More dataflow analysis

CSE 501

Spring 15

Reaching Definitions Summary

Lattice	Sets of definitions represented by bit-vectors
Transfer function	$OUT[B] = f_b(IN[B])$ $f_b(x) = (x - KILL[x]) \cup GEN[x]$
Meet operation	$IN[B] = \cup\ OUT[Predecessors]$
Boundary condition	OUT[entry] = 0….0
Initial condition	OUT[B] = 0….0

Questions

- Does the algorithm halt?
  - yes, because transfer function is monotonic
  - if increase IN, increase OUT
  - in limit, all bits are 1
- If bit is 0, does the corresponding definition ever reach basic block?
- If bit is 1, does the corresponding definition always reach the basic block?

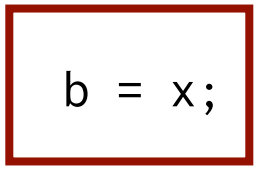
Live Variable Analysis

- A variable **v** is live at program point **p** if the value of **v** is used along some path in the flow graph from **p**
- Otherwise it is dead

- For each basic block,
  determine which variable is live
- Use bit vector to represent variables,
  one bit per **variable**

Live Variable Analysis

- Insight: look BACKWARDS to trace definitions from uses!



x must be live on entry to basic block!
Not true for b!

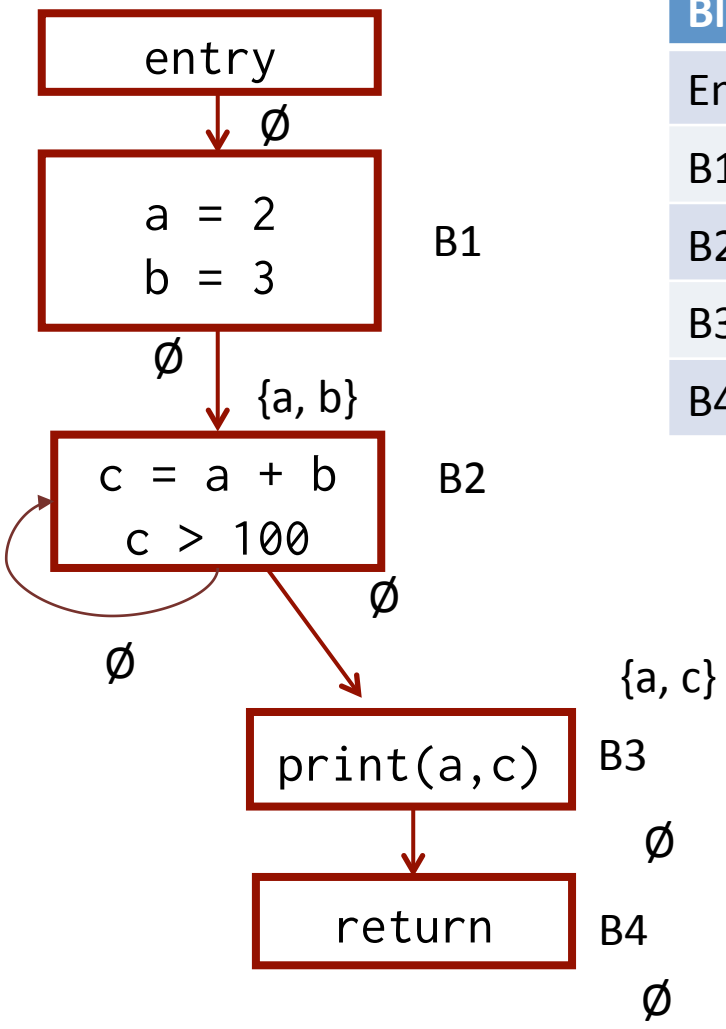
Transfer Function for Live Variable

- For statement s: x = y + z
  - Generates new live variables: USE[s] = {y, z}
  - Kills previously live variables: DEF[s] = x
  - Variables that were not killed are propagated: OUT[s] – DEF[s]
  - So: IN[s] = USE[s] U (OUT[s] – DEF[s])
- Similarly for basic block B:
  - IN[B] = USE[B] U (OUT[B] – DEF[B])

Set up

- Boundary condition: IN[exit] = Ø
- Initial conditions: IN[B] = Ø
- Meet operation:
  - OUT[B] = U IN[Successors]

