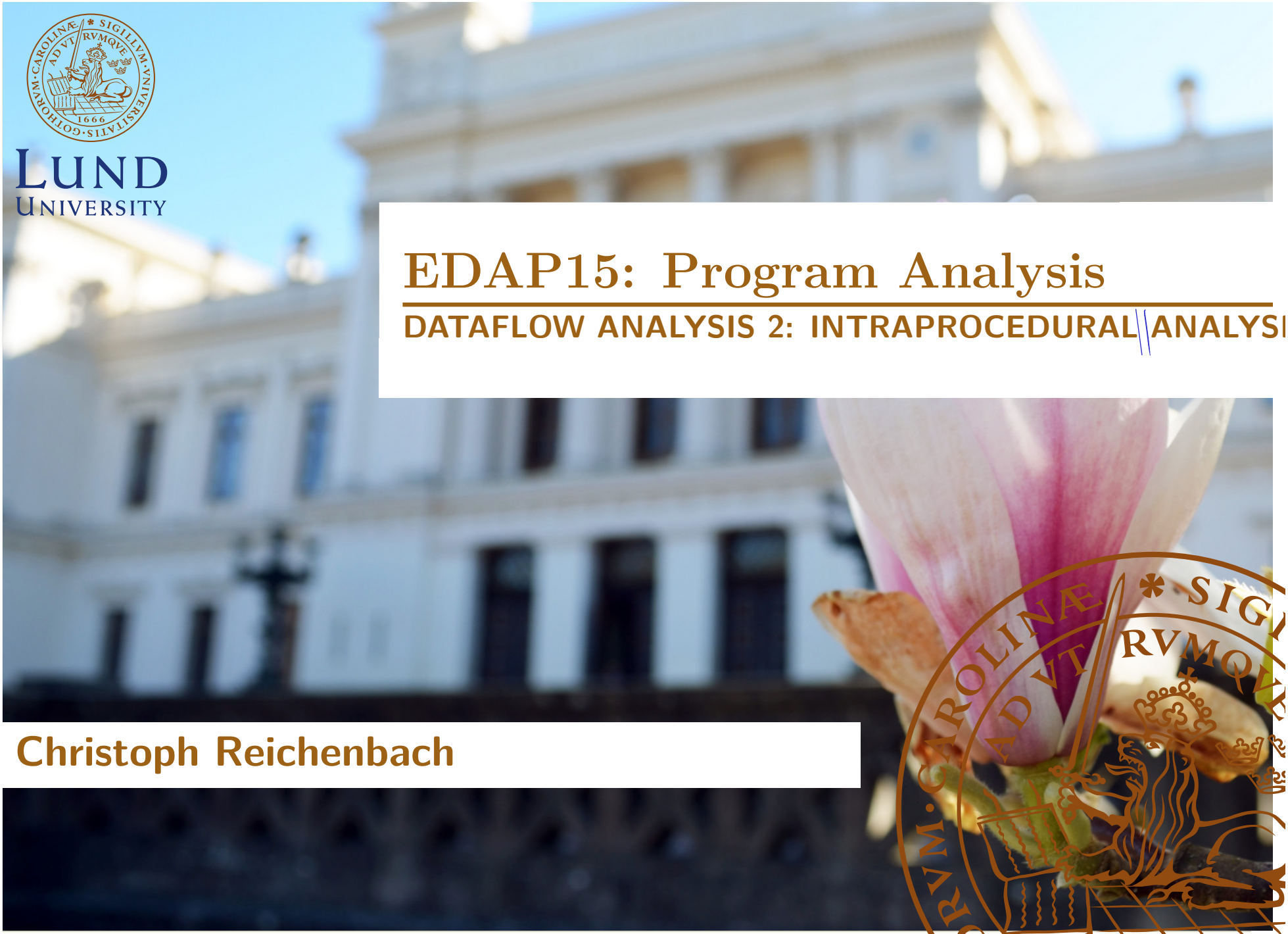


EDAP15: Program Analysis

DATAFLOW ANALYSIS 2: INTRAPROCEDURAL ANALYSI

Christoph Reichenbach

Data Flow Analysis on CFGs

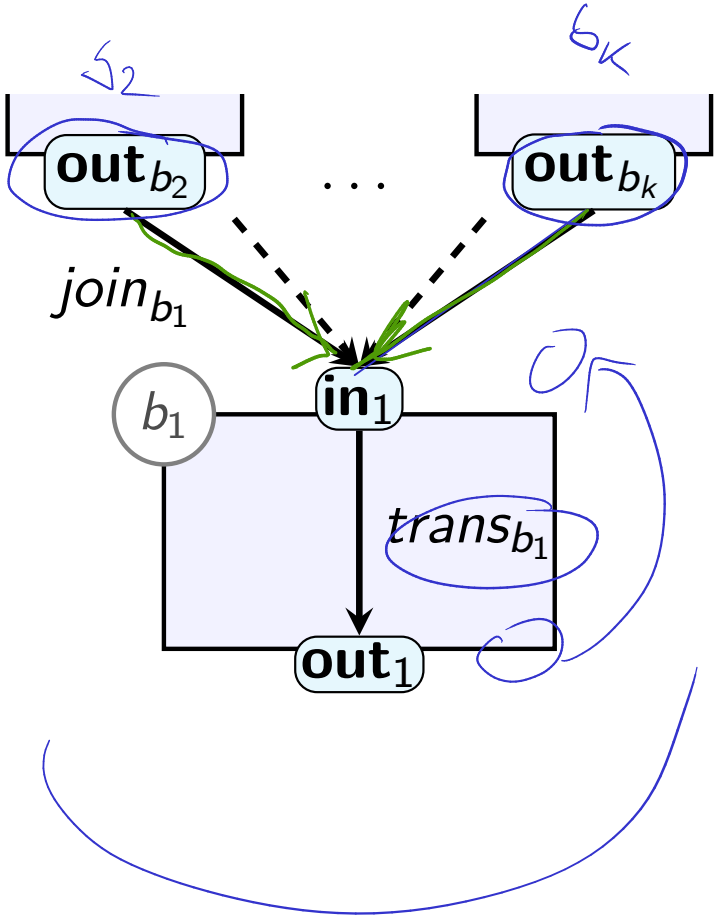
- $join_b$: *Join Function*
- $trans_b$: *Transfer Function*
- $\mathbf{in}_b$:

$$\mathbf{in}_{b_1} = join_{b_1}(\mathbf{out}_{b_2}, \ldots, \mathbf{out}_{b_k})$$

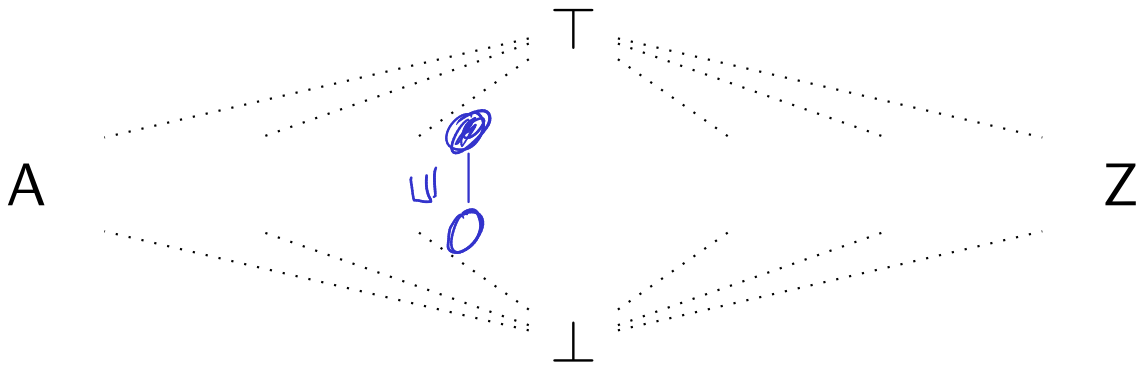
- $\mathbf{out}_b$:

