Foundations of Data Flow Analysis

- Meet operator
- Transfer functions
- Correctness, Precision, Convergence, Efficiency
- Summary homework: Chapter 8.2

Questions on Data Flow Analysis

- Correctness
  - Equations will be satisfied when the program terminates. Is this the solution that we want?
- Precision: how good is the answer?
  - Is the answer ONLY a union of all possible execution paths?
- Convergence: will the answer terminate?
  - Or, will there always be some nodes that change?
- Speed: how fast is the convergence?
  - how many times will we visit each node?

A Unified Dataflow Framework

- Data flow problems are defined by
  - Domain of values: V (e.g., set of definitions in reaching definition analysis, set of variables in liveness analysis, set of expressions in global CSE)
    - $V = \{x \mid x \subseteq \{d_1, d_2, d_3\}\}$ where d_1, d_2, d_3 are definitions
  - Meet operator ($V \times V \rightarrow V$), initial value
  - A set of transfer functions $F: V \rightarrow V$
- Usefulness of this unified framework
  - We can answer above four questions for a family of problems which have the same properties for their meet operators and transfer functions

I. Meet Operator

- We expect the meet operator to satisfy the following properties:
  - commutative: $x \wedge y = y \wedge x$
  - idempotent: $x \wedge x = x$
  - associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
  - There is a Top element T such that $x \wedge T = x$

I. Meet Operator

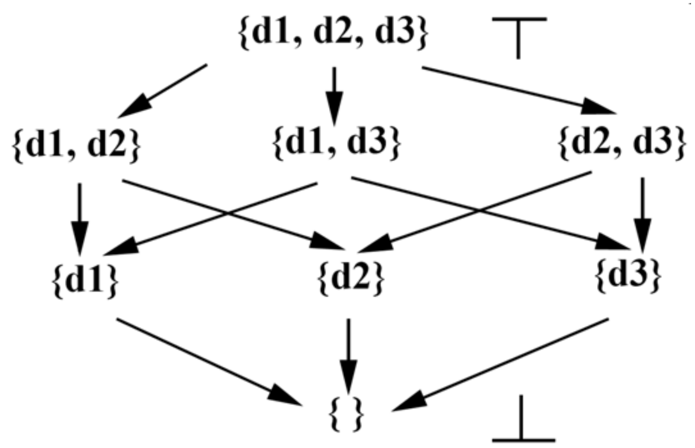
- meet operators that satisfy those properties define a *partial ordering* on values, with $\leq$
  - Let us define $x \leq y$ if and only if $x \wedge y = x$
  - Then, $\leq$ is a partial ordering. Why?
    - Transitive: if $x \leq y$ and $y \leq z$ than $x \leq z$
      - $x \wedge y = x$, $y \wedge z = y$; $x \wedge z = x \wedge y \wedge z = x \wedge (y \wedge z) = x \wedge y = x$
    - Anti-symmetric: if $x \leq y$ and $y \leq x$ then $x = y$
      - $x \wedge y = x$, $y \wedge x = y$, $x \wedge y = y \wedge x$ (commutative); $x=y$
    - Reflexive: $x \leq x$
      - $x \wedge x = x$ (idempotent)

Review of Partial Ordering (Discrete Math)

- Binary Relation: a set of order pairs
  - E.g., $\leq$ defines a binary relation on integers R={(1,1), (1,2), (1,3)..., (2,2), (2,3),.....}
- Partial ordering: a binary relation R that is reflexive, anti-symmetric, and transitive
  - Reflexive: $(x,x) \in R$
  - Transitive: if $(x,y)\in R$, $(y,z)\in R$ => $(x,z)\in R$
  - Anti-symmetric: if $(x,y)\in R$, $(y,x)\in R$ => $x=y$
  - e.g., the set of integers is partially ordered with $\leq$ relation

Partial Ordering Example

- Let domain of values $V = \{x | x \subseteq \{d_1, d_2, d_3\}\}$
- Let $\wedge = \cap$
  - How partial ordering with $\leq$ is defined?