Example

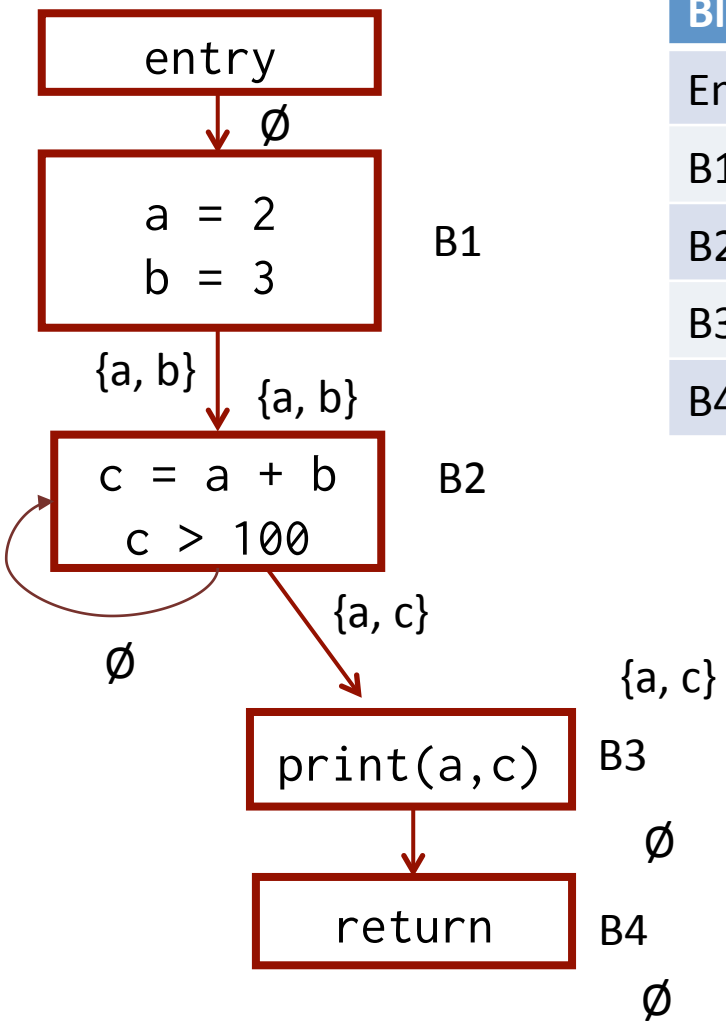


IN[B] = (OUT[B] – DEF[B]) U USE[B]
OUT[B] = U IN[Successors]

Block	DEF	USE
Entry	Ø	Ø
B1	{a, b}	Ø
B2	{c}	{a, b}
B3	Ø	{a, c}
B4	Ø	Ø

Are we done?

Example



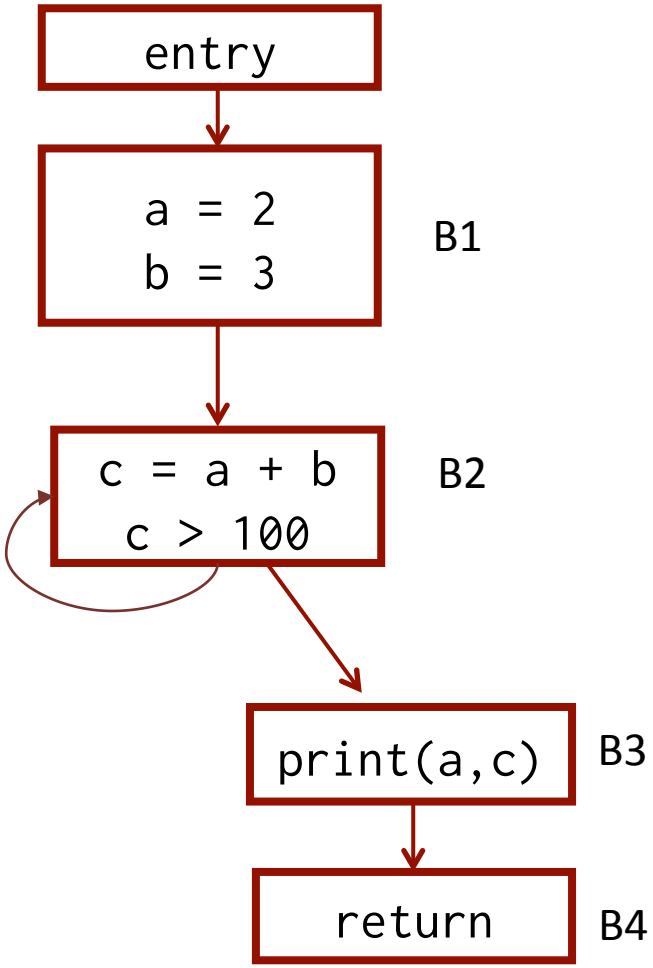
IN[B] = (OUT[B] – DEF[B]) U USE[B]
OUT[B] = U IN[Successors]

Block	DEF	USE
Entry	Ø	Ø
B1	{a, b}	Ø
B2	{c}	{a, b}
B3	Ø	{a, c}
B4	Ø	Ø

Are we done?

Example

IN[B] = (OUT[B] – DEF[B]) U USE[B]
OUT[B] = U IN[Successors]



Block	DEF	USE
Entry	Ø	Ø
B1	{a, b}	Ø
B2	{c}	{a, b}
B3	Ø	{a, c}
B4	Ø	Ø

Block	IN	OUT
Entry	Ø	Ø
B1	Ø	{a, b}
B2	{a, b}	{a, b, c}
B3	{a, c}	Ø
B4	Ø	Ø

Live Variables Summary

Lattice	Sets of variables represented by bit-vectors
Transfer function	$IN[B] = f_b(OUT[B])$ $f_b(x) = USE[x] \cup (x - DEF[x])$
Meet operation	$OUT[B] = \cup\ IN[Successors]$
Boundary condition	$IN[exit] = \emptyset$
Initial condition	$IN[B] = \emptyset$

Two analyses

	Live Variables	Reaching Definitions
Lattice	Sets of variables represented by bit-vectors	Sets of definitions represented by bit-vectors
Transfer function	IN[B] = f_b(OUT[B]) f_b(x) = USE[x] U (x – DEF[x]) Backward	OUT[B] = f_b(IN[B]) f_b(x) = (x – KILL[x]) U GEN[x] Forward
Meet operation	OUT[B] = U IN[Successors]	IN[B] = U OUT[Predecessors]
Boundary condition	IN[exit] = Ø	OUT[entry] = 0….0
Initial condition	IN[B] = Ø	OUT[B] = 0….0

Dataflow Analysis: Steps

- Decide on flow values (lattice)
- Forward / backward?
- Design transfer functions
- Design meet operation
- What are the boundary and initial conditions?