$$\mathbf{out}_{b_1} = trans_{b_1}(\mathbf{in}_{b_1})$$

- *Forward Analysis*
- *Bakward Analysis*

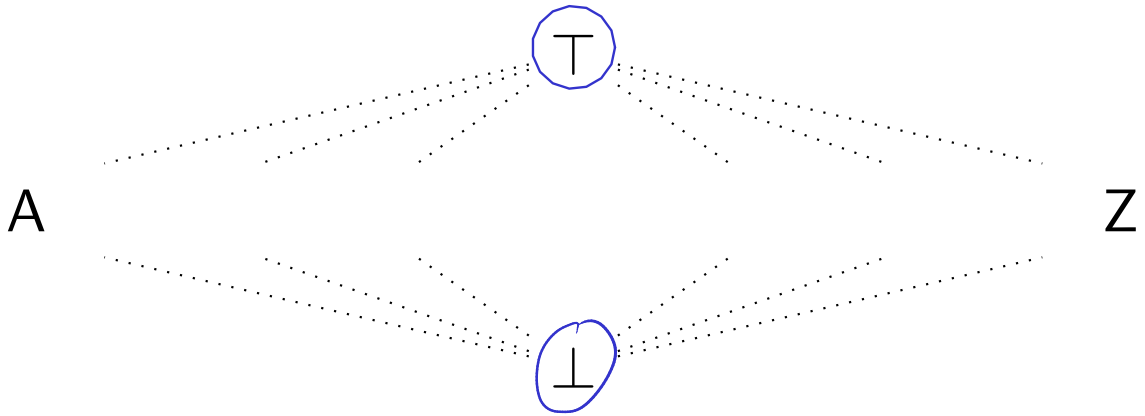


Join and Transfer Functions



- L: Abstract Domain
  - Ordered by $(\sqsubseteq) \subseteq L \times L$

Join and Transfer Functions

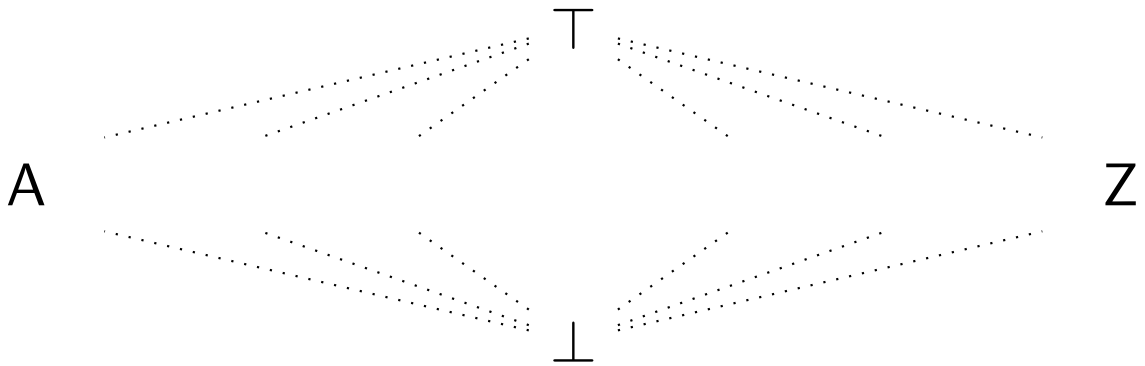


- L: Abstract Domain
  - Ordered by $(\sqsubseteq) \subseteq L \times L$

    $\top \in L$ for all x : $x \sqsubseteq \top$ Top element

    $\bot \in L$ for all x : $\bot \sqsubseteq x$ Bottom element (optional)

Join and Transfer Functions



- L: Abstract Domain
  - Ordered by $(\sqsubseteq) \subseteq L \times L$

    $\top \in L$ for all x : $x \sqsubseteq \top$ Top element

    $\bot \in L$ for all x : $\bot \sqsubseteq x$ Bottom element (optional)
- $trans_b : L \to L$
  - monotonic
- $join_b : \underline{L \times \ldots \times L} \to L$
  - pointwise monotonic

$$x \sqsubseteq y$$
$$\Downarrow$$
$$trans_b(x) \sqsubseteq trans_b(y)$$

$$x \sqsubseteq y$$
$$\Downarrow$$
$$join_b(z_1, \ldots, z_k, x, \ldots, z_n) \sqsubseteq \overset{\text{join}}{\cancel{trans}}_b(z_1, \ldots, z_k, y, \ldots, z_n)$$

Lattices ('gitter' in Swedish)

Image by Emma Mae (Flickr) via Wikimedia commons

Partially Ordered Set

Lattices L are based on a *partially ordered set* $\langle \mathcal{L}, \sqsubseteq \rangle$:

- Set: $\mathcal{L}$ describes possible information
- $(\sqsubseteq) \subseteq \mathcal{L} \times \mathcal{L}$:
- Intuition for $a \sqsubseteq b$ (for program analysis):
  - b has at least as much information as a

Partially Ordered Set

Lattices L are based on a *partially ordered set* $\langle \mathcal{L}, \sqsubseteq \rangle$:

- Set: $\mathcal{L}$ describes possible information
- $(\sqsubseteq) \subseteq \mathcal{L} \times \mathcal{L}$:
- Intuition for $a \sqsubseteq b$ (for program analysis):
  - b has at least as much information as a
- $(\sqsubseteq)$ is a *partial order*:

$$
\begin{array}{lll}
a \sqsubseteq a & & \textit{Reflexivity} \\
a \sqsubseteq b \text{ and } b \sqsubseteq a & \Longrightarrow \; a = b & \textit{Antisymmetry} \\
a \sqsubseteq b \text{ and } b \sqsubseteq c & \Longrightarrow \; a \sqsubseteq c & \textit{Transitivity}
\end{array}
$$

Partially Ordered Set

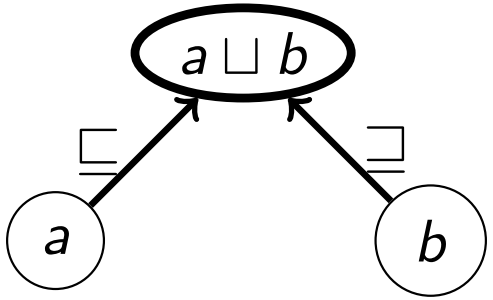
Lattices L are based on a *partially ordered set* $\langle \mathcal{L}, \sqsubseteq \rangle$:

- Set: $\mathcal{L}$ describes possible information
- $(\sqsubseteq) \subseteq \mathcal{L} \times \mathcal{L}$:
- Intuition for $a \sqsubseteq b$ (for program analysis):
  - b has at least as much information as a
- $(\sqsubseteq)$ is a *partial order*:

$$
\begin{array}{lll}
a \sqsubseteq a & & \textit{Reflexivity} \\
a \sqsubseteq b \text{ and } b \sqsubseteq a & \Longrightarrow \quad a = b & \textit{Antisymmetry} \\
a \sqsubseteq b \text{ and } b \sqsubseteq c & \Longrightarrow \quad a \sqsubseteq c & \textit{Transitivity}
\end{array}
$$

- Example:
  - $\mathcal{L} = \{\textit{unknown}, \textit{true}, \textit{false}, \underline{\textit{true-or-false}}\}$
  - $\underline{\textit{unknown}} \sqsubseteq \underline{\textit{true}} \sqsubseteq \underline{\textit{true-or-false}}$
  - $\textit{unknown} \sqsubseteq \underline{\textit{false}} \sqsubseteq \textit{true-or-false}$

Least Upper Bound

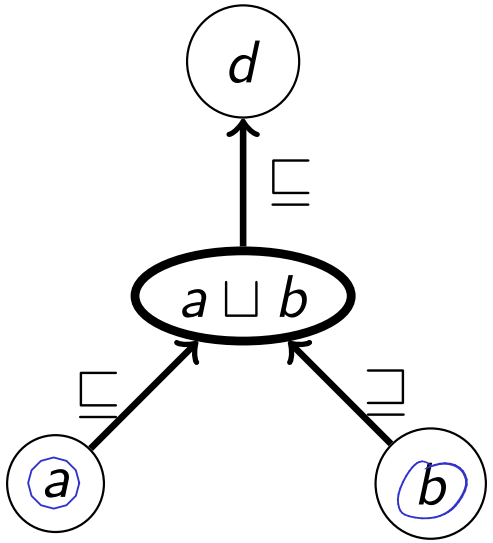


Combining potentially contradictory information:

- *Join operator*: $(\sqcup) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$
- Pointwise monotonic:

$$a \sqsubseteq a \sqcup b \text{ and } b \sqsubseteq a \sqcup b$$

Least Upper Bound



Combining potentially contradictory information:

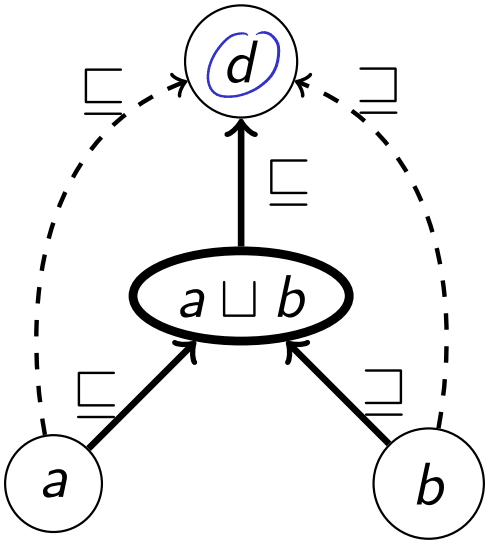
- *Join operator*: $(\sqcup) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$
- Pointwise monotonic:

$$a \sqsubseteq a \sqcup b \text{ and } b \sqsubseteq a \sqcup b$$

- *Least* element with this property:

$$a \sqsubseteq d \text{ and } b \sqsubseteq d \implies a \sqcup b \sqsubseteq d$$

Least Upper Bound



Combining potentially contradictory information:

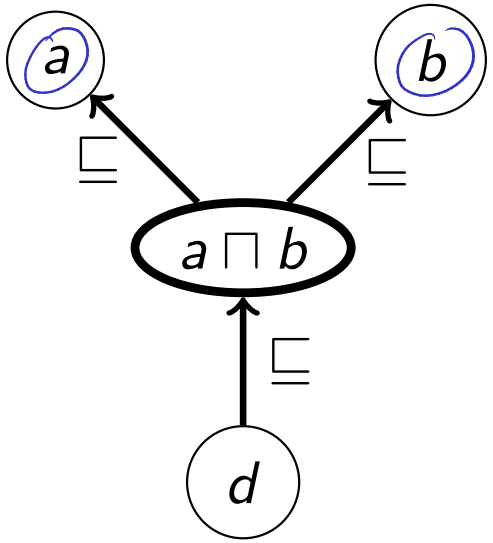
- *Join operator*: $(\sqcup) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$
- Pointwise monotonic:

$$a \sqsubseteq a \sqcup b \text{ and } b \sqsubseteq a \sqcup b$$

- *Least* element with this property:

$$a \sqsubseteq d \text{ and } b \sqsubseteq d \implies \underline{a \sqcup b} \sqsubseteq d$$

Greatest Lower bound



Converse operation:

- *Meet operator*: $(\sqcap) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$
- Pointwise monotonic:

$$a \sqcap b \sqsubseteq a \text{ and } a \sqcap b \sqsubseteq b$$

- *Greatest* element with this property:

$$d \sqsubseteq a \text{ and } d \sqsubseteq b \implies d \sqsubseteq a \sqcap b$$

Lattices

$$L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$$

- $\mathcal{L}$: Underlying set
- $(\sqsubseteq) \subseteq \mathcal{L} \times \mathcal{L}$: Partial Order
- $(\sqcup) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$: Join (computes l.u.b.)
- $(\sqcap) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$: Meet (computes g.l.b.)

Lattices

$$L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$$

- $\mathcal{L}$: Underlying set
- $(\sqsubseteq) \subseteq \mathcal{L} \times \mathcal{L}$: Partial Order
- $(\sqcup) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$: Join (computes l.u.b.)
- $(\sqcap) : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$: Meet (computes g.l.b.)

- can show:

$$\begin{array}{rrcl} \text{Commutativity:} & a \sqcup b & = & b \sqcup a \\ \text{Associativity:} & a \sqcup (b \sqcup c) & = & (a \sqcup b) \sqcup c \end{array}$$

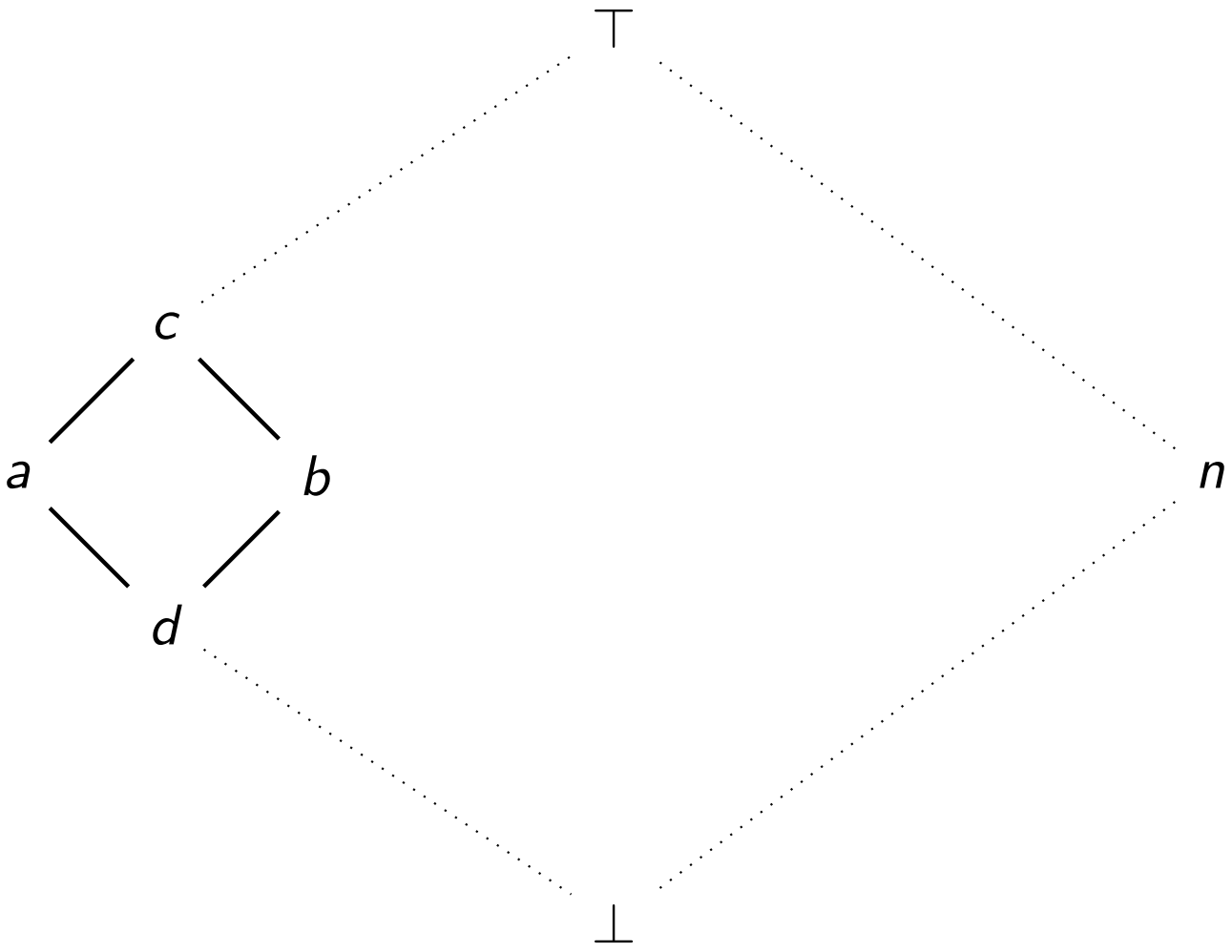
(Analogous for $\sqcap$)

