Overview

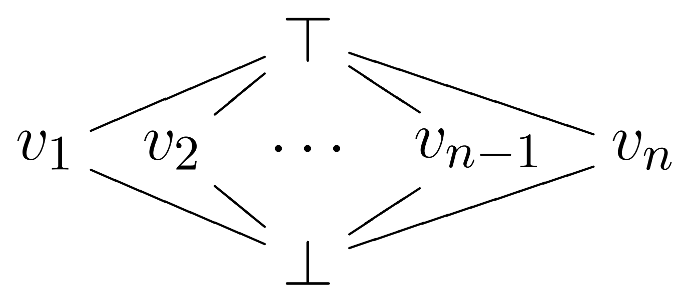
- Abstract Interpretation
  - What is it, intuitively?
  - Relationship to dataflow analysis
- Value ranges
- Fixpoints and infinite lattices
  - Dataflow problems with infinite lattices
  - Widening
  - Narrowing
- Two approaches to generating correct analyses
  - Representation functions
  - Correctness relations

Abstract Interpretation: Intuitively

- “Execute” the program on an abstract program state
  - Just like writing an interpreter, but...
  - Abstract program state represents all possible program states at a particular program point
  - Covers all possible program inputs
- What to do for multiple incoming control-flow edges? Join!
- What to do for program loops? Iterate!

Relationship to Dataflow Analysis

- Abstract interpretation is a dataflow analysis
  - A different way to construct *correct* analyses
  - Induces a specific ordering on the "worklist"
- Abstract program states are typically complete lattices
  - Trivial join lattice for any domain V with values $v_1, v_2, \cdots, v_n \in V$ implies an abstract interpretation.



  - Will permit lattices with infinite height
  - Can combine multiple analyses into a single lattice
- Trivial example: constant propagation

Generating Analyses

- Start with the values in domain V you are interested in. Example: The integers $\mathbb{Z} = \{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\}$.
- Next, consider the operations that can be performed on values in V, e.g., $+$, $-$, $*$, $/$. For $v_1, v_2 \in V$ we say that $v_1 \rightsquigarrow v_2$ if the value v_1 can be transformed to v_2.
- Determine the form of the elements in the lattice L.
- Construct the operations performed on the elements of the lattice L. For $l_1, l_2 \in V$ we say that $l_1 \triangleright l_2$ if the lattice element l_1 can be transformed to l_2.

Back to Reality: Constant Propagation

- What does the $\rightsquigarrow$ for constant propagation involve?

Back to Reality: Constant Propagation

- What does the $\rightsquigarrow$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.

Back to Reality: Constant Propagation

- What does the $\rightsquigarrow$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.
- What then does $\triangleright$ involve?

Back to Reality: Constant Propagation

- What does the $\leadsto$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.
- What then does $\rhd$ involve?
- Negation, addition, subtraction, multiplication, etc., of elements in the lattice.

Back to Reality: Constant Propagation

- What does the $\leadsto$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.
- What then does $\triangleright$ involve?
- Negation, addition, subtraction, multiplication, etc., of elements in the lattice.
- For negation, the following hold:

$$-(\top) \triangleright \top \qquad -(\bot) \triangleright \bot \qquad -(c) \triangleright -c$$

Back to Reality: Constant Propagation

- What does the $\leadsto$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.
- What then does $\rhd$ involve?
- Negation, addition, subtraction, multiplication, etc., of elements in the lattice.
- For negation, the following hold:

$$-(\top) \rhd \top \qquad -(\bot) \rhd \bot \qquad -(c) \rhd -c$$

- Binary operations will have, e.g., $l_1 \times l_2 \rhd l_3$.

Back to Reality: Constant Propagation

- What does the $\leadsto$ for constant propagation involve?
- Negation, addition, subtraction, multiplication, etc., of integers.
- What then does $\triangleright$ involve?
- Negation, addition, subtraction, multiplication, etc., of elements in the lattice.
- For negation, the following hold:

$$-(\top) \triangleright \top \qquad -(\bot) \triangleright \bot \qquad -(c) \triangleright -c$$

- Binary operations will have, e.g., $l_1 \times l_2 \triangleright l_3$.
  - What would $+$ look like?

Value Ranges

- Constant propagation is boring: we can do better.
- **Definition**: A *value range*, denoted $[a : b]$, represents all values x such that:

$$a \in \mathbb{Z} \cup \{-\infty\} \qquad b \in \mathbb{Z} \cup \{\infty\} \qquad a \leq x \leq b$$

- Examples:
  - $[17 : 17]$ represents the value 17.
  - $[17 : 42]$ represents any value between 17 and 42.
  - $[-\infty : -1]$ represents any negative integer.
  - $[0 : \infty]$ represents any non-negative integer.
- Is this representation more or less expressive than in constant propagation?

Value Range Lattice

- To define a lattice, we need:

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.
- $\top = [-\infty : \infty]$. Why?

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.
- $\top = [-\infty : \infty]$. Why?
- What does $\perp$ mean? Denote it by $\perp = [\infty : -\infty]$.

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.
- $\top = [-\infty : \infty]$. Why?
- What does $\perp$ mean? Denote it by $\perp = [\infty : -\infty]$.
- **Definition**: $[a_1 : b_1] \sqcup [a_2 : b_2] = [\min(a_1, a_2) : \max(b_1, b_2)]$