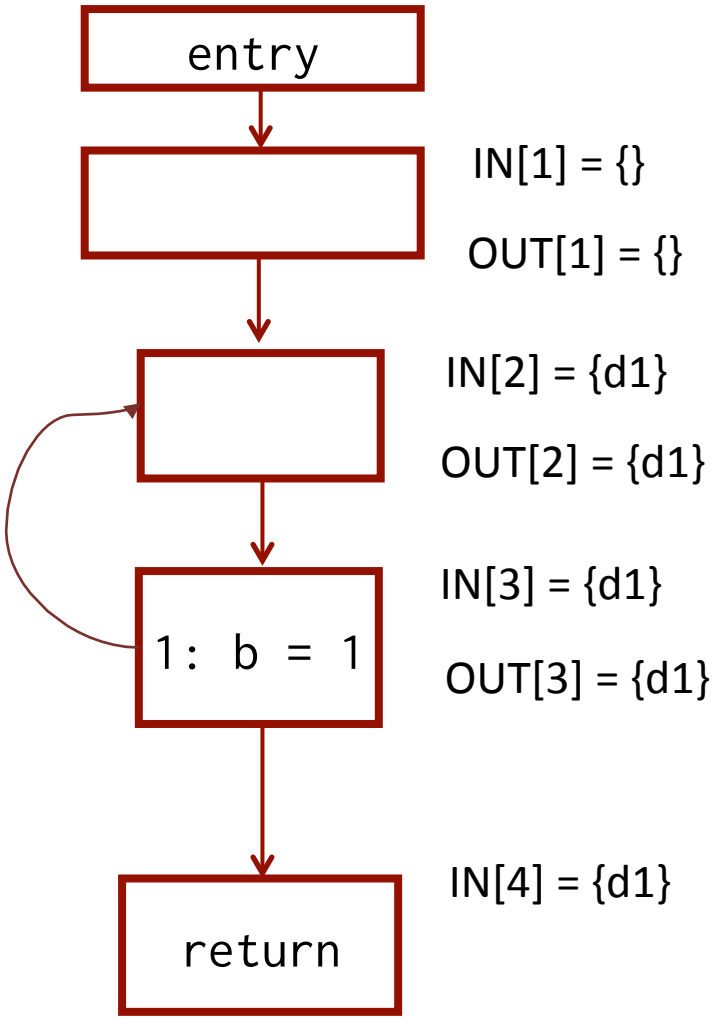
“Must-Reach” definitions

- A definition **D** must reach a program point **P** iff
  - **D** appears at least once along all paths that leads to **P**
  - **D** is not redefined along any path after the last appearance of **D** and before **P**

- How do we design a dataflow analysis for this?