Complete Lattices

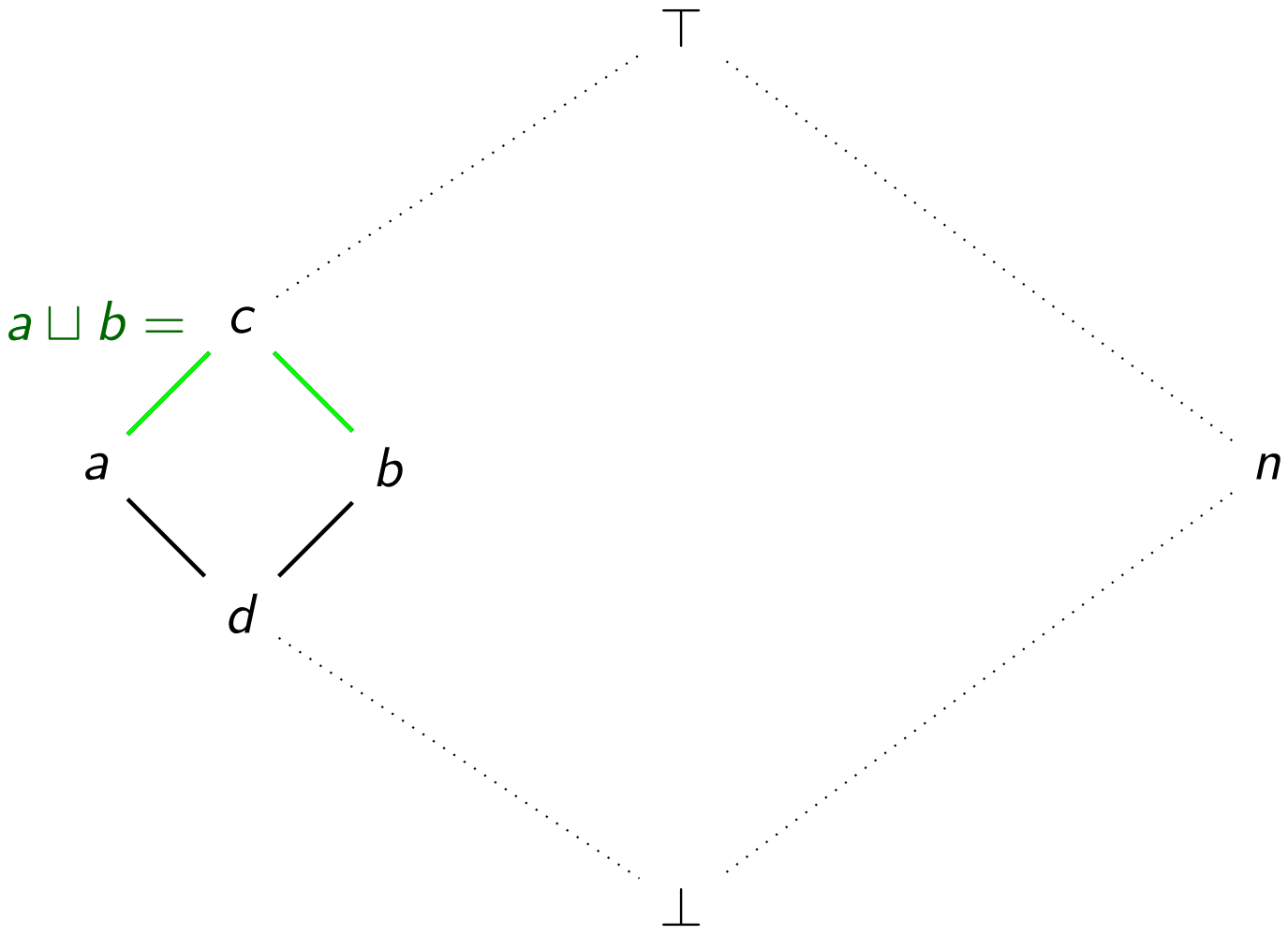
A lattice $L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$ is *complete* iff:

- For any $\mathcal{L}' \subseteq \mathcal{L}$ there exist:
  - $\top = \bigsqcup \mathcal{L}'$
  - $\bot = \bigsqcap \mathcal{L}'$

