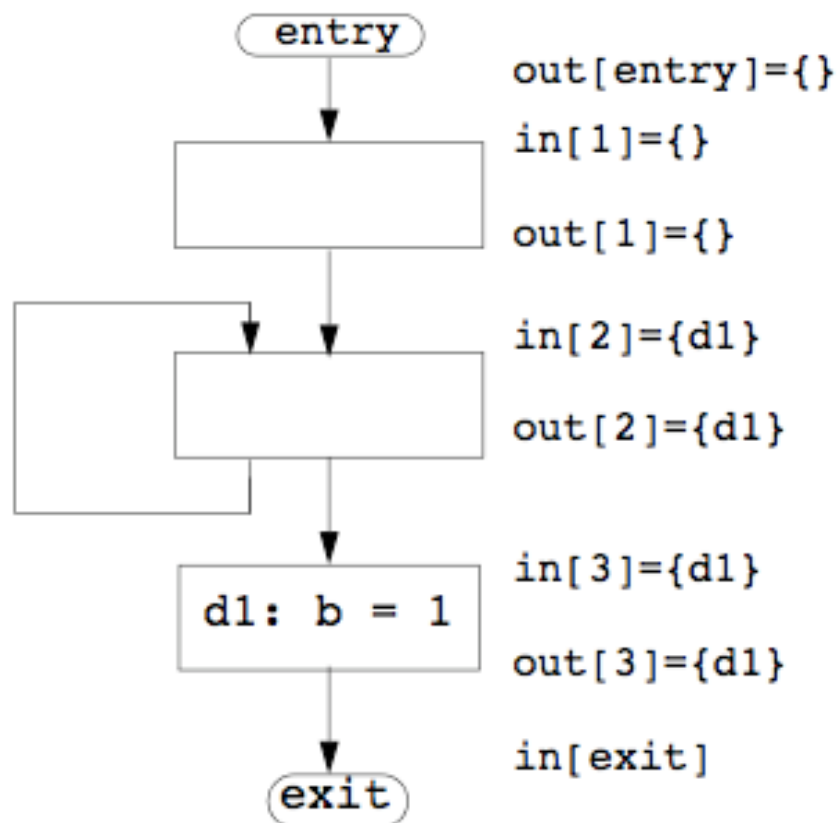
	Reaching Definitions	Live Variables
Domain	Sets of definitions	Sets of variables
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge out[pred(b)]$	backward: $in[b] = f_b(out[b])$ $out[b] = \wedge in[succ(b)]$
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	$f_b(x) = Use_b \cup (x - Def_b)$
Meet Operation (\wedge)	\cup	\cup
Boundary Condition	$out[entry] = \emptyset$	$in[exit] = \emptyset$
Initial interior points	$out[b] = \emptyset$	$in[b] = \emptyset$

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?

	MAY Reach	MUST Reach
Domain	Sets of definitions	
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge out[pred(b)]$	
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	
Meet Operation (\wedge)	\cup	
Boundary Condition	$out[entry] = \emptyset$	
Initial interior pts	$out[b] = \emptyset$	