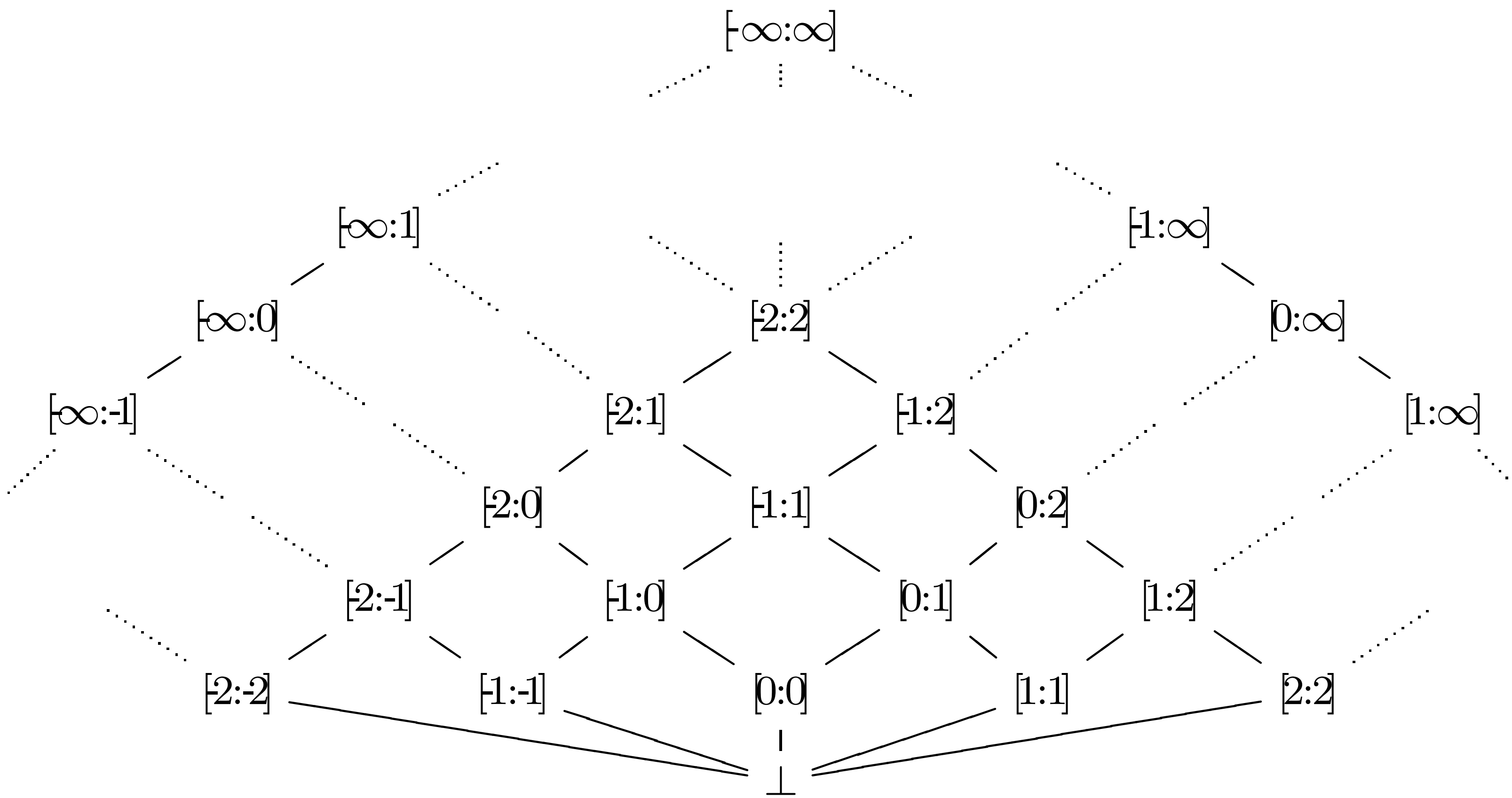
Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.
- $\top = [-\infty : \infty]$. Why?
- What does $\bot$ mean? Denote it by $\bot = [\infty : -\infty]$.
- **Definition**: $[a_1 : b_1] \sqcup [a_2 : b_2] = [\min(a_1, a_2) : \max(b_1, b_2)]$
- **Definition**: $[a_1 : b_1] \sqcap [a_2 : b_2] = [\max(a_1, a_2) : \min(b_1, b_2)]$

Value Range Lattice

- To define a lattice, we need:
  - A partial ordering relation $\sqsubseteq$.
  - A join operator $\sqcup$.
  - A meet operator $\sqcap$.
- **Definition**: $[a_1 : b_1] \sqsubseteq [a_2 : b_2]$ when $a_1 \geq a_2 \wedge b_1 \leq b_2$.
- $\top = [-\infty : \infty]$. Why?
- What does $\perp$ mean? Denote it by $\perp = [\infty : -\infty]$.
- **Definition**: $[a_1 : b_1] \sqcup [a_2 : b_2] = [\min(a_1, a_2) : \max(b_1, b_2)]$
- **Definition**: $[a_1 : b_1] \sqcap [a_2 : b_2] = [\max(a_1, a_2) : \min(b_1, b_2)]$
- How wide is this lattice? How high?

Value Range Lattice: Graphically

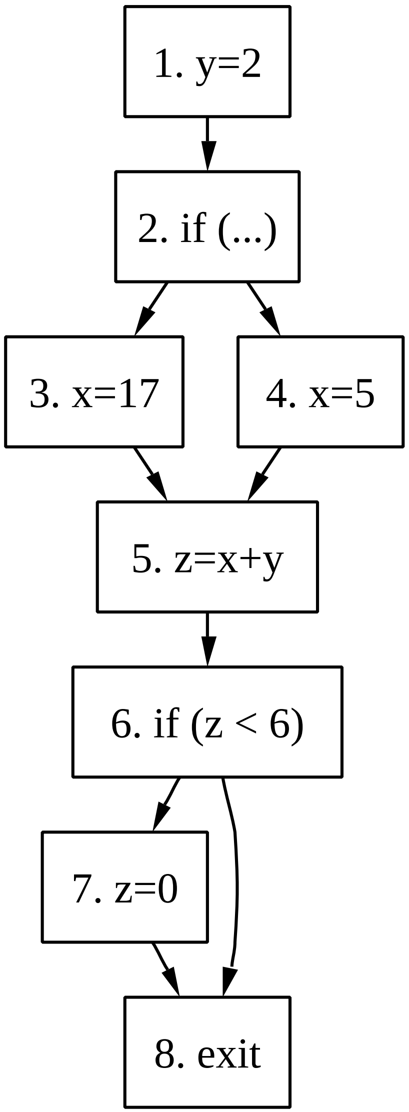


(Marvel at it. It took me *forever* to get right.)