SO ARE WE DONE NOW?
OR IS THE PARTY JUST GETTING STARTED?
memegenerator.net

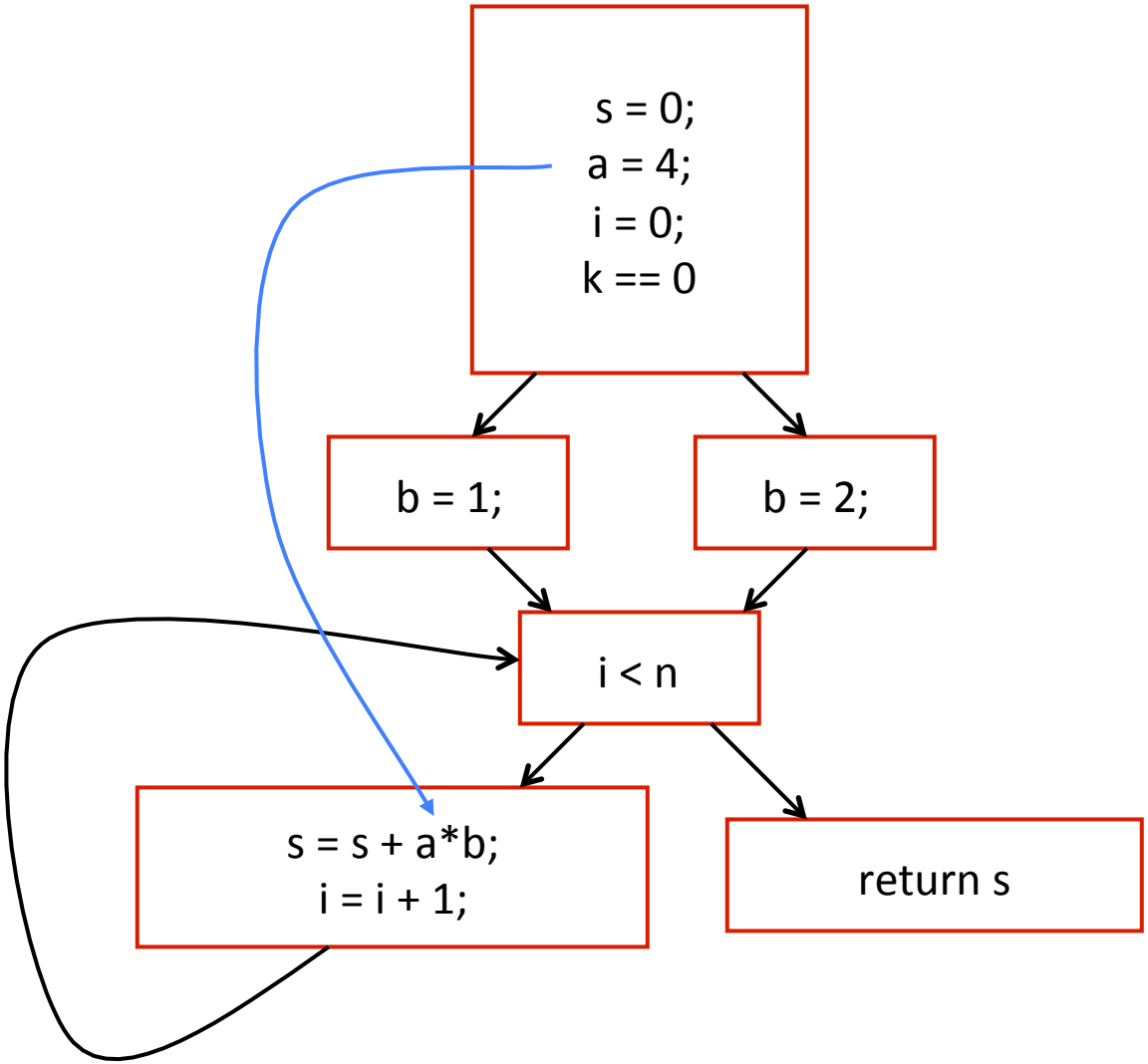
Legal solutions



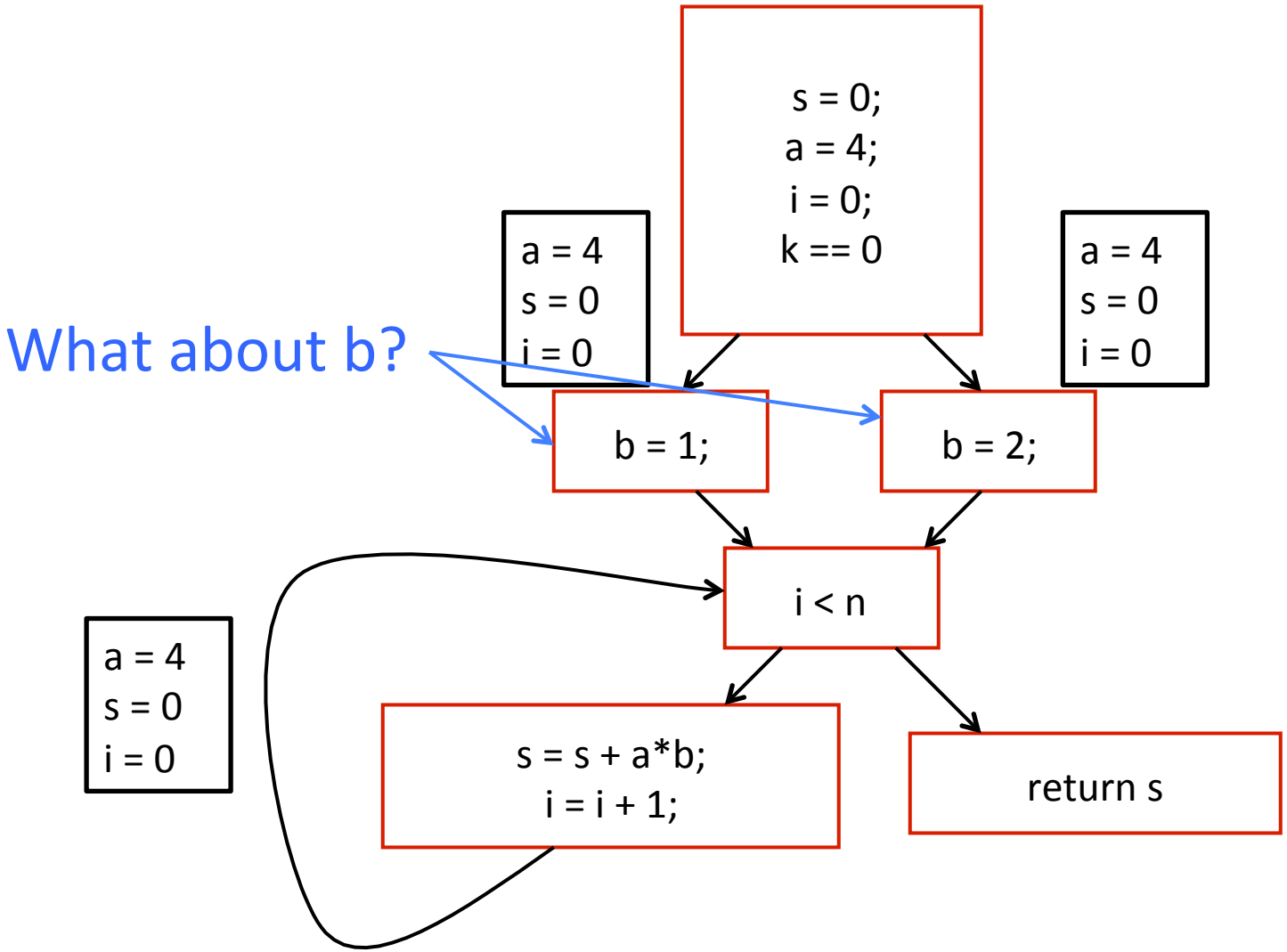
What do we care about?