  - Top and Bottom elements
    - Top T such that $x \wedge T = x$ is $\{d_1, d_2, d_3\}$
    - Bottom $\perp$ such that $x \wedge \perp = \perp$ is $\{\ \}$

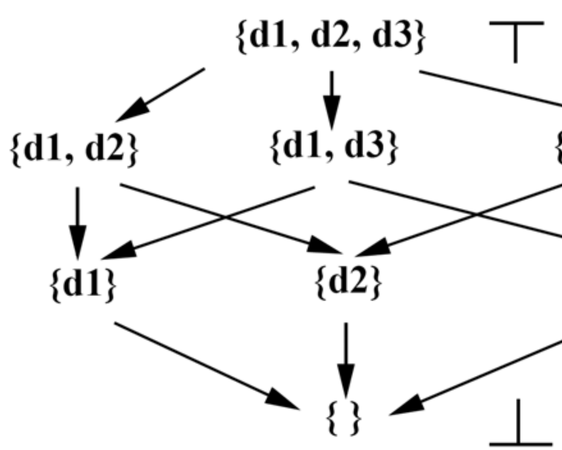
Partial Ordering Example

- Let domain of values $V = \{x | x \subseteq \{d_1, d_2, d_3\}\}$
- Let $\wedge = \cup$
  - How partial ordering with $\leq$ is defined?

  - Top and Bottom elements
    - Top T such that $x \wedge T = x$ is $\{ \}$
    - Bottom $\perp$ such that $x \wedge \perp = \perp$ is $\{d_1, d_2, d_3\}$

Semi-Lattice

- Values and meet operator in a data flow problem defines a semi-lattice (i.e., there exists T, but not necessarily $\perp$)
  - If x, y are ordered: $x \leq y \Rightarrow x \wedge y = x$
  - What if x and y are not ordered? $w \leq x, w \leq y \Rightarrow w \leq x \wedge y$
    - Why? $w \wedge x = w$, $w \wedge y = w$, then $w \wedge (x \wedge y) = (w \wedge x) \wedge y = w \wedge y = w$, so $w \leq x \wedge y$
    - This means that w cannot be greater than $x \wedge y$



Review of Lattice

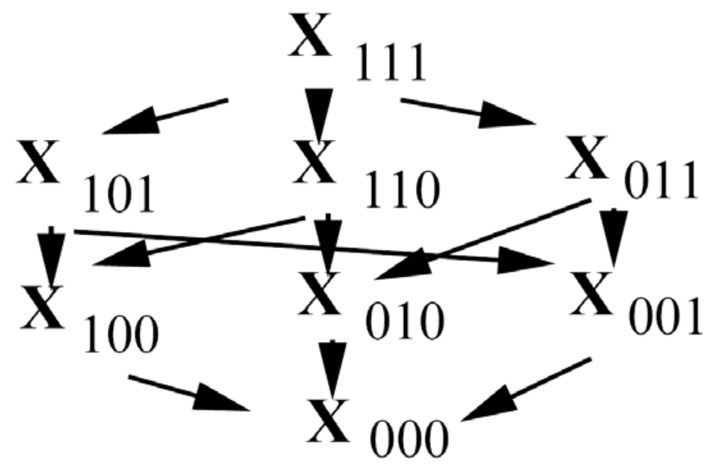
- Lattice: Characterizing various computation models (e.g., Boolean Algebra)
  - A partially ordered set in which every pair of elements has a unique greatest lower bound (*glb*) and a unique least upper bound (*lub*)
  - Each finite lattice has both a least ($\perp$) and a greatest (T) element such that for each element a, a≤T and $\perp$≤a
  - Due to the uniqueness of *lub* and *glb*, binary operations $\vee$ and $\wedge$ (meet) are defined such that $a \vee b$ = *lub*(a,b) and $a \wedge b$ = *glb*(a,b)

Representation of Each Variable

- When $\wedge = \cap$, we can represent a variable by
  - 1 means it exists, 0 means it does not exist

Lattice for each variable:

Lattice for three variables:



Descending Chain

- The height of a lattice
  - Def: the largest number of ≥ relations that will fit in a descending chain:

    $x_0 > x_1 > \ldots$
  - E.g., height of a lattice in reaching definitions:

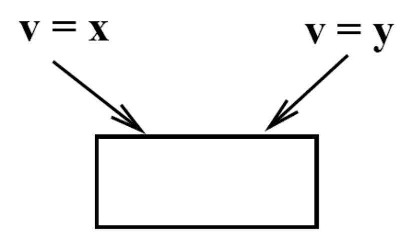
    Number of definitions (# of 1 bit transitions)
- Important property: finite descending chain
  - Useful for proving convergence
  - For finite lattice, there is finite descending chain, obviously
  - Can infinite lattice have a finite descending chain?

An example: constant propagation and folding

- Domain of values: undef, .. -2, -1, 0, 1, 2, .., not-a-constant
- What is the meet operator and the lattice for this problem?