Value Range Operations

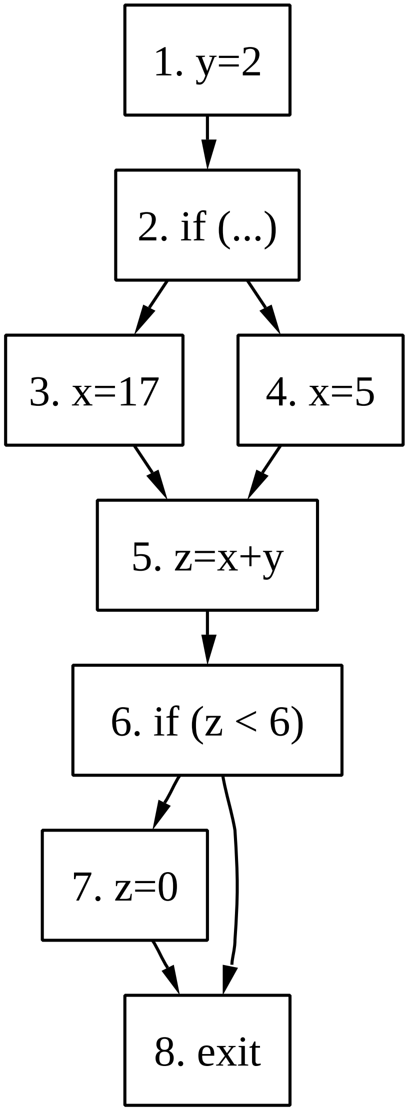
- Negation: $-[a : b] \rhd [-b : -a]$.
- Addition: $[a_1 : b_1] + [a_2 : b_2] \rhd [a_1 + a_2 : b_1 + b_2]$
- Subtraction: $[a_1 : b_1] - [a_2 : b_2] \rhd [a_1 - b_2 : b_1 - a_2]$
- Multiplication: $[a_1 : b_1] \cdot [a_2 : b_2] \rhd$ $[\min(a_1a_2, a_1b_2, b_1a_2, b_1b_2) : \max(a_1a_2, a_1b_2, b_1a_2, b_1b_2)]$
- Key points to revisit later:
  - We know how to map from elements (integers) in V to elements (value ranges) in L.
  - We can prove that the operations on elements of V are “abstracted” by the operations on elements on L. Important relationship between $\rightsquigarrow$ and $\rhd$.
- But now, let’s try some abstract interpretation...

Abstract Interpretation Example



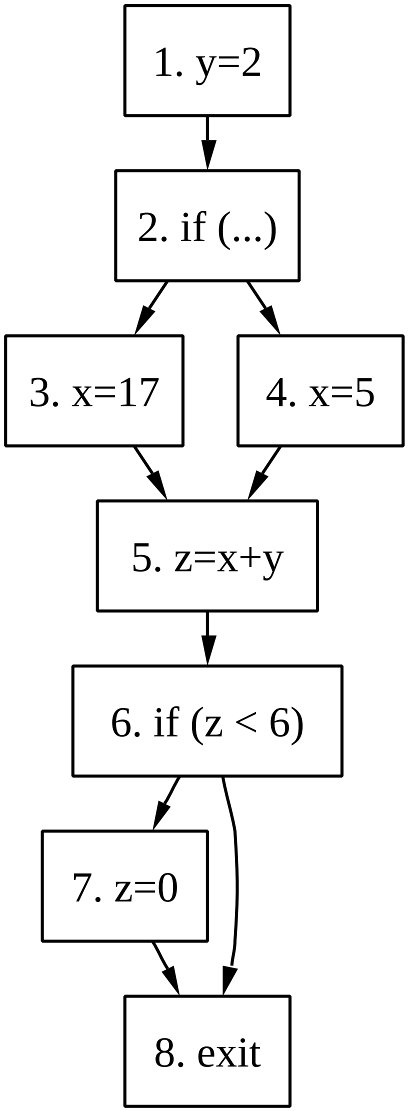
- Example: Try it with constant propagation lattice.
  - Not much of an improvement.
- Example: Try it with value range lattice.
  - Start at entry node.

Abstract Interpretation Example



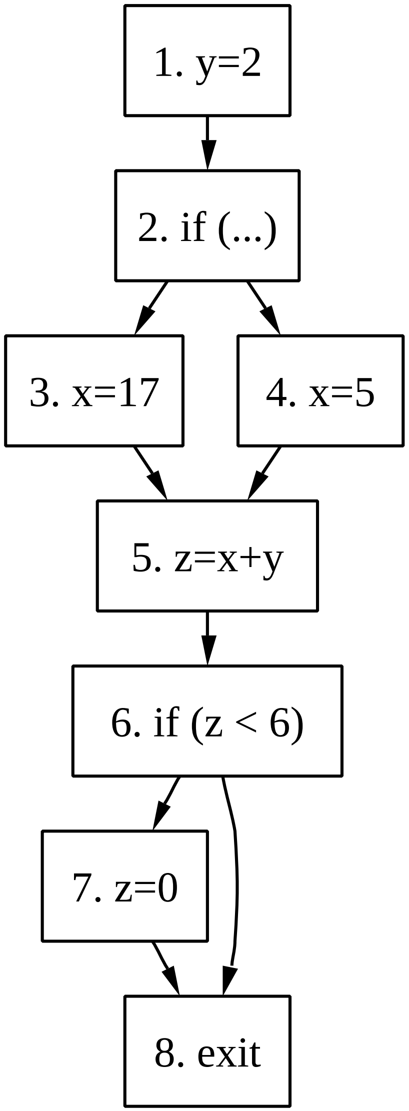
- Example: Try it with constant propagation lattice.
  - Not much of an improvement.
- Example: Try it with value range lattice.
  - Start at entry node.
  - Apply $\sqcup$ at control-flow joins

Abstract Interpretation Example



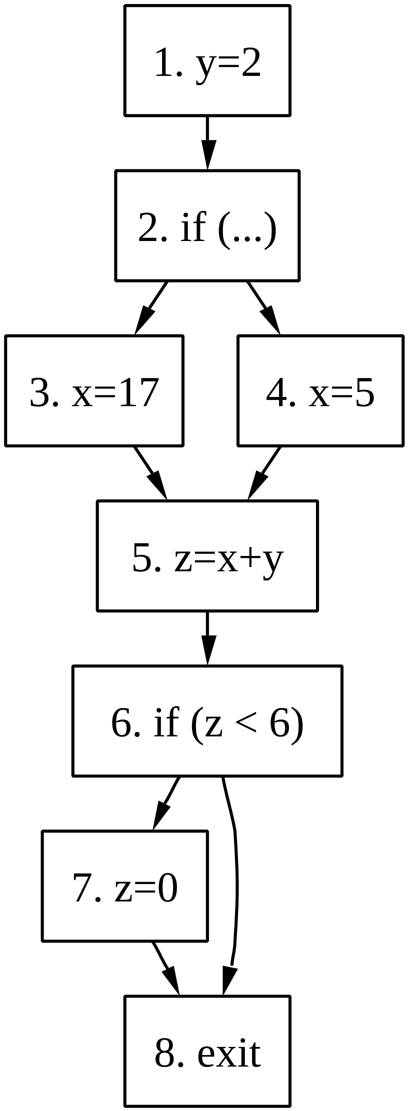
- Example: Try it with constant propagation lattice.
  - Not much of an improvement.
- Example: Try it with value range lattice.
  - Start at entry node.
  - Apply $\sqcup$ at control-flow joins
  - Apply $\triangleright$ for each operation.

