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

Recapping Lecture 2: Data Flow Framework

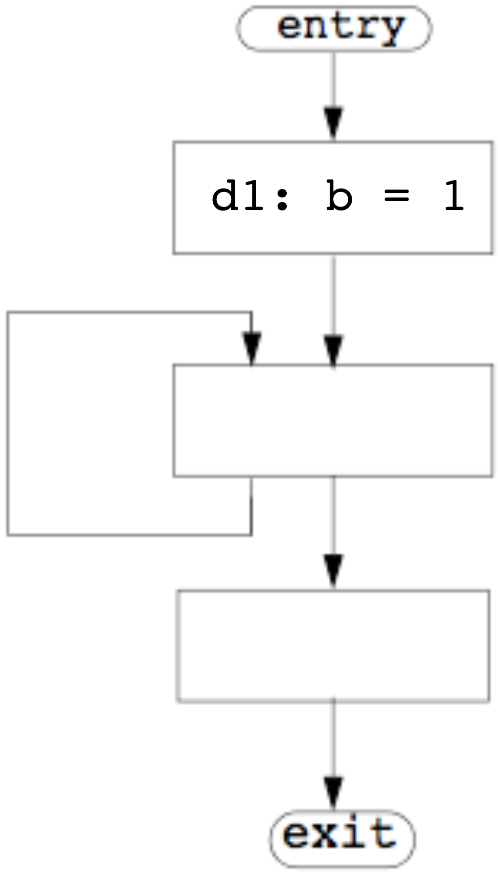
	Reaching Definitions	**Live Variables**
Domain	Sets of definitions	Sets of variables
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge\ out[pred(b)]$	backward: $in[b] = f_b(out[b])$ $out[b] = \wedge\ in[succ(b)]$
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	$f_b(x) = Use_b \cup (x - Def_b)$
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) <u>must</u> 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?**

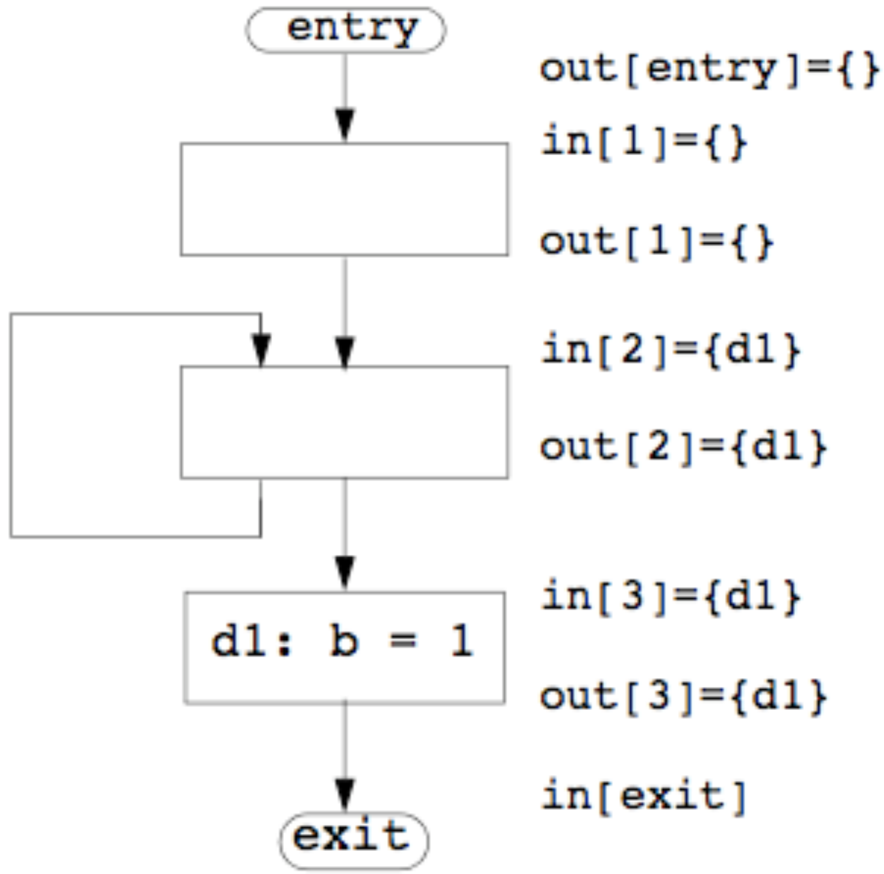
	MAY Reach	**MUST Reach**
Domain	Sets of definitions	
Direction	forward: $out[b] = f_b(in[b])$ $in[b] = \wedge\ out[pred(b)]$	
Transfer function	$f_b(x) = Gen_b \cup (x - Kill_b)$	
Meet Operation ($\wedge$)	$\cup$	
Boundary Condition	$out[entry] = \varnothing$	
Initial interior pts	$out[b] = \varnothing$	