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

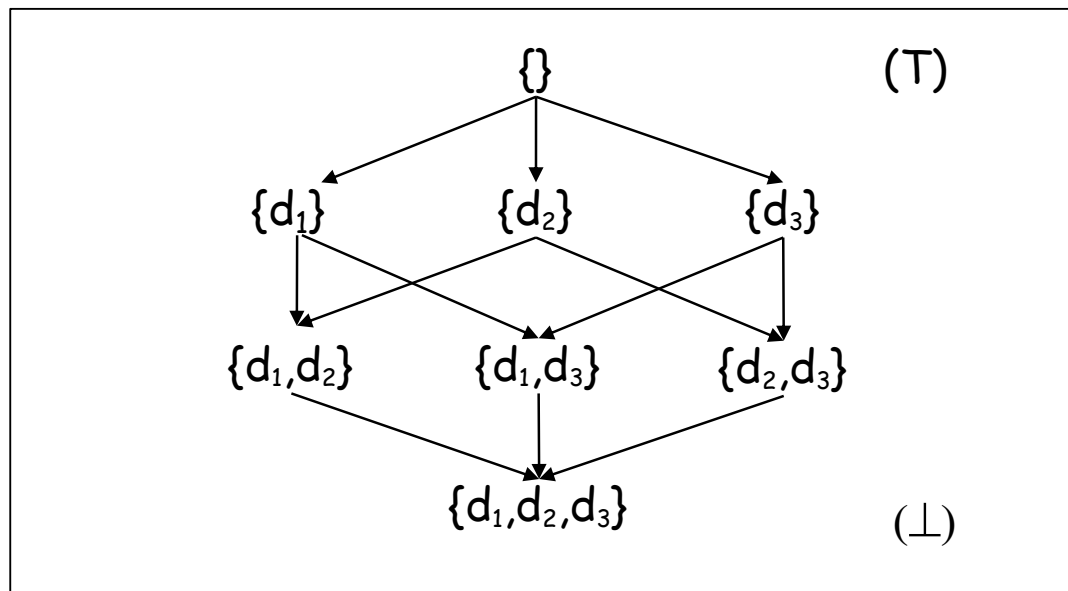
- Data-flow problems (F, V, \wedge) are defined by
 - A semi-lattice
 - domain of values V
 - meet operator $\wedge: V \times V \rightarrow V$
 - A family of transfer functions $F: V \rightarrow V$

Semi-lattice: Structure of the Domain of Values

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

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



- $x \wedge y =$ first common descendant of x & y **important**
- 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 \perp exists, if $x \wedge \perp = \perp$ for all x .