Abstract Interpretation Example



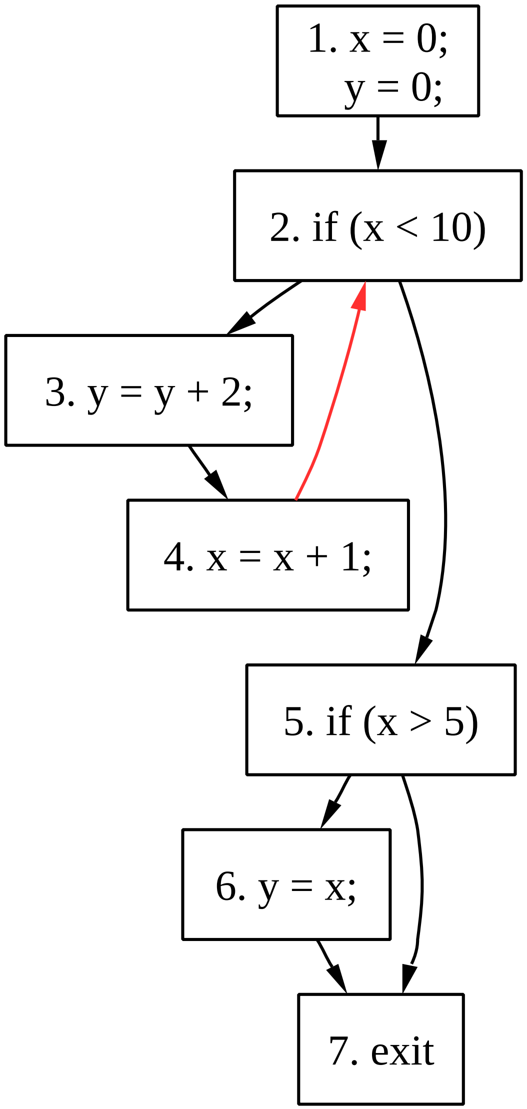
- Example: Try it with constant propagation lattice.
  - Not much of an improvement.
- Example: Try it with value range lattice.
  - Start at entry node.
  - Apply $\sqcup$ at control-flow joins
  - Apply $\triangleright$ for each operation.
  - Note: Introducing $<$ into $\triangleright$ improves analysis.

Abstract Interpretation Example



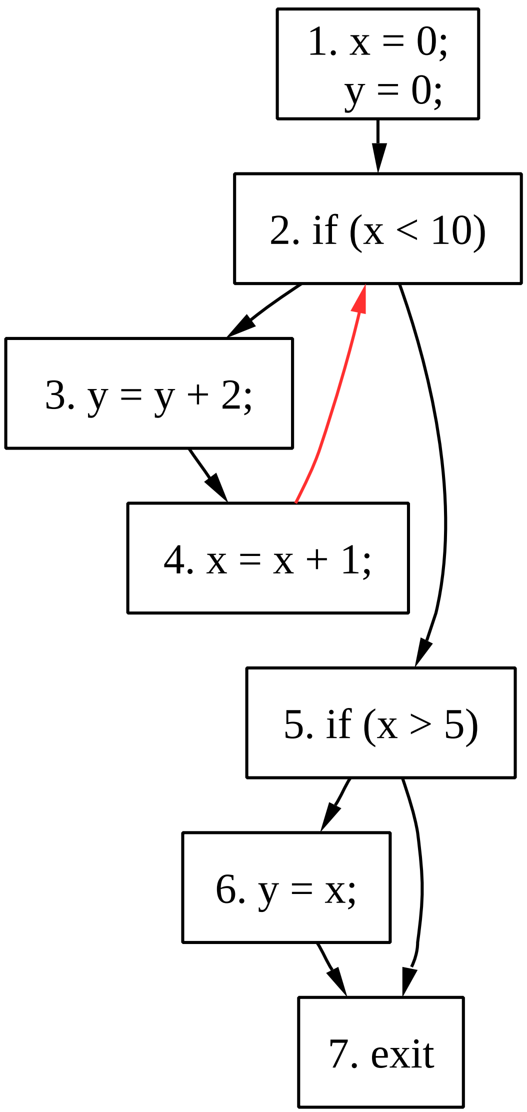
- Example: Try it with constant propagation lattice.
  - Not much of an improvement.
- Example: Try it with value range lattice.
  - Start at entry node.
  - Apply $\sqcup$ at control-flow joins
  - Apply $\triangleright$ for each operation.
  - Note: Introducing $<$ into $\triangleright$ improves analysis.

Analyzing Loops



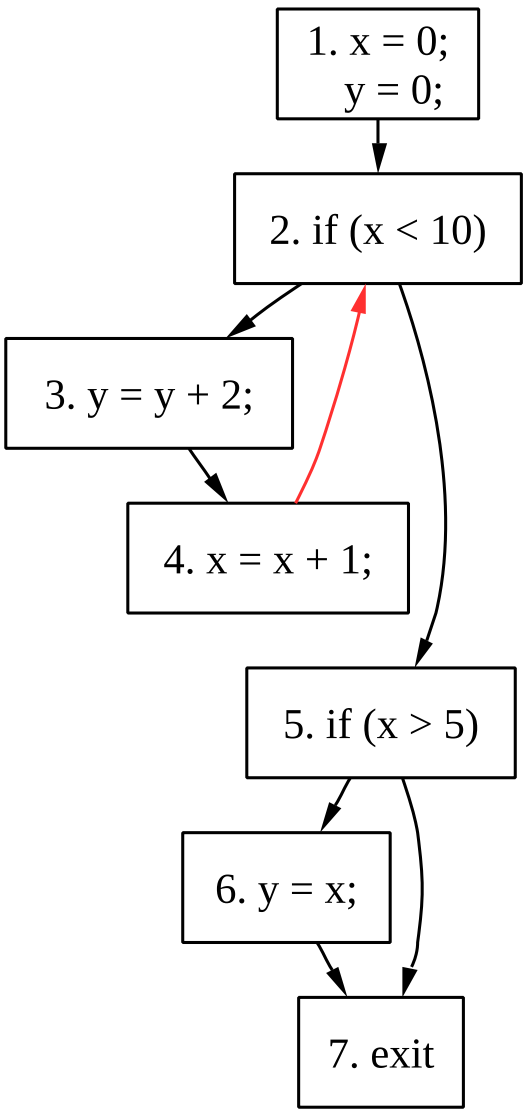
- What do we do at node 2? Join with $\perp$ (as in dataflow analysis).