- Correctness
- Precision
- Convergence
- Runtime

Back to Constant Propagation



Back to Constant Propagation



Constant Propagation: lattice

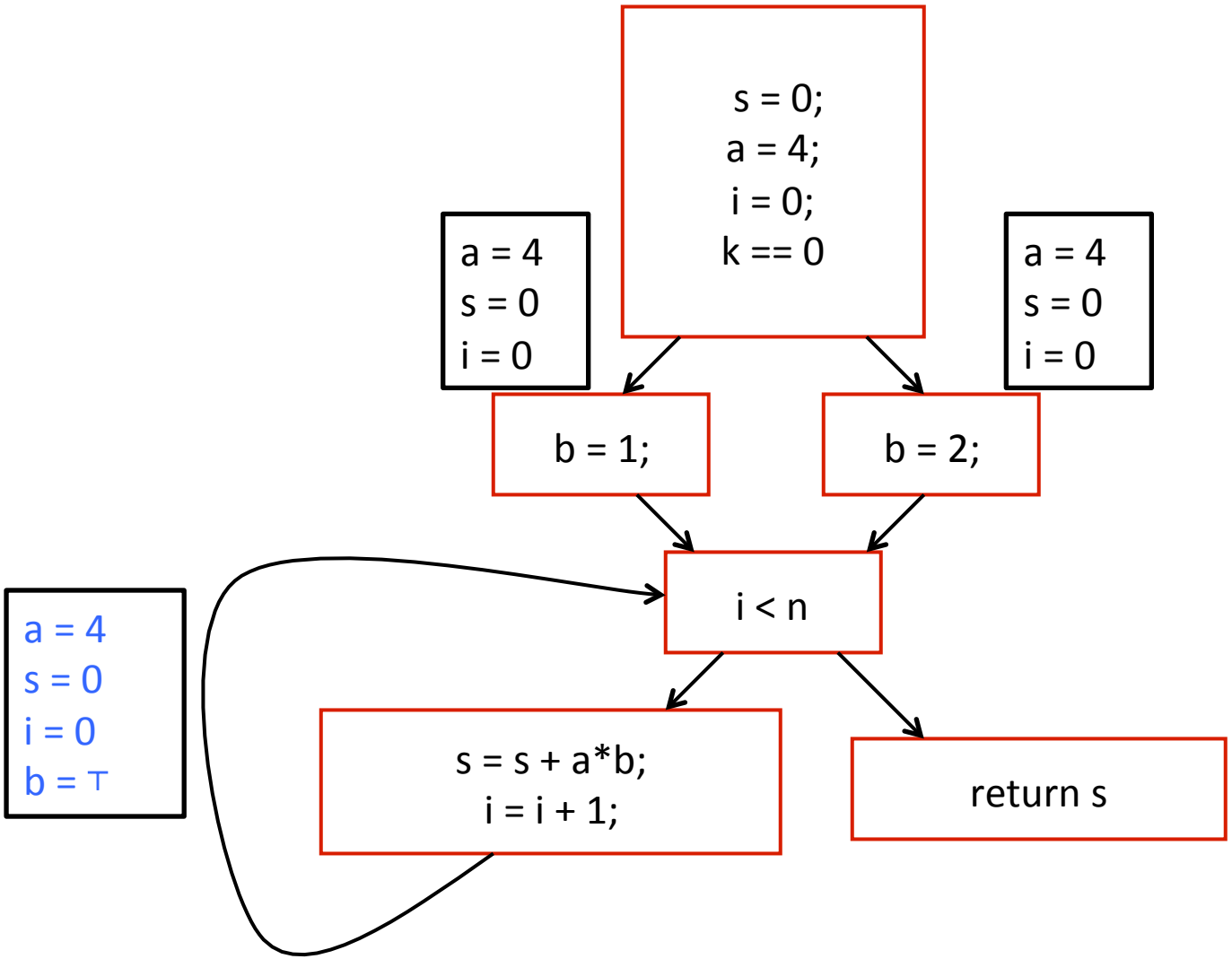
- Undefined: $\bot$
- Constant: …, -10, -9, …, 0, 1, 2, … 42, …
- NAC: $\top$

- Undefined and NAC are not the same!
  - Undefined: variable has not been initialized
  - NAC: variable definitely has a value (we just don't know what)

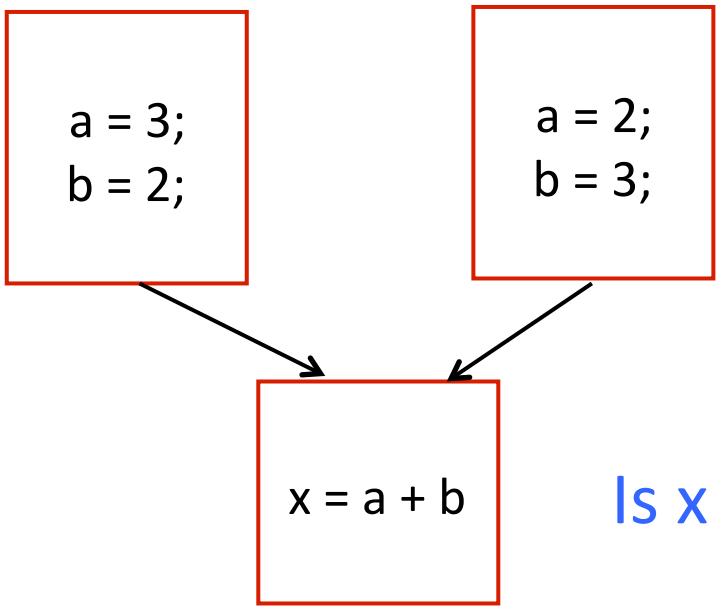
Constant Propagation: meet

- Meet rules:
  - constant $\wedge$ constant = constant (if equal)
  - constant $\wedge$ constant = $\top$ (if not equal)
  - constant $\wedge$ $\bot$ = constant
  - constant $\wedge$ $\top$ = $\top$