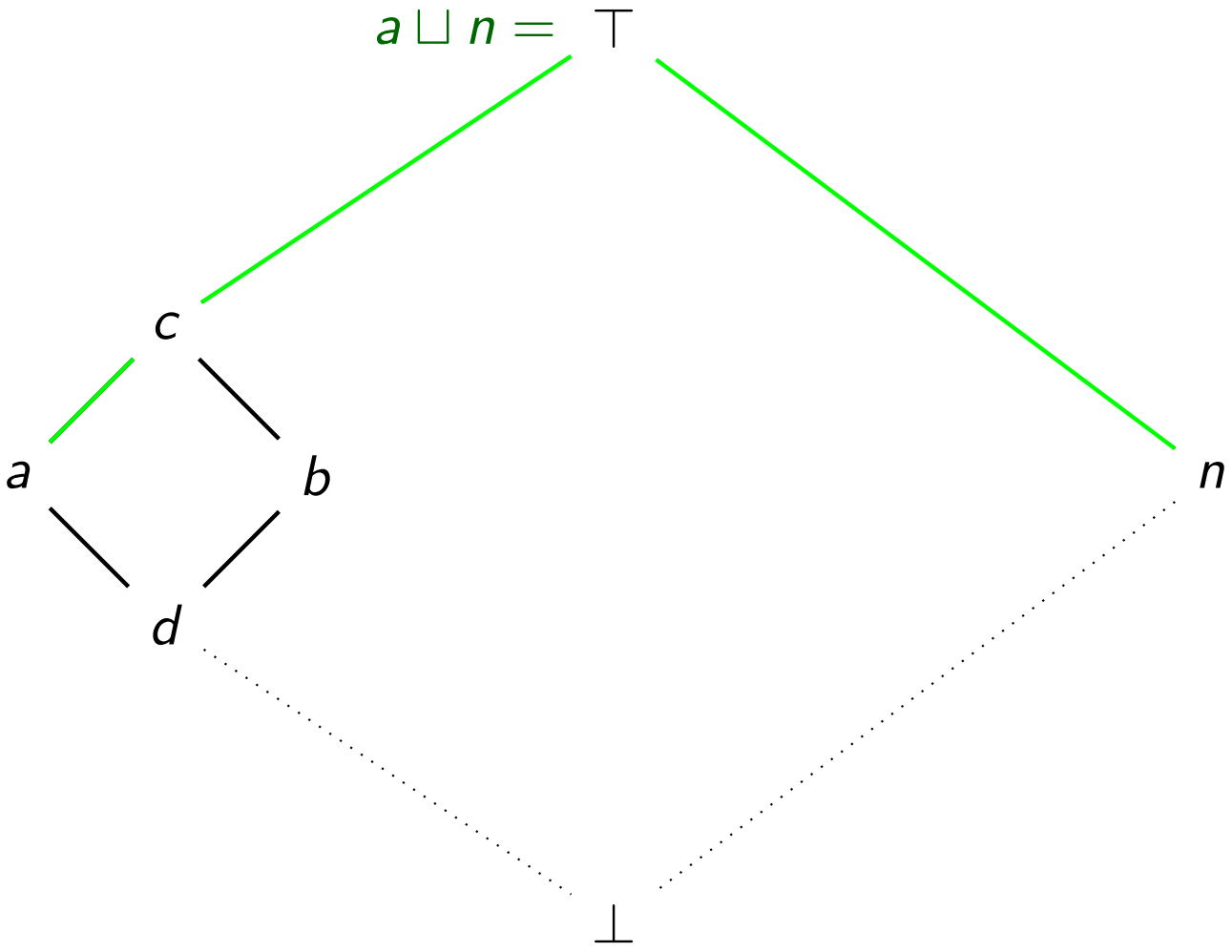
Complete Lattices: Visually



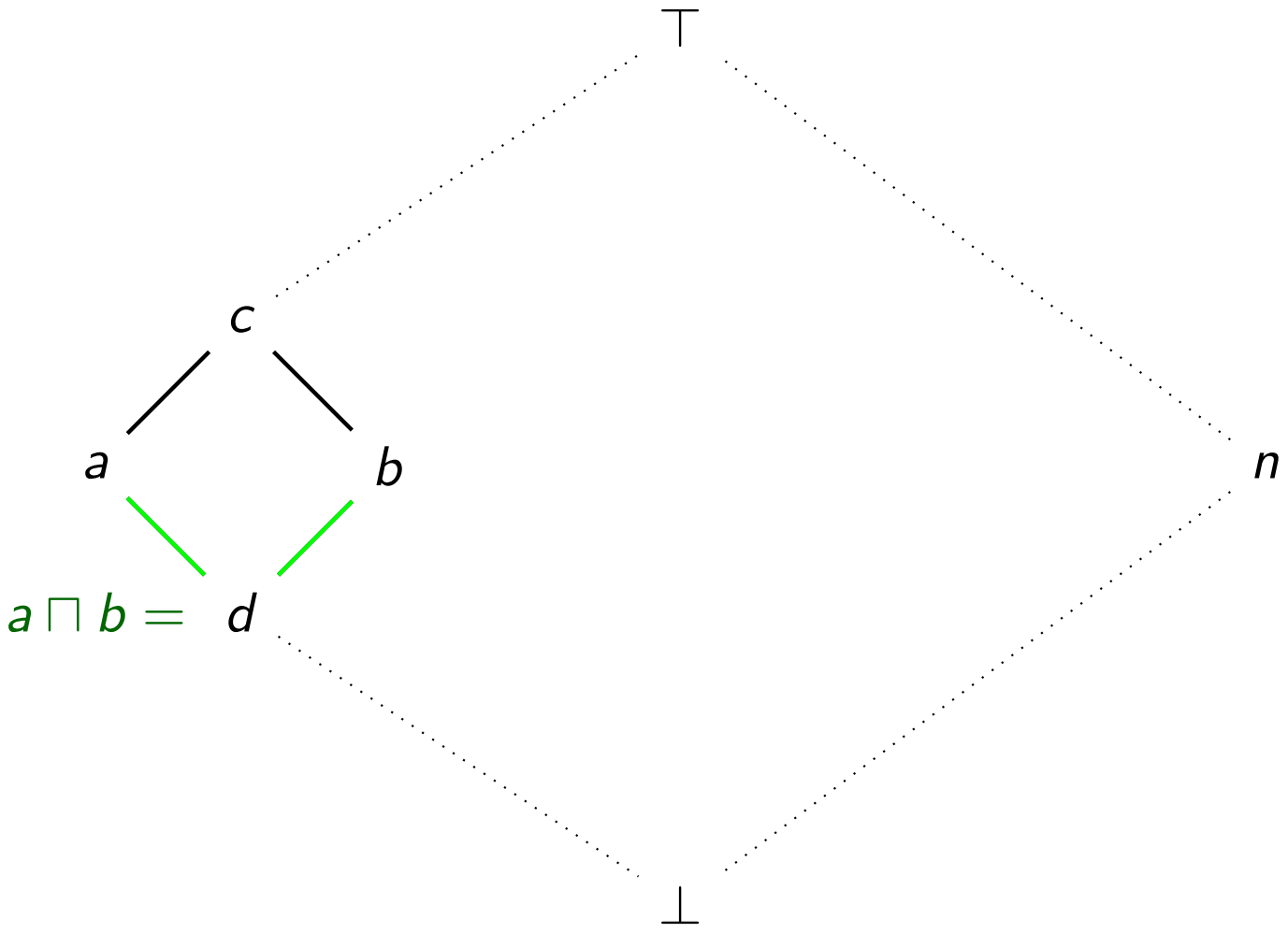
Complete Lattices: Visually



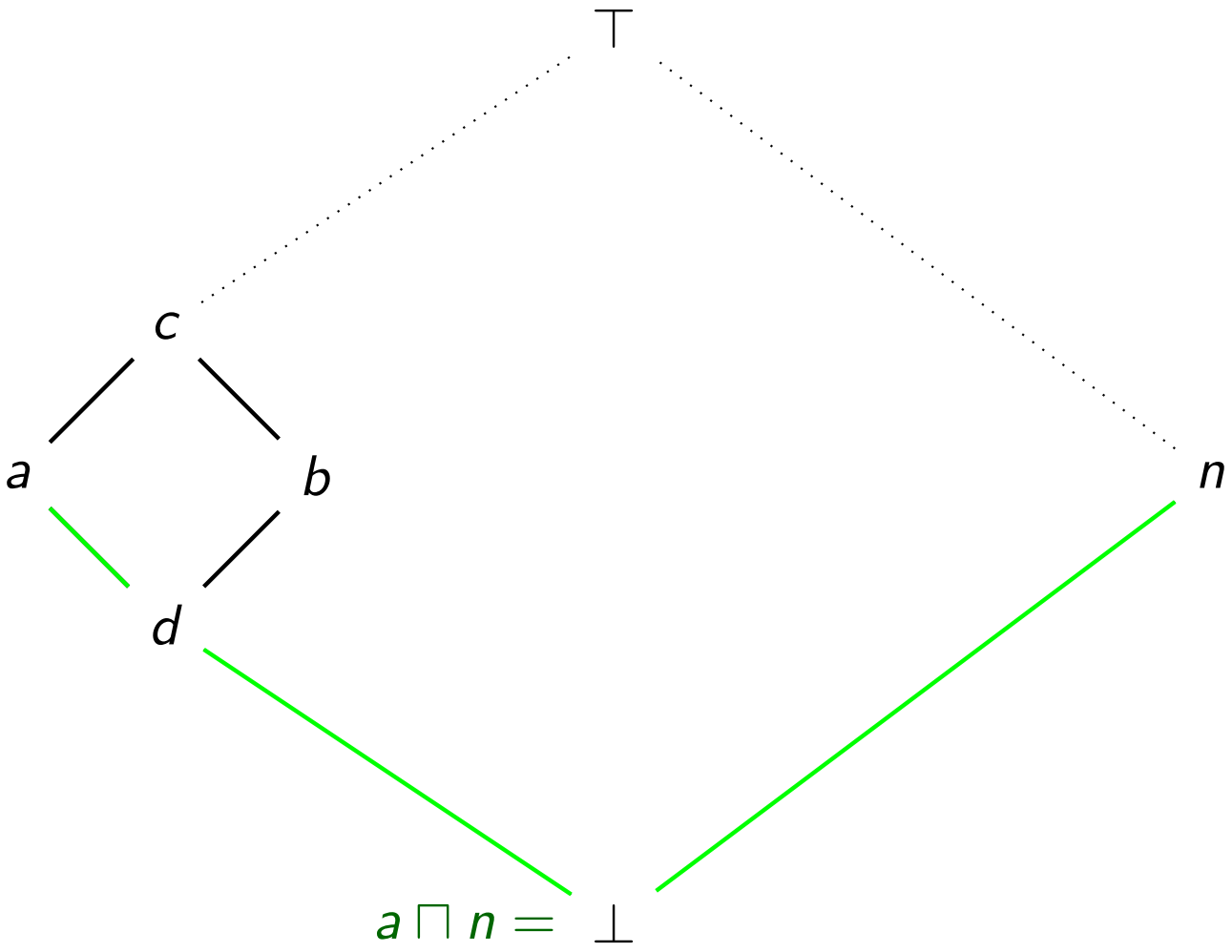
Complete Lattices: Visually



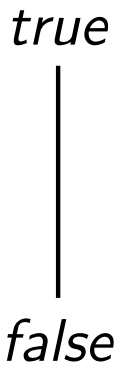
Complete Lattices: Visually



Complete Lattices: Visually

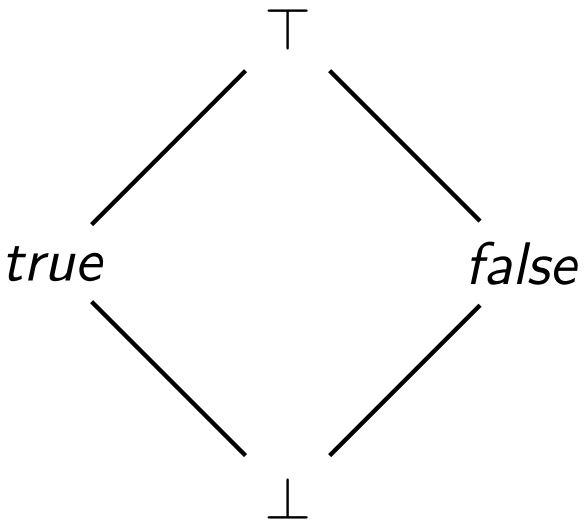


Example: Binary Lattice



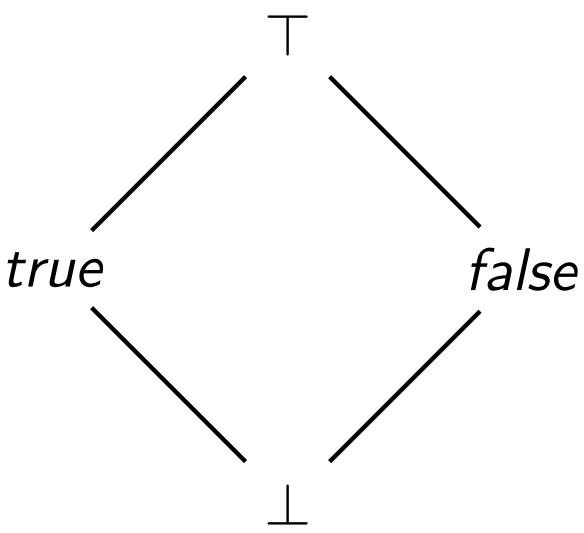
- $\top = \mathit{true}$
- $\bot = \mathit{false}$
- $\sqcup =$ *logical "or"*
- $\sqcap =$ *logical "and"*

Example: Booleans



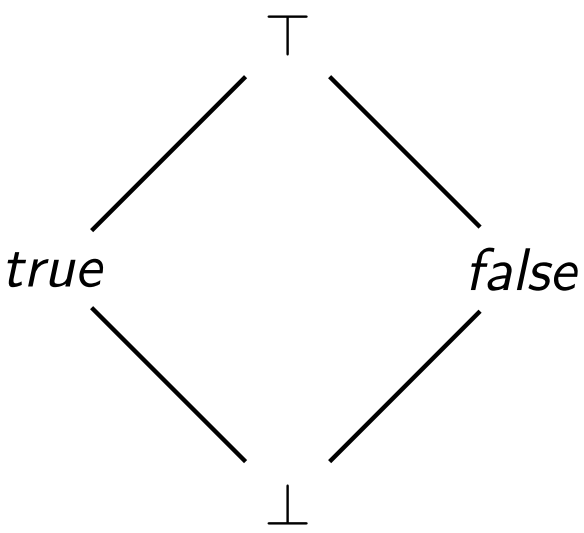
- If $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$:
  - Lattice sometimes called $\mathbb{B}_{\bot}^{\top}$

Example: Booleans



- If $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$:
  - Lattice sometimes called $\mathbb{B}_{\bot}^{\top}$
- Interpretation for data flow e.g.:
  - $\top$ = true-or-false
  - $\bot$ = unknown
  - $a \sqcup b$: either a or b
  - $a \sqcap b$: both a and b

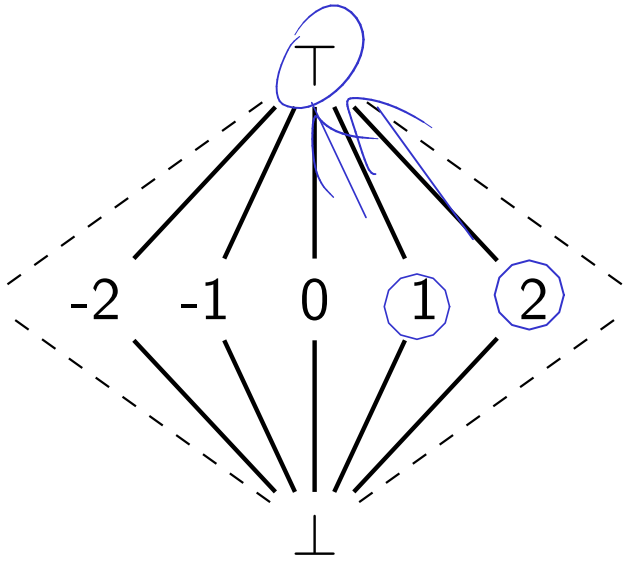
Example: Booleans



- If $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$:
  - Lattice sometimes called $\mathbb{B}_{\bot}^{\top}$
- Interpretation for data flow e.g.:
  - $\top$ = true-or-false
  - $\bot$ = unknown
  - $a \sqcup b$: either a or b
  - $a \sqcap b$: both a and b

Other interpretations possible

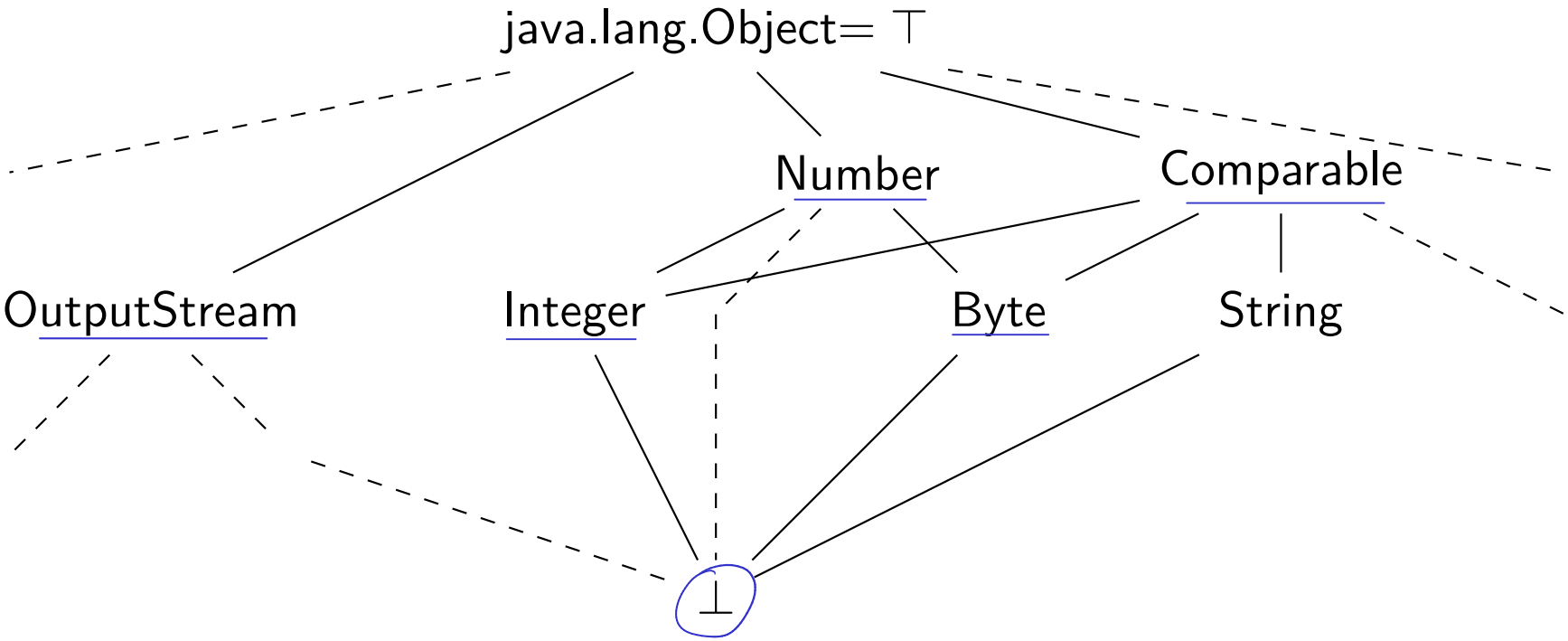
Example: Flat Lattice on Integers



- Sometimes written $\mathbb{Z}_{\bot}^{\top}$
- $\top = \cancel{\emptyset}\ \mathbb{Z}$
- $\bot = \cancel{\mathbb{Z}}\ \emptyset$
- $a \sqcup b = \begin{cases} a & \text{iff} \quad a = b \\ \top & \quad\ \ \text{otherwise} \end{cases}$
- $a \sqcap b = \begin{cases} a & \text{iff} \quad a = b \\ \bot & \quad\ \ \text{otherwise} \end{cases}$