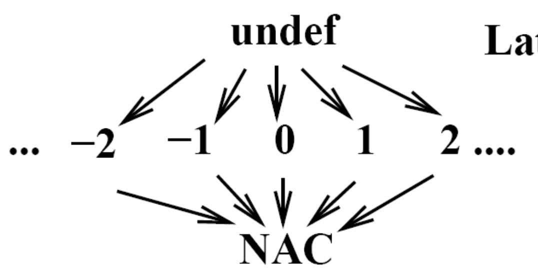
Meet operator $\wedge$

x	y	$x \wedge y$	
c1	c2	NAC	c1!=c2
c1	c2	c1	c1=c2
undef	c1	c1	
undef	NAC	NAC	
undef	undef	undef	
NAC	c1	NAC	
NAC	undef	NAC	
NAC	NAC	NAC	



v: a variable that takes x or y

Lattice



- Finite descending chain of length 2

Finite descending chain is important for convergence

- Its height can be the upper bound of the running time
- One more property needed: monotone framework

II. Transfer Functions

- Basic Property $f: V \rightarrow V$
  - Has an identity function
    - There exists an f such that $f(x) = x$ for all x
  - Closed under composition
    - if $f_1, f_2 \in F$, $f_1 \cdot f_2 \in F$
- Some useful properties of $\wedge$
  - $x \wedge y \leq x$
  - If $x \leq y$, then $w \wedge x \leq w \wedge y$

Monotonicity

- A framework (F, V, $\wedge$) is **monotone** iff
  - $x \le y$ implies $f(x) \le f(y)$
  - i.e. a "smaller or equal" input to the same function will always give a "smaller or equal" output
- Equivalently, a framework (F, V, $\wedge$) is **monotone** iff
  - $f(x \wedge y) \le f(x) \wedge f(y)$ Why equivalent?

    (1) $x \wedge y \le x$, so $f(x \wedge y) \le f(x)$ (2) $x \wedge y \le y$, so $f(x \wedge y) \le f(y)$. Now $f(x \wedge y) \le f(x) \wedge f(y)$
  - This means (1) merge input then apply f is **smaller than or equal to** (2) apply f individually then merge
    - How do our iterative algorithms work? Like (1) or like (2) ?

Example

- The case of reaching definition analysis
  - $f(x) = Gen \cap (x\text{-}Kill)$, $\wedge = \cup$
  - Def. 1: $x_1 \leq x_2$: $Gen \cup (x_1 - Kill) \leq Gen \cup (x_2 - Kill)$
  - Def. 2: $(Gen \cup ((x_1 \cup x_2) - Kill)) \leq$
    $(Gen \cup (x_1 - Kill) \cup Gen \cup (x_2 - Kill))$
    - Actually, it is = (identical) for reaching definitions
  - Reaching definitions are monotone

Meaning of Monotone Framework

Monotone framework does not mean that $f(x) \le x$

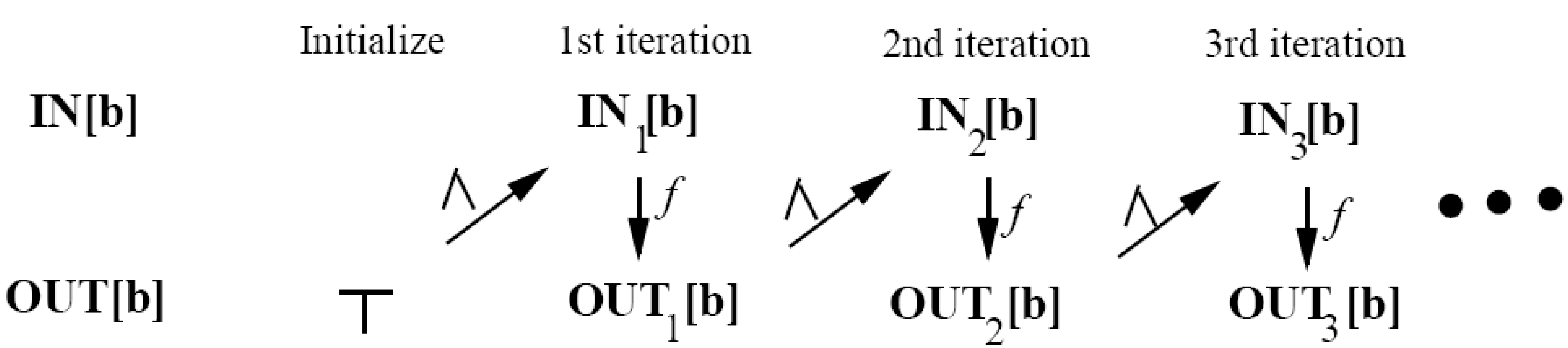
- e.g., reaching definitions; suppose f_b : Gen = $\{d_1\}$ Kill = $\{d_2\}$, then if $x = \{d_2\}$ $f(x) = \{d_1\}$

Then, what does the monotonicity really mean?