Constant Propagation

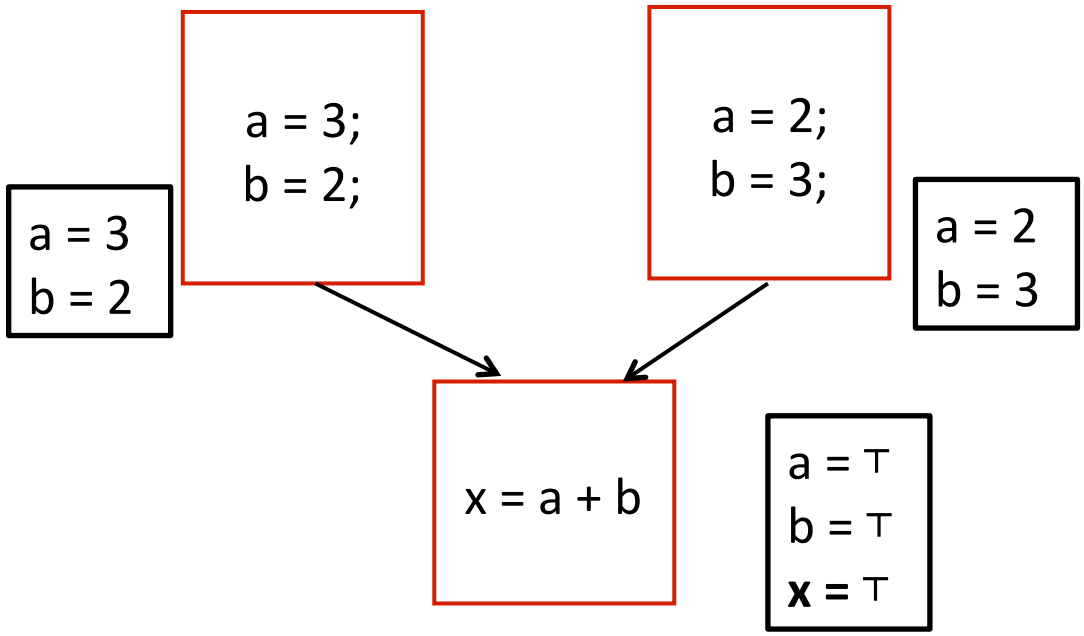


How about this?



Is x a constant?

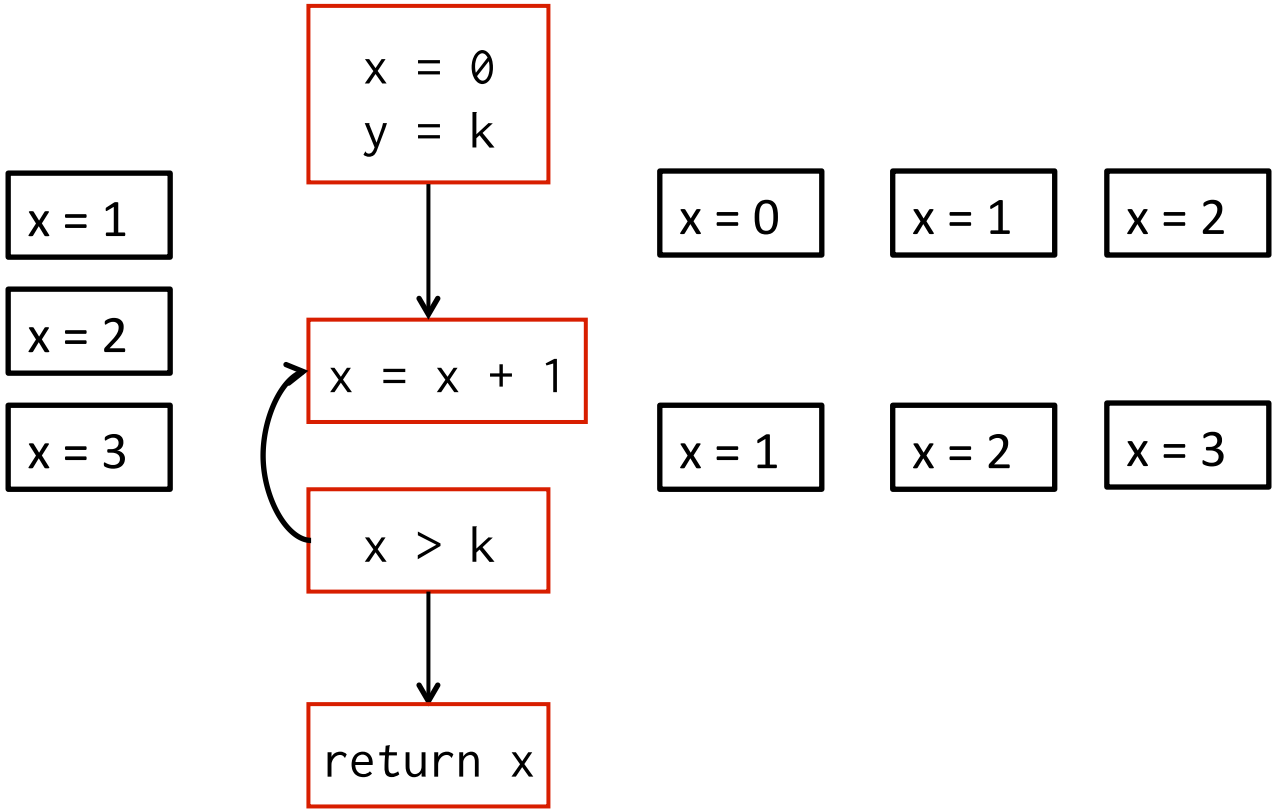
How about this?



constant ∧ constant = ⊤ (if not equal) !!

How about this?

- Undefined: $\perp$
- Constant: …, -10, -9, …, 0, 1, 2, … 42, …
- Meet: if $x \leq y$, $x \wedge y = y$ (= x otherwise)



What do we care about?