Problem 1: Example



- Quiz 1: Does the precise answer satisfy the equations?
- Quiz 2: Are there alternative solutions that satisfy the equations?
- Quiz 3: Are the alternative solutions incorrect?
- Quiz 4: How do you get the precise answer?
- Quiz 5: What is a wrong answer?

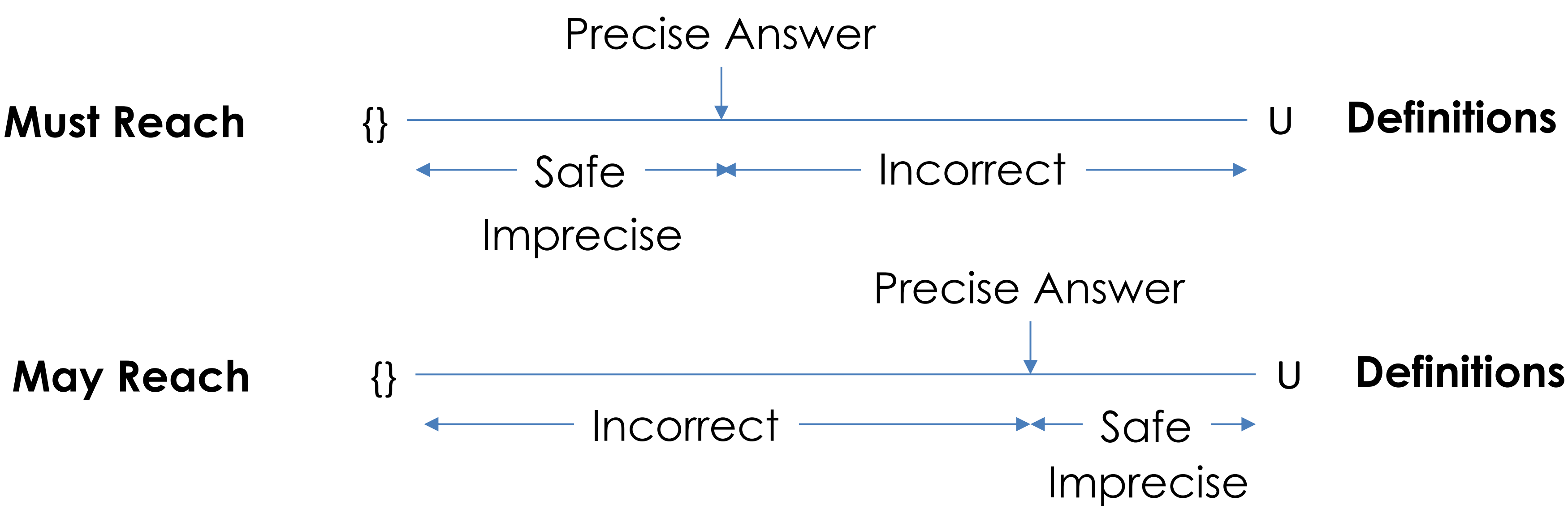
Problem 2: A legal solution to (**May**) Reaching Def?



- Quiz 1: Is this a solution to the equation?
- Quiz 2: Will the worklist algorithm generate this answer?
- Quiz 3: Is this solution safe or incorrect?