- It is related to convergence

Convergence of iterative solutions

- If input(second iteration) ≤ input(first iteration)
  - result(second iteration) ≤ result(first iteration)
    - If $x \leq y$, then $w \wedge x \leq w \wedge y$
  - If input are going down, the output is going down
  - This property and the finite-descending chain give you the convergence of iterative solution



Distributive Framework

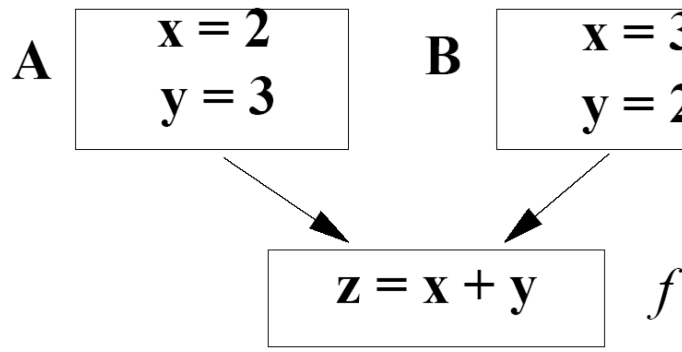
- A framework (F, V, $\wedge$) is **distributive** iff
  - $f(x \wedge y) = f(x) \wedge f(y)$
  - i.e. merge input then apply f is equal to apply the transfer function individually then merge result
  - e.g. reaching definitions, live variables
  - What we do in iterative approaches is somewhat like $f(x \wedge y)$, while the ideal solution is somewhat like $f(x) \wedge f(y)$
  - $f(x \wedge y) \le f(x) \wedge f(y)$ means that $f(x \wedge y)$ gives you less precise information
  - An example problem that is not distributive:

Constant propagation

Non-distributive: Constant Propagation

$z = x + y$

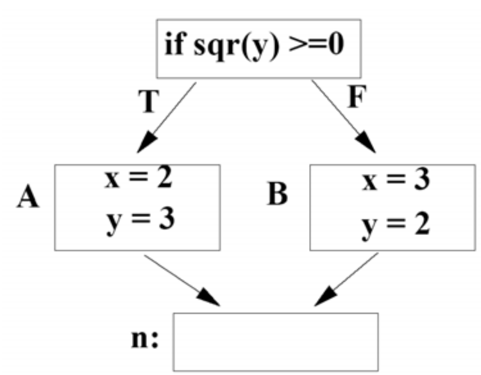
x	y	z	
c1	NAC	NAC	
	c2	c1+c2	
	undef	undef	
undef	NAC	undef	
	c2	undef	
	undef	undef	
NAC	undef	undef	
		NAC	



- Out[A] = {x = 2, y = 3}, Out[B] = { x = 3, y = 2 }
- f(Out[A]) = {z = 5, x = 2, y = 3}, f(Out[B]) = {z = 5, x = 3, y = 2}
- f(Out[A]) $\wedge$ f(Out[B]) = {z = 5, x = NAC, y = NAC}
- f(Out[A] $\wedge$ Out[B]) = {z = NAC, x = NAC, y = NAC}

III. Data Flow Analysis

- Definition
  - Let $f_1, \ldots, f_m : \in$, f_i is the transfer function for node i
    - $f_p = f_{nk} \cdot f_{nk-1} \cdot f_{n1}$, p is a path through nodes $n_1, .., n_k$
    - f_p = identity function, if p is an empty path
- Ideal data flow answer
  - For each node n:
    - $\wedge f_{pi}(\text{init})$, for all possibly "executed" paths *pi*, reaching *n*



  - Unfortunately, determining all possibly executed paths is undecidable

Meet-Over-Paths (MOP)

- Takes an error in conservative direction (e.g., reaching def: consider more (all possible) paths)
- Meet-Over-Paths (MOP)
  - For each node n:
    MOP(n) = $\wedge f_{pi}$(init), for all paths *pi*, reaching n
  - A path exists as long there is an edge in the code
  - Consider more paths than necessary
  - MOP = Perfect-Solution $\wedge$ Solution-to-Unexecuted-Paths
  - MOP $\leq$ Perfect-Solution
  - Potentially more constrained, so solution is small and safe
- Desirable solution: as close to MOP as possible

Solving Data Flow Equations

- What we did for iterative solution:
  - We just solved those equations, not for all execution paths
  - Any solution satisfying equations: Fixed Point (FP) Solution
- Iterative algorithms - the case of reaching definitions
  - Initialize out[b] to {}
  - If converges, it computes Maximum Fixed Point (MFP) solution: MFP is the largest of all solutions to equations
  - How iterative algorithms give you the MFP?