- Correctness
- Precision
- Convergence
- Runtime

Back to Marlowe and Ryder

Dataflow Framework

- <G, L, F, M>
- G = flow graph
- L = (semi-)lattice
- F = transfer function
- M = application of transfer function to nodes in G

Semi-lattice

- Three definitions:
  1. Set of values **s** and a meet operator $\wedge$ that is:
     - Idempotent: $x \wedge x = x$, for all x in **s**
     - Commutative: $x \wedge y = y \wedge x$, for all x, y in **s**
     - Associative: $x \wedge (y \wedge z) = (x \wedge y) \wedge z$

  2. Set of values **s** and a partial order with greatest lower bound for any non-empty subset:
     - Reflexive: $x \leq x$, for all x in **s**
     - Anti-symmetric: if $x \leq y$ and $y \leq x$ then $x = y$, for all x, y in **s**
     - Transitive: if $x \leq y$ and $y \leq z$ then $x \leq z$

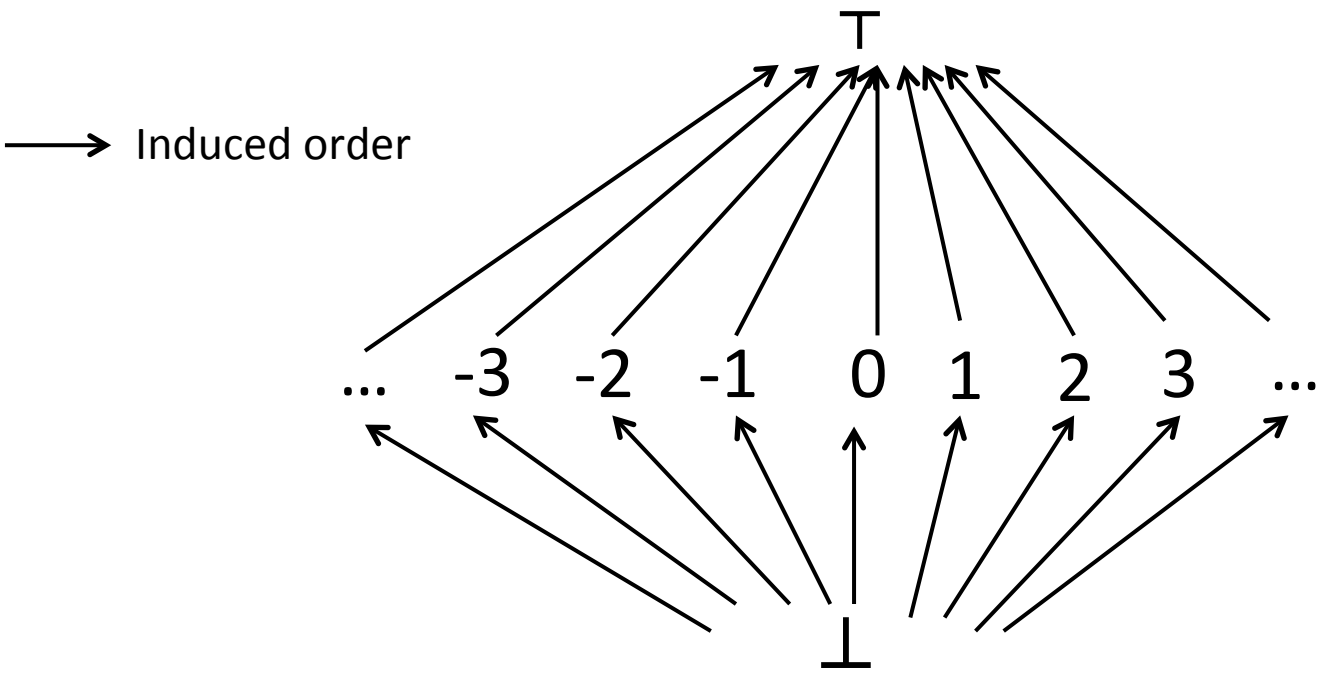
  – Check: $x \leq y \leftrightarrow x \wedge y = x$, for all x, y in **s**

  – What is **s**?

Semi-lattice

3. Graphically:

- S = { ⊥ , ⊤, **Z**}



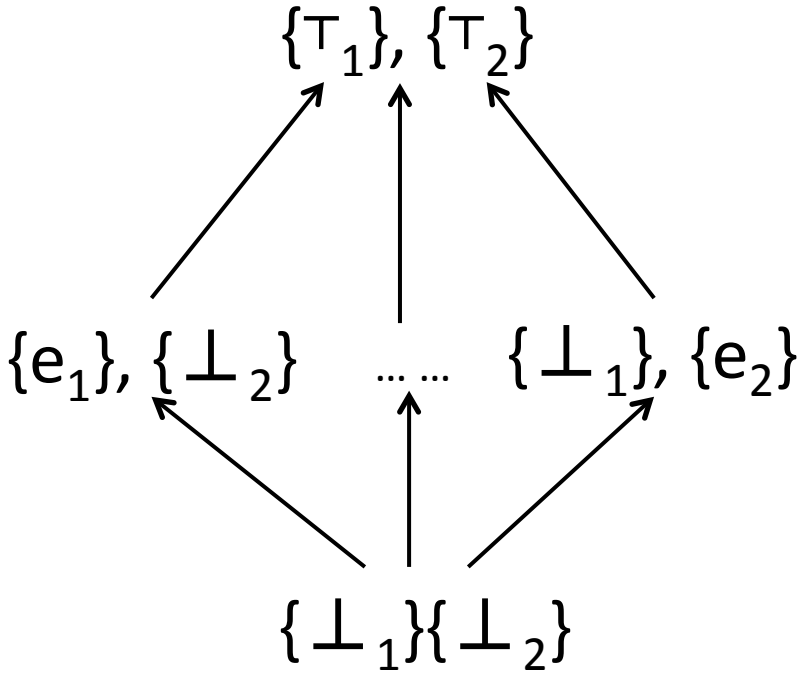
Semi-lattice

- **Meet**- semi-lattice has a greatest lower bound (GLB) for all subsets of **s**
- **Join**- semi-lattice has a least upper bound (LUB) for all subsets of **s**