Analyzing Loops



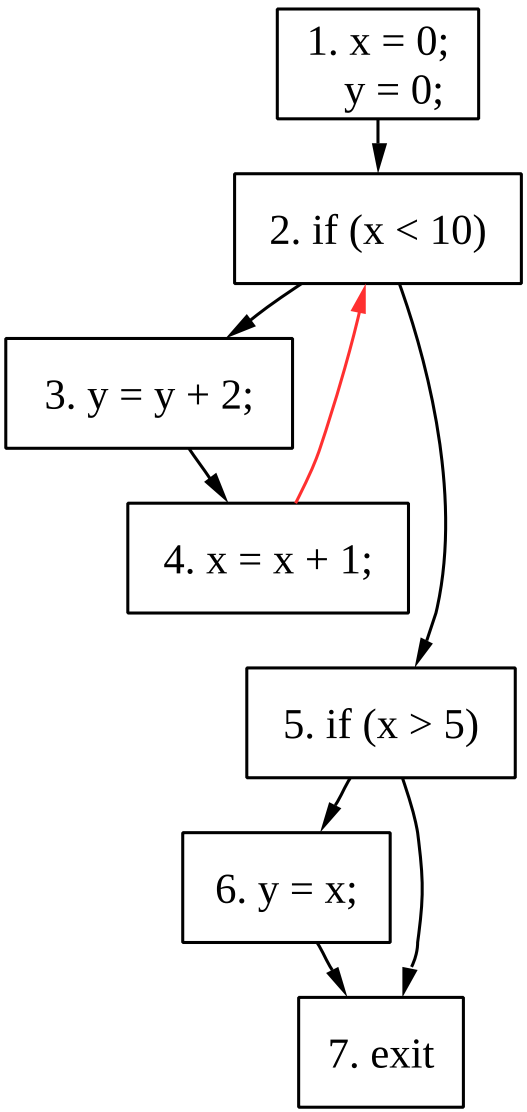
- What do we do at node 2? Join with $\bot$ (as in dataflow analysis).
- What do we do at the back edge from 2 to 4? Iterate around this loop until it stabilizes.

Analyzing Loops



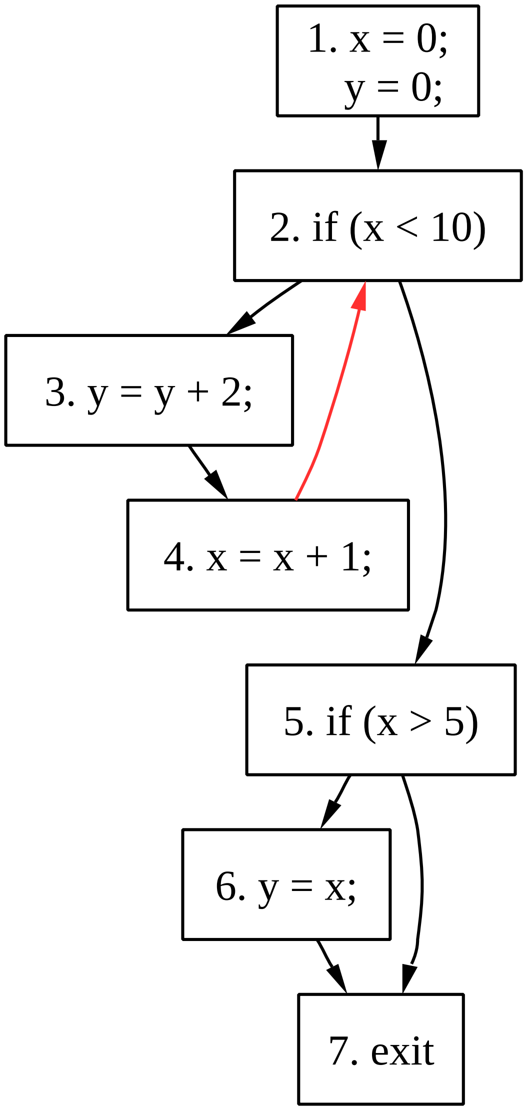
- What do we do at node 2? Join with $\perp$ (as in dataflow analysis).
- What do we do at the back edge from 2 to 4? Iterate around this loop until it stabilizes.
- Does it every stabilize?

Analyzing Loops



- What do we do at node 2? Join with $\perp$ (as in dataflow analysis).
- What do we do at the back edge from 2 to 4? Iterate around this loop until it stabilizes.
- Does it every stabilize?
- Need to introduce *widening*: jumps values closer to $\top$ on back edges.

Analyzing Loops



- What do we do at node 2? Join with $\bot$ (as in dataflow analysis).
- What do we do at the back edge from 2 to 4? Iterate around this loop until it stabilizes.
- Does it every stabilize?
- Need to introduce *widening*: jumps values closer to $\top$ on back edges.