Summary

- There may be multiple (fixed-point) solutions to a set of equations
  - Non-solutions are **incorrect**
  - All solutions to the equations are considered **safe, but maybe imprecise**
  - Initialization of the interior points gives different solutions
- There is a *direction* to the solutions
- Answer is **precise** only when the initialization of interior points is
  - U for must-reach
  - {} for may-reach



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?**

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 x V $\rightarrow$ V
  - A family of transfer functions F: V $\rightarrow$ V

Lecture Outline

I	Semi-lattice (set of values, meet operator)
II	Transfer functions
III	Correctness, precision and convergence
IV	Meaning of data flow solution

Semi-lattice: Structure of the Domain of Values

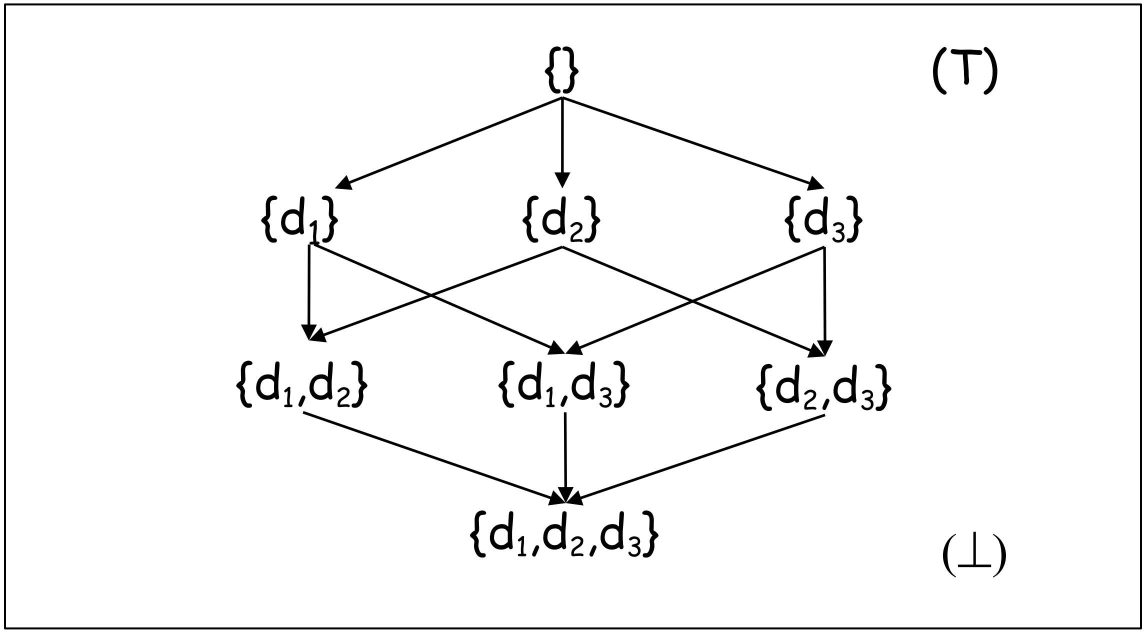
- **A semi-lattice S = <a set of values** V, **a meet operator** $\wedge$**>**

- **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

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



- $x \wedge y$ = 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

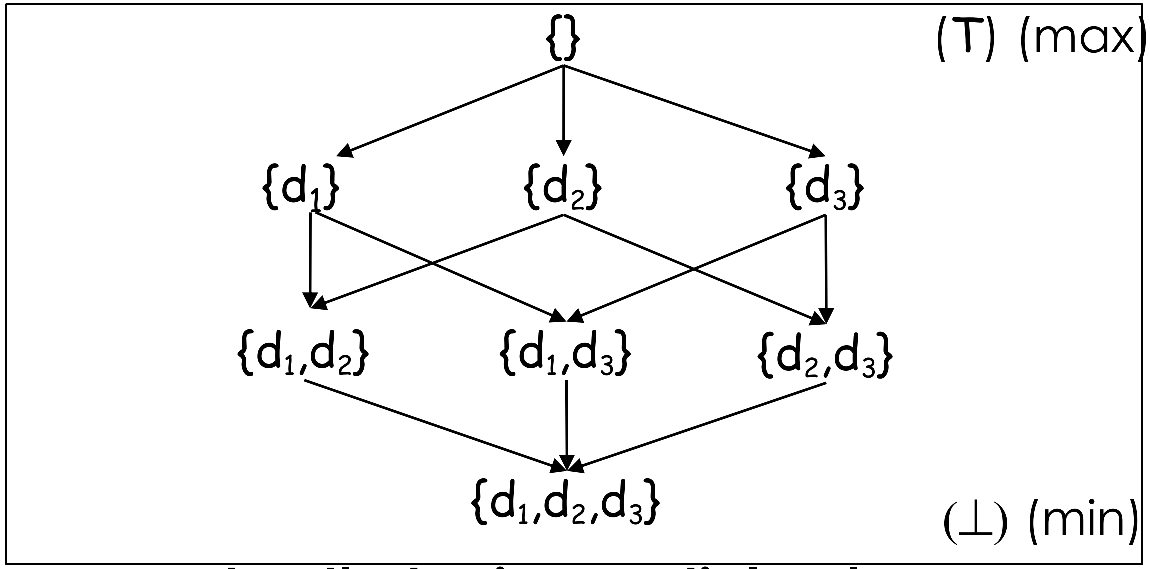
- **Partial order of a meet semi-lattice**