A Meet Operator Defines a Partial Order

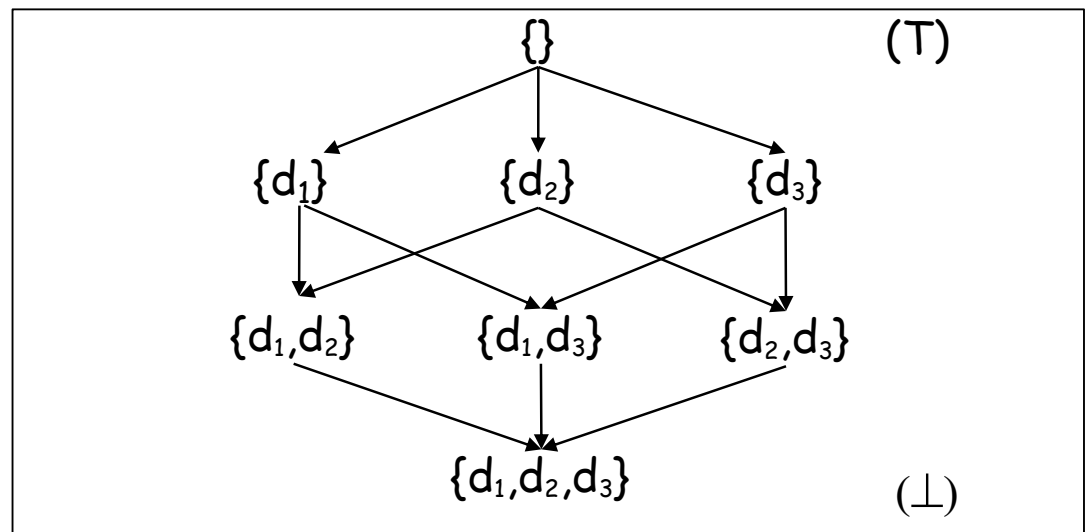
- Partial order of a meet semi-lattice

\leq : $x \leq y$ if and only if $x \wedge y = x$

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

- Meet operator: \cup

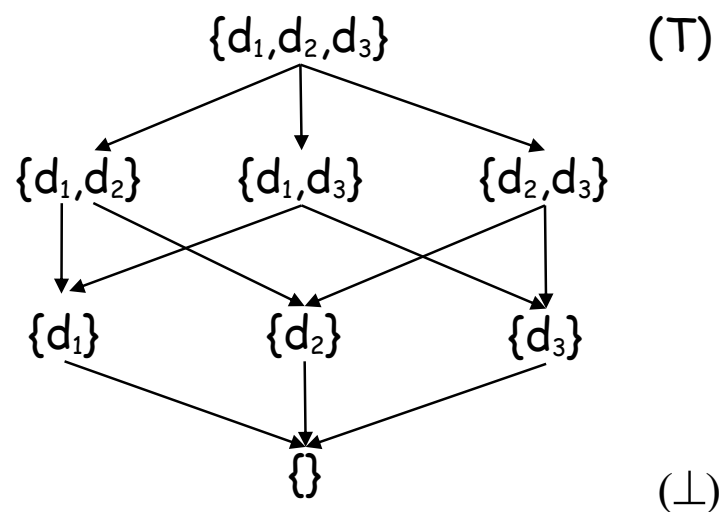
Partial order \leq :



- 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

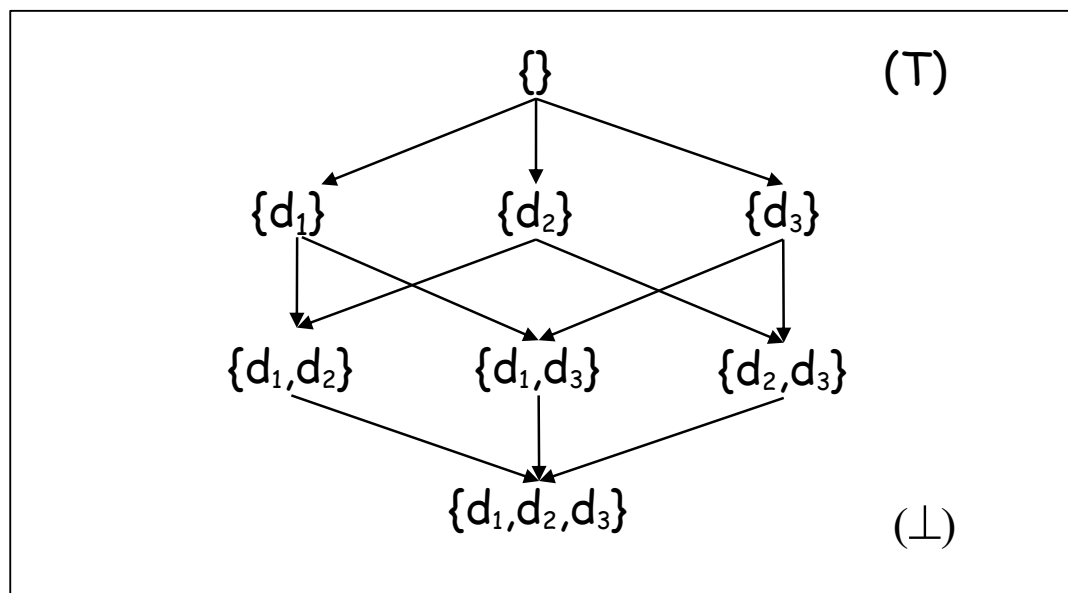
- Semi-lattice
 - $V = \{x \mid \text{such that } x \subseteq \{d_1, d_2, d_3\}\}$
 - $\wedge = \cap$



– \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: $x \wedge y =$ First common descendant of x & y
- Largest: top element \top , if $x \wedge \top = x$ for all x .
- Smallest: bottom element \perp , if $x \wedge \perp = \perp$ for all x .