- Basically comes down to partial order definition

Some Properties

- Define **product** of lattices as union
- $\mathbf{s_1}$: $\{\bot_1, \top_1, e_1\}$ $\mathbf{s_2}$: $\{\bot_2, \top_2, e_2\}$
- $\mathbf{s_1}$ x $\mathbf{s_2}$:



Useful for doing multiple analysis simultaneously

Some Properties

- Height of a lattice is the largest number of > relations that will fit in a descending chain: $x_0 > x_1 > \ldots \bot$

- We only want finite descending chains

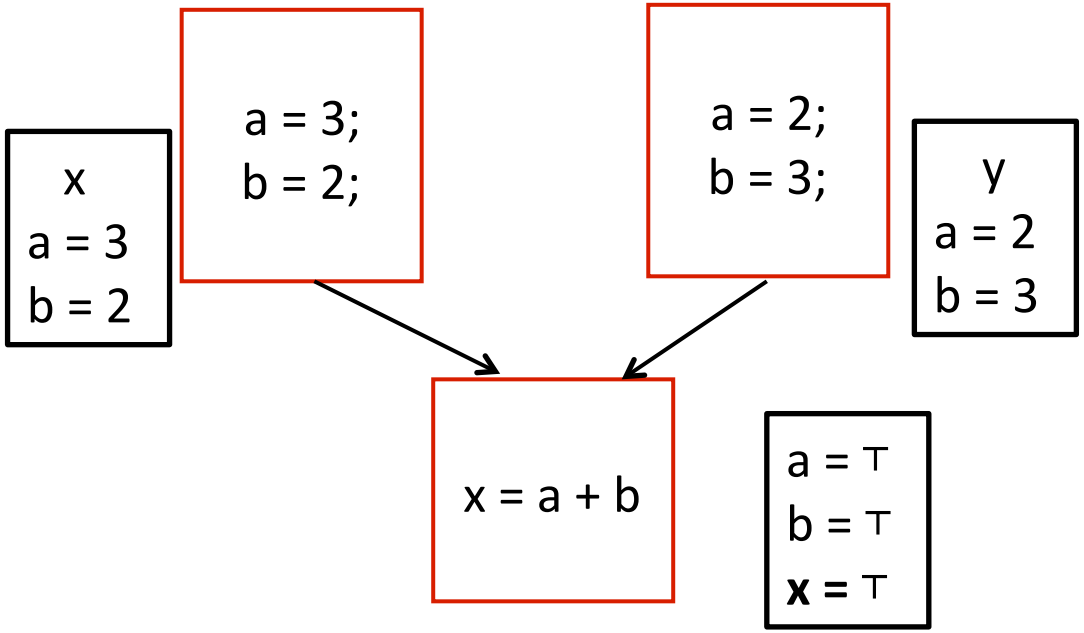
Transfer Functions

- For a family of transfer functions F
  - f: **s** → **s** for all f ∈ F

- Furthermore:
  - Identity function: ∃ f . f(x) = x for all x in **s**
  - Closed under composition:
    - $f_1, f_2 \in F, \rightarrow f_1 \bullet f_2 \in F$

Distributivity of Frameworks

- (G, L, F, M) is distributive iff
  $f(x \wedge y) = f(x) \wedge f(y)$



$f(x \wedge y)$
$= f(\{3,2,?\} \wedge \{2,3,?\})$
$= f(\{\top, \top, ?\})$
$= \{\top, \top, \top\}$

$f(x) \wedge f(y)$
$= f(\{3, 2, ?\}) \wedge f(\{2, 3, ?\})$
$= \{3, 2, 5\} \wedge \{2, 3, 5\}$
$= \{\top, \top, 5\}$

Maximal Fixed Point (MFP) Solution

- Fact: the iterative solution to dataflow equations is the most precise
- Intuition:
  - Start with the top element at each program point
  - Refine during each iteration to satisfy all dataflow equations
  - Final result will be closest to the top
- Hence for any solution FP of dataflow equations: $FP \leq MFP$

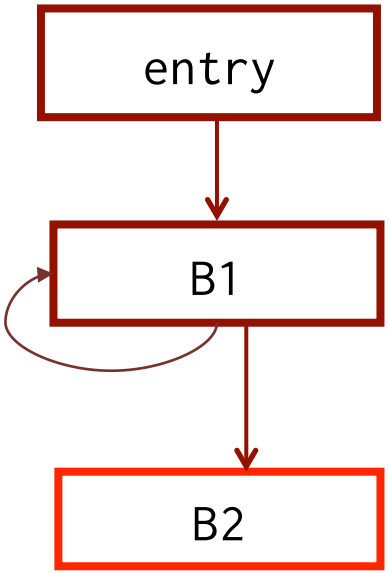
Meet Over Paths (MOP) Solution

- Another approach to solve the dataflow equations:
  - Enumerate each path p_k = [entry, n_1, n_2, …, n_k]
  - Define IN[p_k] = f_{nk-1}(… (f_{n1} (f_{n0}(d_0)))) , where d_0 is the flow element for entry
  - Compute final solution as

    IN[n] = U { IN[p] . p is a path from entry to p }

MFP and MOP

- Fact: MFP ≤ MOP
- Why not compute MOP in practice?



How many paths can reach B2?

MFP and MOP

- Fact: For transfer functions that are distributive, then MFP = MOP

- Recall: $f(x \wedge y) = f(x) \wedge f(y)$
- Hence $f(x_1) \wedge f(x_2) \wedge f(x_3) \ \ldots = f(\bigwedge x_i)$

- We can compute MOP using iterative algorithm!