  ≤ : $x \le y$ if and only if $x \wedge y = x$

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

- Meet operator: U

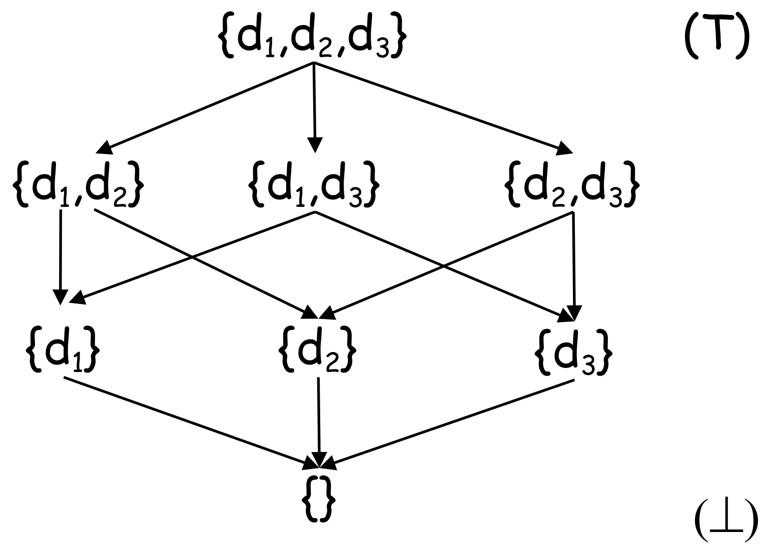
  Partial order ≤ :



- **Properties of meet operator guarantee that ≤ is a partial order**
  - Reflexive: $x \le x$
  - Antisymmetric: if $x \le y$ and $y \le x$ then $x = y$
  - Transitive: if $x \le y$ and $y \le z$ then $x \le z$

Another Example

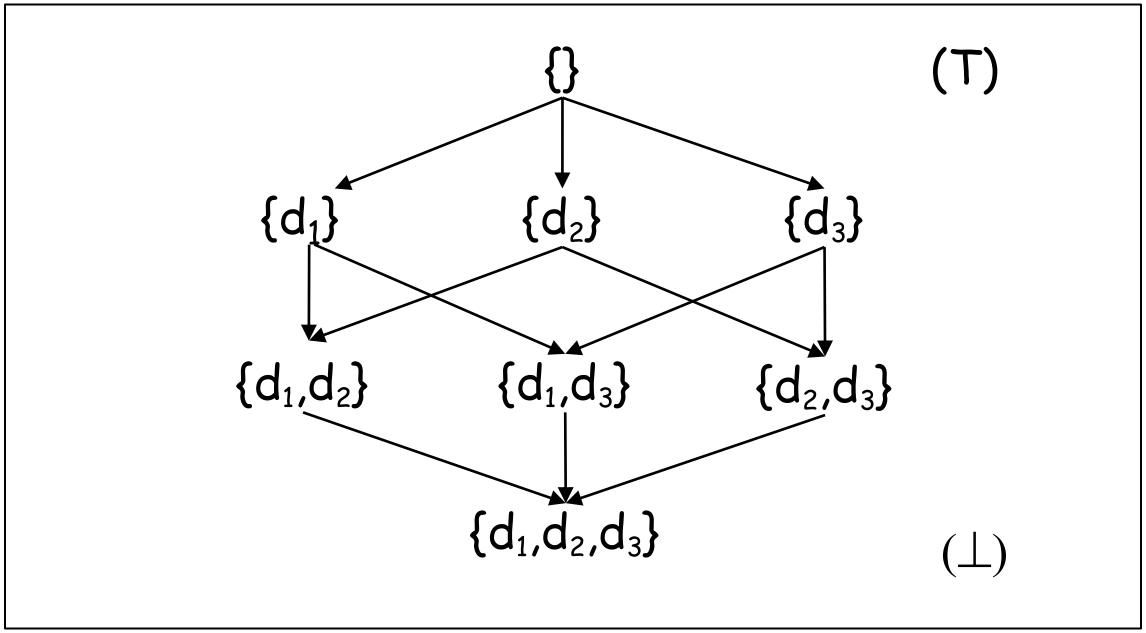
- Semi-lattice
  - V = {x | such that $x \subseteq \{d_1, d_2, d_3\}$}
  - $\wedge = \cap$