Drawing a Semi-Lattice Diagram

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

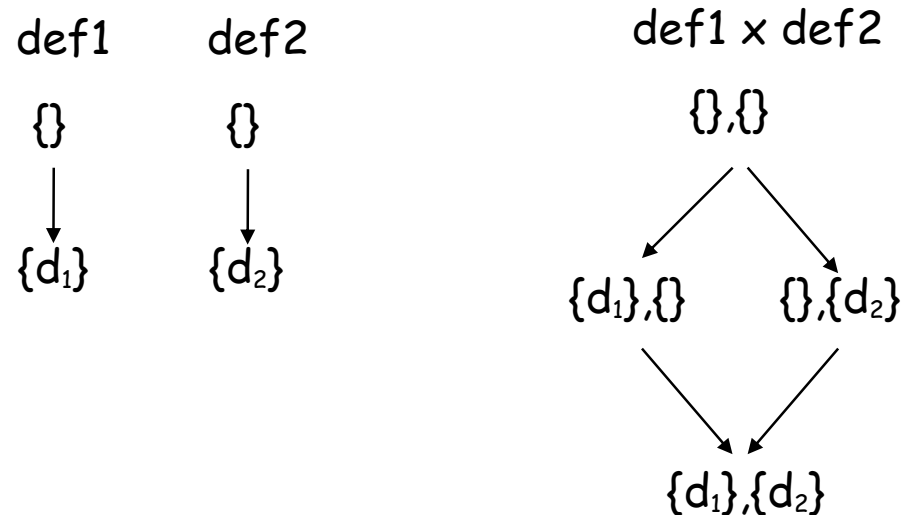
Summary

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 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



- $\langle x_1, x_2 \rangle \leq \langle y_1, y_2 \rangle$ 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.

$$x_0 > x_1 > \dots$$

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

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

Example

- Reaching definitions: $f(x) = \text{Gen} \cup (x - \text{Kill})$, $\wedge = \cup$

– 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:

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

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

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: $f: \text{Gen} = \{d_1\} ; \text{Kill} = \{d_2\}$

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:
{ in[b], out[b] | b in the program }
- We need to prove the invariant:
 - Values assigned to the same in[b] or 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.

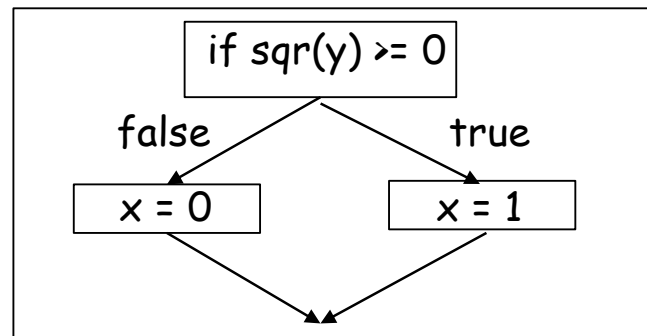
Sketch of Inductive Proof

For each IN/OUT of an interior program point:

- Invariant: new value \leq old value in any step
- Start with T (largest value)
- Proof by induction
 - 1st transfer function or meet operator: new value \leq old value (T)
 - Meet operation:
 - Assume new inputs \leq old inputs, new output \leq old output
 - Transfer function (in a monotone framework)
 - Assume new inputs \leq old inputs, new output \leq old output

IV. What Does the Solution Mean?

- **IDEAL data flow solution**
 - Let $f_1, \dots, f_m : \in F$, 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, \dots, n_k
 $f_p =$ identify function, if p is an empty path
 - For each node n : $\wedge f_{p_i}$ (boundary value),
for all possibly executed paths p_i reaching n
 - Example



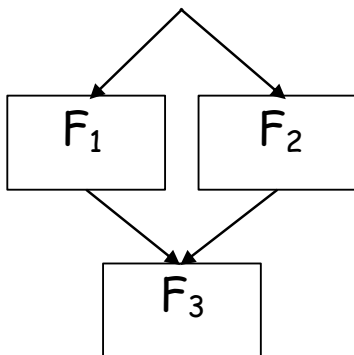
- **Determining all possibly executed paths is undecidable**

Meet-Over-Paths MOP

- Err in the conservative direction
- **Meet-Over-Paths MOP**
 - Assume every edge is traversed
 - For each node n :
 - $MOP(n) = \wedge f_{p_i}$ (boundary value), for all paths p_i reaching n
- **Compare MOP with IDEAL**
 - MOP includes more paths than IDEAL
 - $MOP = IDEAL \wedge \text{Result}(\text{Unexecuted-Paths})$
 - $MOP \leq IDEAL$
 - 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
 - ⇒ $FP \leq MFP = MOP \leq IDEAL$