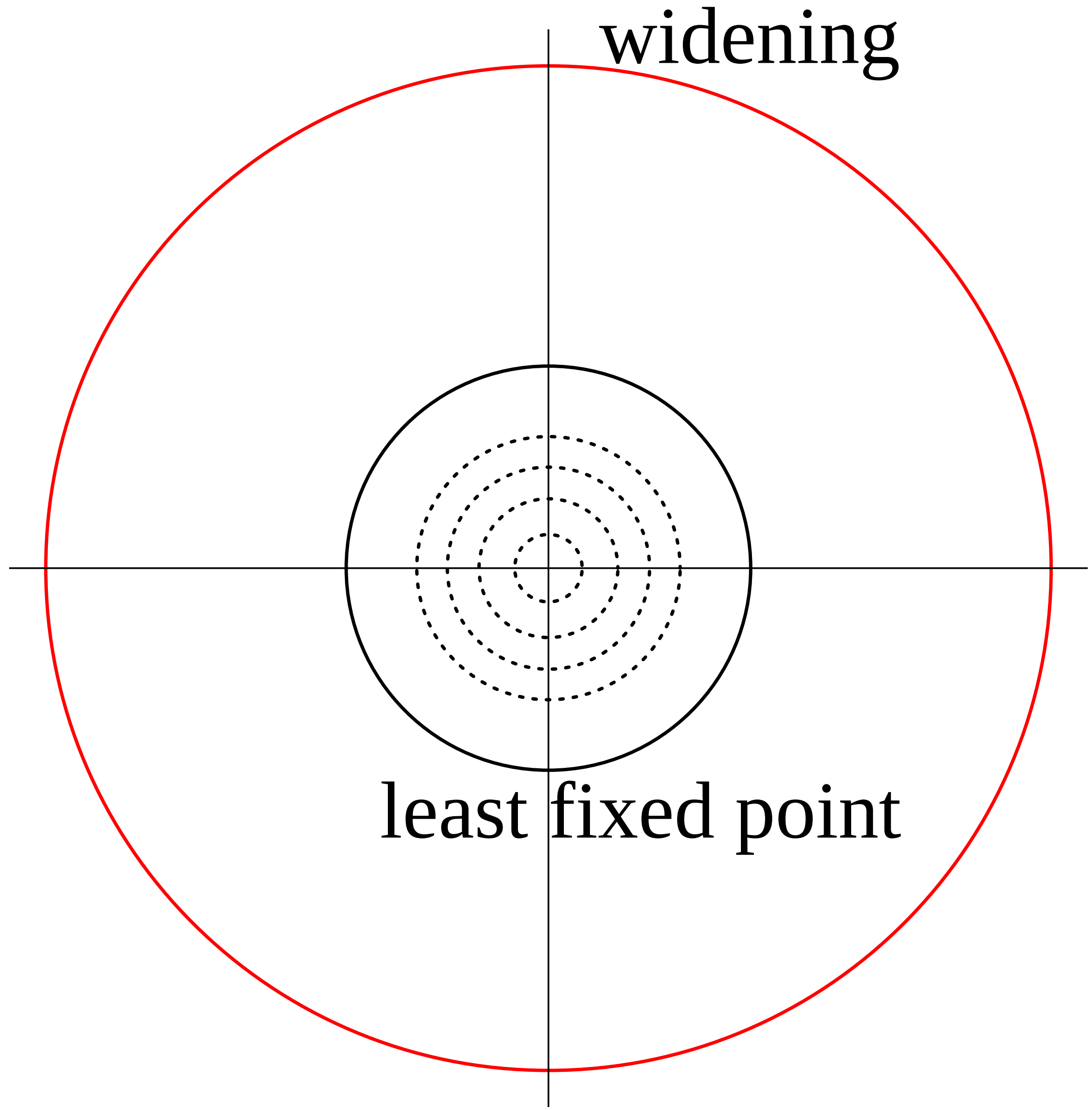
Widening

- *Widening* reduces the number of iterations around a loop to a finite quantity, even in an infinite lattice.
- Formally, $\nabla : L \times L \rightarrow L$ is a widening operator iff:
  - It is an upper bound operator, such that $\forall l_1, l_2 \in V \qquad l_1 \sqsubseteq (l_1 \nabla l_2) \sqsupseteq l_2$.
  - For all ascending chains of lattice elements $l_1, l_2, \cdots$, the ascending chain $l_1 \nabla l_2 \nabla l_3 \nabla \cdots$ stabilizes.
- Widening operator for value ranges:

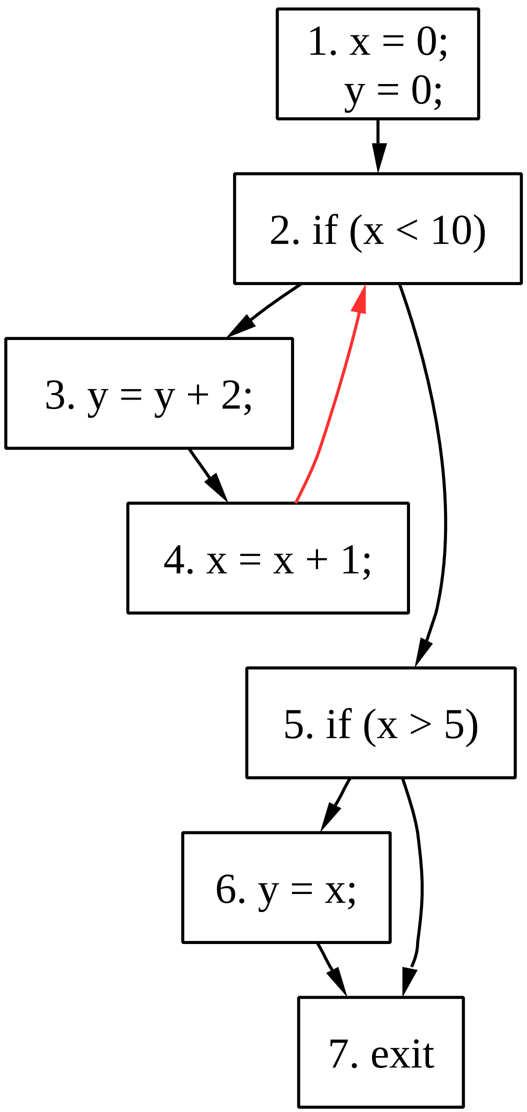
$$[a_1 : b_1] \nabla [a_2 : b_2] = [LB(a_1, a_2) : UB(b_1, b_2)]$$

$$LB(a_1, a_2) = \begin{cases} a_1 & \text{if } a_1 \leq a_2 \\ -\infty & \text{otherwise} \end{cases} \quad UB(b_1, b_2) = \begin{cases} b_1 & \text{if } b_1 \geq b_2 \\ \infty & \text{otherwise} \end{cases}$$

Widening: Graphically



Applying Widening Operators



- Apply $l_1 \nabla l_2$ on back edges. l_1 is the previous value (at the head of the edge) and l_2 is the new value (at the tail of the edge).
- Now we get a fixed point even with our infinite lattice.
- Let's look at x:
  1. $[0:0] \nabla ([0:0] \sqcup [1:1]) = [0:\infty]$.
  2. $[0:\infty] \nabla ([0:0] \sqcup [1:\infty]) = [0:\infty]$.