    Initialize T (init). Move down only when we see a definition
- Properties:
  - FP $\leq$ MFP **$\leq$** MOP $\leq$ Perfect-Solution
  - Which has been proved

Correctness and Precision

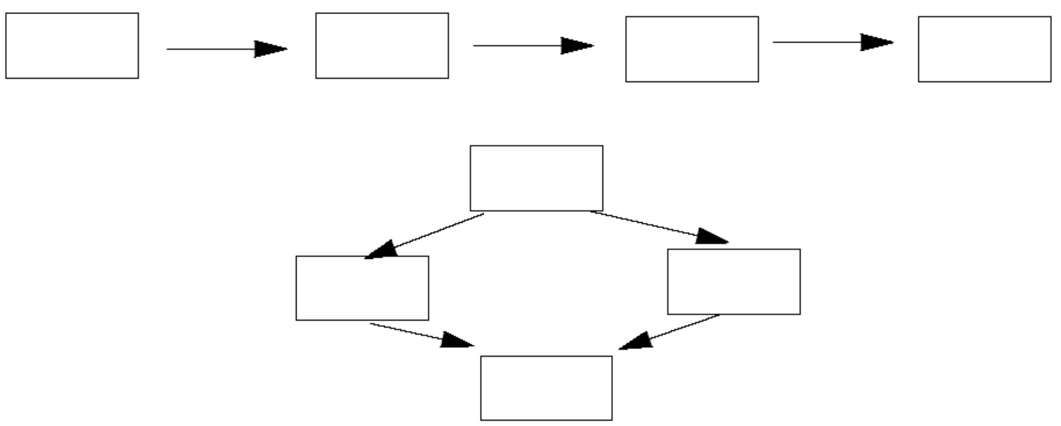
- If data flow framework is monotone, then if the algorithm converges, IN[b] ≤ MOP[b]
- If data flow framework is distributive, then if the algorithm converges, IN[b] = MOP[b]
  - Why? meet-early (iterative) = meet-late (MOP)
  - True for reaching definitions and live variables
- If monotone but not distributive
  - MFP ≠ MOP
  - True for constant propagation

Properties to Guarantee Convergence

- Monotone data flow framework converges if there is a finite descending chain
  - For each variable IN[b] and OUT[b], consider the sequence of values set to each variable across iterations
  - If sequence for IN[b] is monotonically decreasing, sequence for OUT[b] is monotonically decreasing. (OUT[b] is initialized to T)
  - If sequence for OUT[b] is monotonically decreasing, sequence for IN[b] is monotonically decreasing.

Speed of Convergence

- Convergence depends on the order of node visits



  - Use **reverse post-order** for of forward problems
    - Roughly corresponds to topologically sorted order in acyclic graph

- Reverse "direction" for backward problems

Computing Reverse Post-order

- Step 1: compute depth-first post-order

```
main () {
    count = 1;
    Visit(root);
}
Visit (n) {
    for each successor s that has not been visited
            Visit (s);
    PostOrder(n) = count;
    count++;
}
```

- Step 2: reverse the post-order

```
For each node i;
    rPostOrder = NumNodes – PostOrder(i)
```

rPost-order Forward Iterative Algorithm

- Input: Control Flow Graph CFG = (N, E, Entry, Exit)

```
/* Initialize */
        OUT[Entry]  = { }
        for all nodes i
              OUT[i] = { }
        Change = TRUE

/* Iterate   */
        While ((Changes) {
              Change = FALSE
              For each node i in rPostOrder {
                        IN[i] = U (OUT[p]), for all predescessors p of i
                        oldout = OUT[i]
                        OUT[i] = f_i(IN[i]) /* OUT[i] = GEN[i] U (IN[i] – KILL[i]) */
                        if (oldout != OUT[i]) {
                            Change = TRUE
              }
        }
/* Visit each node the same number of times */
```

Speed of Convergence

- If cycles do not add information
  - Information can flow in one pass down a series of nodes of increasing rPostorder order number
  - Passes determined by the number of back edges in the path which is, essentially, the nesting depth of the graph
- What is the depth ?
  - Corresponds the depth of intervals for “reducible” graphs (loops)
  - In real programs: average 2.75

Check List for Data Flow Problems

- Semi-Lattice
  - Set of values, meet operator, top & bottom, finite descending chain
- Transfer Function
  - function of each basic block, monotone, distributive
- Algorithm
  - initialization step (entry/exit)
  - visit order: rPostOrder
  - depth of the graph