  - $\leq$ 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 T, if $x \wedge T = x$ 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
  - $\langle x_1, x_2 \rangle \le \langle y_1, y_2 \rangle$ iff $x_1 \le y_1$ and $x_2 \le y_2$
- **Example**: Union of definitions
  - For each element

$$
\begin{tikzcd}
\text{def1} & \text{def2} \\
\{\} \arrow[d] & \{\} \arrow[d] \\
\{d_1\} & \{d_2\}
\end{tikzcd}
\qquad
\begin{tikzcd}
& \text{def1} \times \text{def2} & \\
& \{\},\{\} \arrow[dl] \arrow[dr] & \\
\{d_1\},\{\} \arrow[dr] & & \{\},\{d_2\} \arrow[dl] \\
& \{d_1\},\{d_2\} &
\end{tikzcd}
$$

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

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 U (x_1 - Kill)

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

      $f(x_1) \wedge f(x_2)$ = (Gen U (x_1 - Kill)) U (Gen U (x_2 - 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 $f(x) \le x$
  - e.g. Reaching definition for two definitions in program
  - suppose: f: Gen = $\{d_1\}$; Kill = $\{d_2\}$
  - Quiz: What are the inputs and outputs of f?

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 (Maximum Fixed Point),
  because any larger value is not a solution.

Sketch of Inductive Proof

The boundary is set to the correct answer and never changed

For each IN/OUT of an interior program point:

- Invariant: new value ≤ old value in any step
- Initialize interior points with T (largest value)
- Proof by induction
  - Base case:
    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

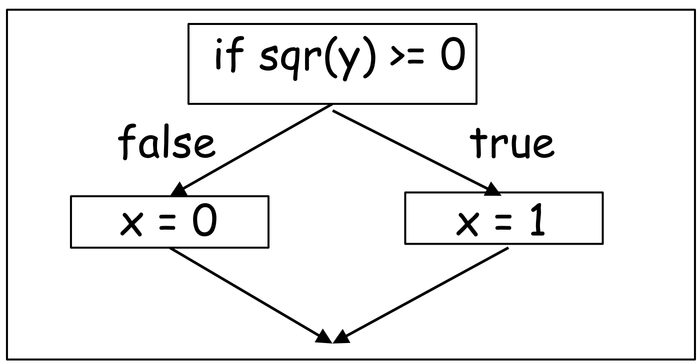
IV. What Does the Solution Mean?

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

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

    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:

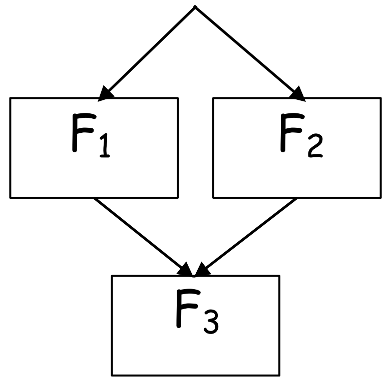
  - MOP(*n*) = $\wedge f_{p_i}$ (boundary value), for all paths p_i reaching *n*

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

- **MOP ≤ 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 ≤ 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

    $\Rightarrow$ 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