Deriving Information from Conditions

- Condition `if (`$x < 10$`)` tells us something about the value of x in the `then` and `else` branches.
- If true, we know that $x \in [-\infty : 9]$. If false, $x \in [10 : \infty]$.
- This information is *in addition to* what we already knew.
  - Meet operation $l_1 \sqcap l_2$ computes the lattice element when both l_1 and l_2 describe the value.
  - What if the meet is $\perp$?
- Example: We know that $x \in [0 : \infty]$ (magically).
  - On `then` branch, $x \in ([0 : \infty] \sqcap [-\infty : 9]) = [0 : 9]$.
  - On the `else` branch, $x \in ([0 : \infty] \sqcap [10 : \infty] = [10 : \infty]$.

Narrowing

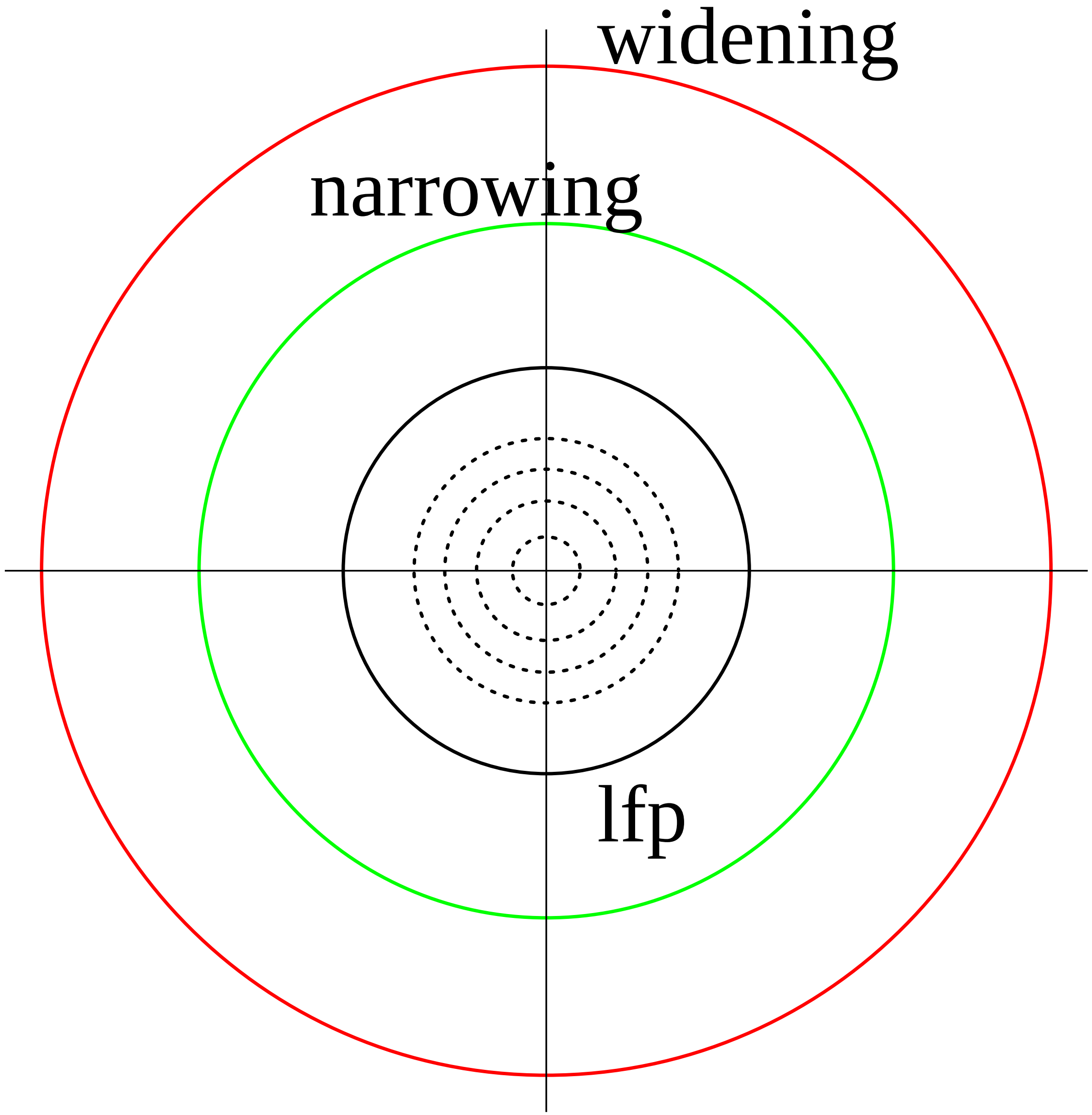
- Apply narrowing after widening to recover some information lost due to widening.
- $\triangle : L \times L \rightarrow L$ is a *narrowing* operator if:
  - $\forall l_1, l_2 \in L \quad l_2 \sqsubseteq (l_1 \triangle l_2) \sqsubseteq l_1$, and
  - For all descending chains of lattice elements $l_1, l_2, \cdots$, the descending chain $l_1 \triangle l_2 \triangle l_3 \triangle \cdots$ stabilizes.
- Narrowing operator for value ranges:

$$[a_1 : b_1] \triangle [a_2 : b_2] = [z_1 : z_2]$$

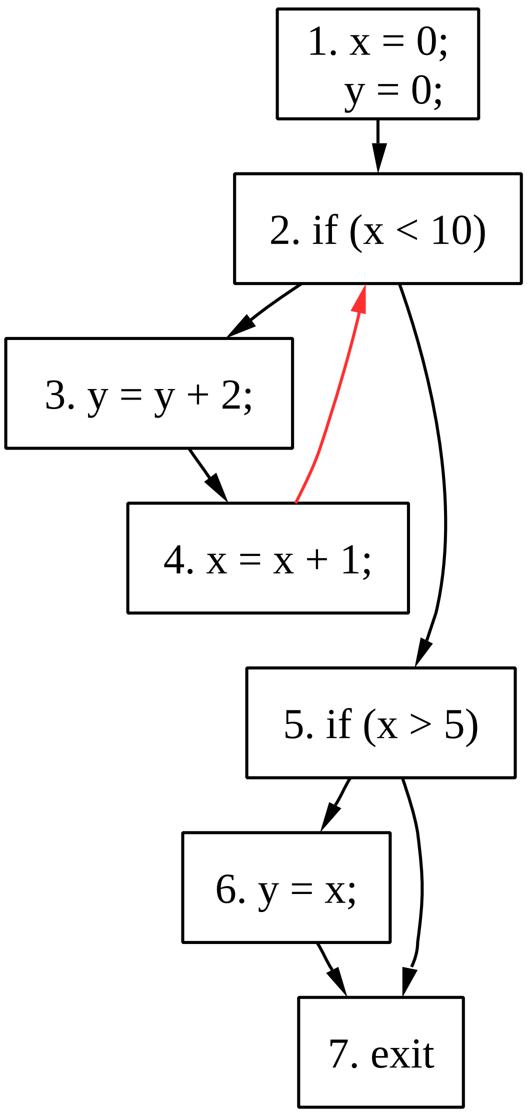
where $z_1 =$ if $a_1 = -\infty$ then a_2 else a_1,

$z_2 =$ if $b_1 = \infty$ then b_2 else b_1

Narrowing: Graphically



Widening/Narrowing Example



- $K = \{0, 1, 2, 5, 10\}$
- Let's look at x again:
  1. $[0 : 0] \nabla ([0 : 0] \sqcup [1 : 1]) = [0 : \infty]$.
  2. $[0 : \infty] \nabla ([0 : 0] \sqcup [1 : 10]) = [0 : \infty]$. *Stable.*
  3. $[0 : \infty] \triangle [0 : 10] = [0 : 10]$. (Interpret the loop)
  4. $[0 : 10] \triangle ([0 : 0] \sqcup [1 : 10]) = [0 : 10]$. *Stable.*
- Now, $x \in [0 : 9]$ on `then` branch, $x \in [10 : 10]$ on `else` branch!

A Better Widening Operator

- Let K be the set of integer constants in the program.
- Define ∇ as:

$$[a_1 : b_1] \nabla [a_2 : b_2] = [LB(a_1, a_2) : UB(b_1, b_2)]$$

$$LB(a_1, a_2) = \begin{cases} a_1 & \text{if } a_1 \leq a_2 \\ k & \text{if } a_2 < a_1 \wedge k = \max\{k \in K | k \leq a_2\} \\ -\infty & \text{if } a_2 < a_1 \wedge \forall k \in K : a_2 < k \end{cases}$$

$$UB(b_1, b_2) = \begin{cases} b_1 & \text{if } b_1 \geq b_2 \\ k & \text{if } b_1 < b_2 \wedge k = \min\{k \in K | b_2 \leq k\} \\ -\infty & \text{if } b_1 < b_2 \wedge \forall k \in K : k < b_2 \end{cases}$$

- Precision/efficiency tradeoff: more steps, but better results.

Generating Correct Analyses

- Have shown how we can create an analysis by abstraction:
  - Abstract the value domain V with the lattice L
  - Abstract all operations (collectively called $\rightsquigarrow$) with $\triangleright$.
- How do we prove that our analysis is correct?
  - Representation functions
  - Correctness relations
- Both methods are equivalent.

Representation Functions

- Let $\beta : V \to L$ be a function that maps any value in V to its “best” representation in L.
- Your analysis is correct if the following is true:

$$\beta(v_1) \sqsubseteq l_1 \wedge v_1 \leadsto v_2 \wedge l_1 \rhd l_2 \Rightarrow \beta(v_2) \sqsubseteq l_2$$

- Intuitively: If a value can be safely described by a lattice element, then any value it is transformed into can be safely described by the corresponding transformation on the lattice element.
- Can we prove this for value ranges?

Correctness relations

- Let $R : V \times L \rightarrow \{\text{true}, \text{false}\}$ be a *correctness* relation.
- Given $v \in V, l \in L$, $v \; R \; l$ is true when v is described by l. $1R[-1:2] = ?$, $7R[17:42] = ?$
- General requirement: preservation of correctness

$$v_1 \; R \; l_1 \wedge v_1 \rightsquigarrow v_2 \wedge l_1 \rhd l_2 \Rightarrow v_2 \; R \; l_2$$

- Two more conditions for correctness when dealing with lattices:
  1. Lattice preserves R: $v \; R \; l_1 \wedge l_1 \sqsubseteq l_2 \Rightarrow v \; R \; l_2$
  2. There is always a “best” approximation l for every v: $(\forall l \in L' \subseteq L : v \; R \; l) \Rightarrow vR(\bigsqcap L')$
- Interesting consequence: $v \; R \; l_1 \wedge v \; R \; l_2 \Rightarrow vR(l_1 \sqcap l_2)$

Combining Analyses

- We mainly talk about a lattice L for values of a single variable.
- Can take the Cartesian product of several of these lattices to handle multiple variables: $L' = L_1 \times L_2 \times ... \times L_N$.
- Variables do not need to be of the same type: L_1 could be a value range lattice, L_2 a boolean lattice, and L_3 a points-to graph lattice.

Abstract Interpretation Tidbits

- You can read about Galois connections to abstract interpretation in the class text, but it will hurt.
- We've only discussed forward semantics: you can do abstract interpretation backwards, and with meet lattices (everything is dual).
- We only handled the "trivial" case of widening on back edges.
  - What to do about irreducible control-flow graphs?
  - So long as you pick widening edges such that every cycle contains at least one widening edge, abstract interpretation "works".
  - Bourdoncle studied these *chaotic* iteration strategies. NP-complete problem, but with good heuristics.

Uses of Value Range Propagation

- Constant propagation, dead-code elimination, etc: can propagate constants and determine when conditions evaluate true or false.
- Array bounds analysis: detect bugs or remove checks that are known to be unnecessary.
- Bit width estimation: limit the sizes of registers when performing hardware synthesis.
- Static branch prediction: produce probabilities that particular branches will be taken.