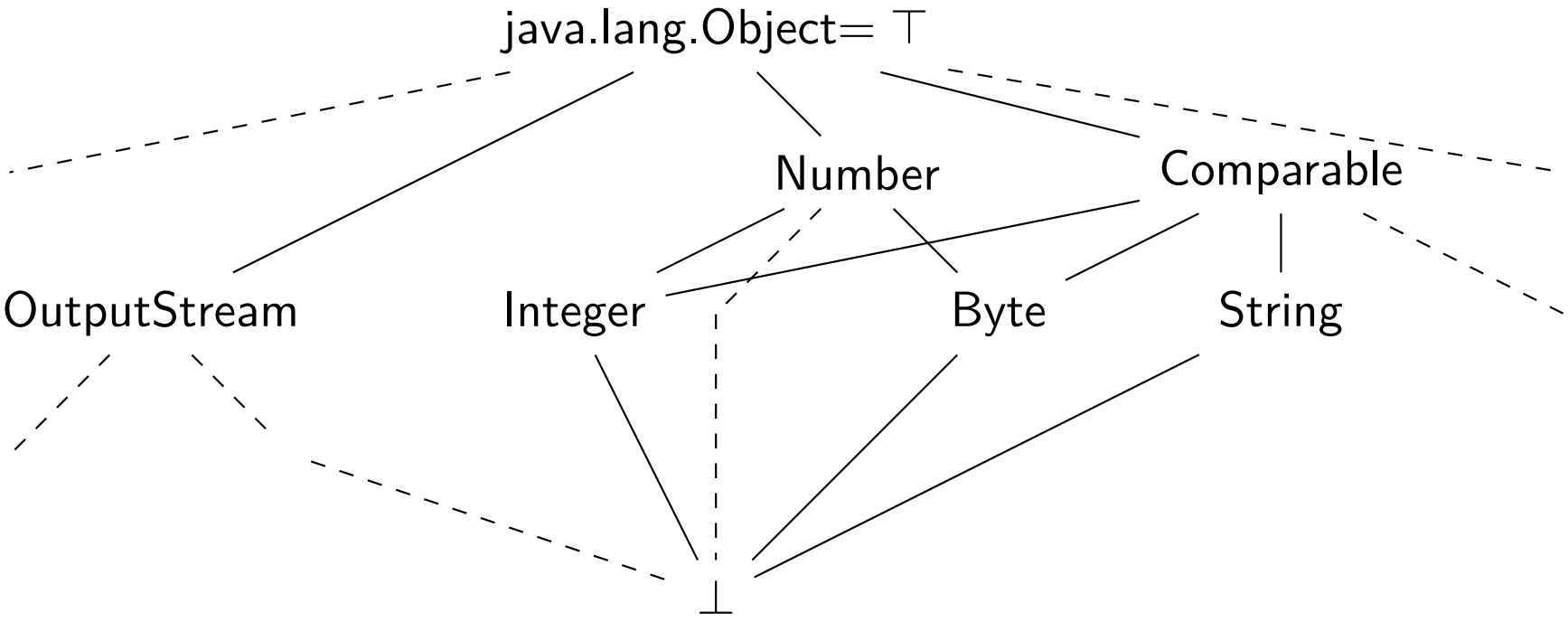
Analogous for other $X_{\bot}^{\top}$ from set X

Example: Type Hierarchy Lattices



- ⊔ constructs most precise supertype

Example: Type Hierarchy Lattices



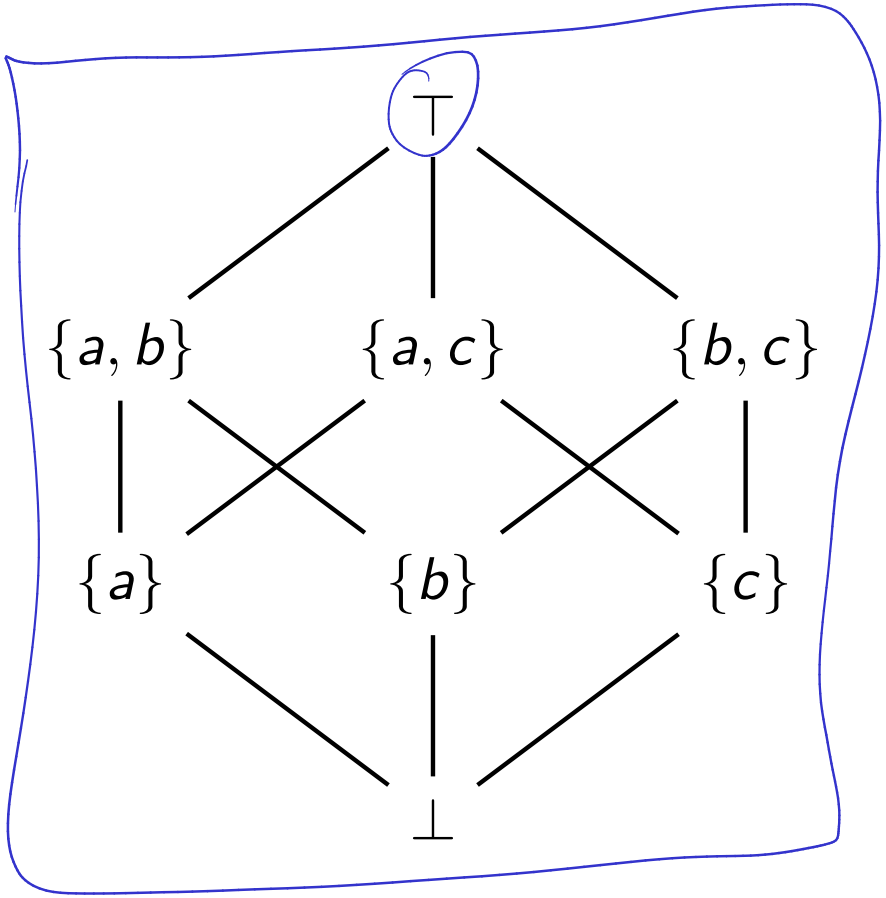
- ⊔ constructs most precise supertype
- ⊓ constructs *intersection types*:

$$\text{java.lang.Comparable} \sqcap \text{java.io.Serializable}$$

- *Java notation*:

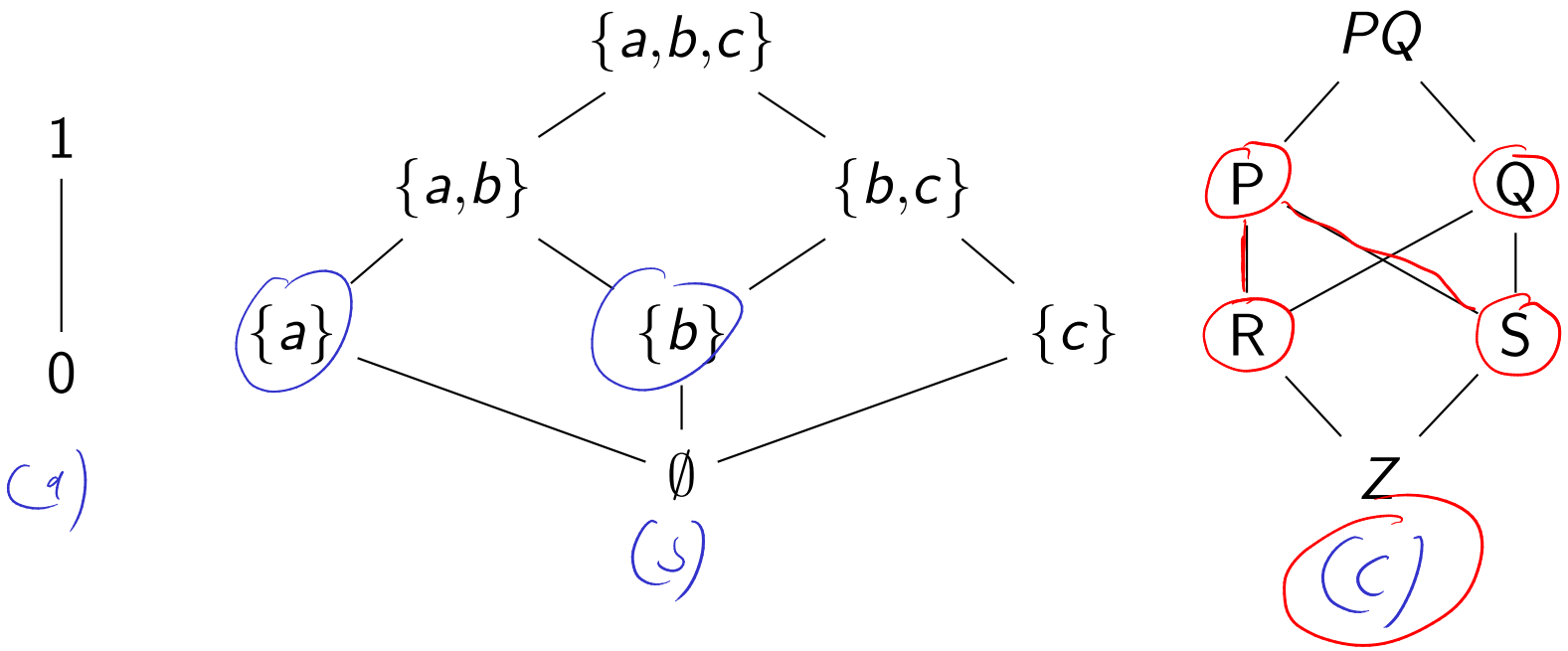
```
java.lang.Comparable & java.io.Serializable
```

Example: Powersets

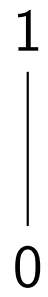


- Take any set $S = \underline{\{a, b, c\}}$ $\mathcal{D}^+ \ \mathcal{D}^- \ \mathcal{D}^0$
- $\mathcal{L} = \mathcal{P}(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\} \ldots$
- $\top = S$
- $\bot = \emptyset$
- $(\sqcup) = (\cup)$
- $(\sqcap) = (\cap)$

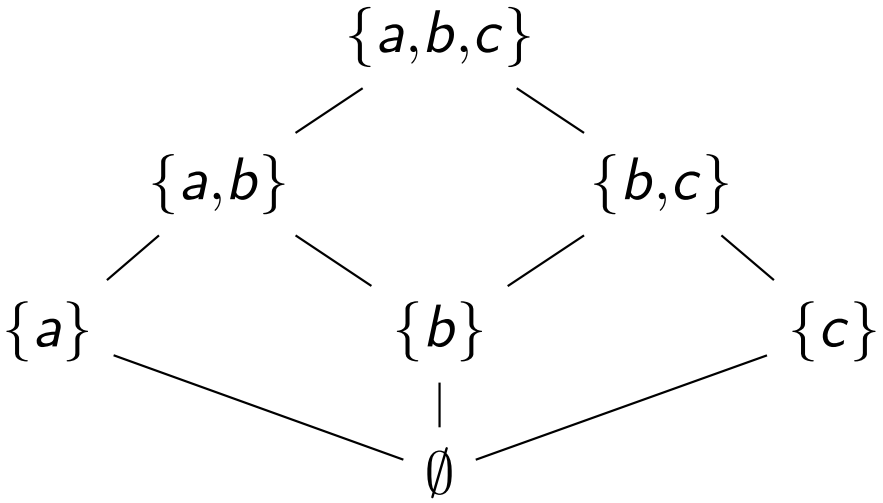
Example: Lattices and Non-Lattices



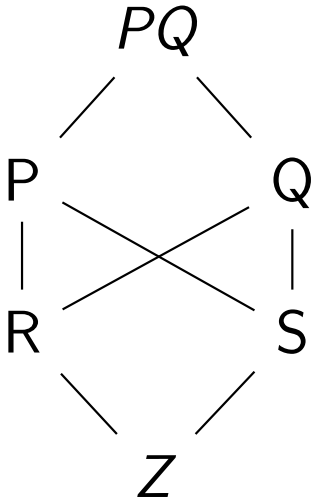
Example: Lattices and Non-Lattices



Lattice



Lattice



Not A Lattice

Right-hand side is missing e.g. a unique $R \sqcup S$

Example: Natural numbers with 0, ω

- $\top = \omega$
- $\bot = 0$
- $a \sqcup b =$ maximum of a and b
- $a \sqcap b =$ minimum of a and b

Product Lattices

- Assume (complete) lattices:
  - $L_1 = \langle \mathcal{L}_1, \sqsubseteq_1, \sqcap_1, \sqcup_1, \top_1, \bot_1 \rangle$
  - $L_2 = \langle \mathcal{L}_2, \sqsubseteq_2, \sqcap_2, \sqcup_2, \top_2, \bot_2 \rangle$

Product Lattices

- Assume (complete) lattices:
  - $L_1 = \langle \mathcal{L}_1, \sqsubseteq_1, \sqcap_1, \sqcup_1, \top_1, \bot_1 \rangle$
  - $L_2 = \langle \mathcal{L}_2, \sqsubseteq_2, \sqcap_2, \sqcup_2, \top_2, \bot_2 \rangle$
- Let $L_1 \times L_2 = \langle \mathcal{L}_1 \times \mathcal{L}_2, \sqsubseteq, \sqcap, \sqcup, \top, \bot \rangle$ where:

Product Lattices

- Assume (complete) lattices:
  - $L_1 = \langle \mathcal{L}_1, \sqsubseteq_1, \sqcap_1, \sqcup_1, \top_1, \bot_1 \rangle$
  - $L_2 = \langle \mathcal{L}_2, \sqsubseteq_2, \sqcap_2, \sqcup_2, \top_2, \bot_2 \rangle$
- Let $L_1 \times L_2 = \langle \mathcal{L}_1 \times \mathcal{L}_2, \sqsubseteq, \sqcap, \sqcup, \top, \bot \rangle$ where:
  - $\langle a, b \rangle \sqsubseteq \langle a', b' \rangle$ iff $a \sqsubseteq_1 a'$ and $b \sqsubseteq_2 b'$
  - $\langle a, b \rangle \sqcap \langle a', b' \rangle = \langle a \sqcap_1 a', b \sqcap_2 b' \rangle$
  - $\langle a, b \rangle \sqcup \langle a', b' \rangle = \langle a \sqcup_1 a', b \sqcup_2 b' \rangle$
  - $\top = \langle \top_1, \top_2 \rangle$
  - $\bot = \langle \bot_1, \bot_2 \rangle$

Point-wise products of (complete) lattices are again (complete) lattices

Summary

- Complete lattices are formal basis for many program analyses
- Complete lattice $L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup, \top, \bot \rangle$
  - $\mathcal{L}$: Carrier set
  - ($\sqsubseteq$): Partial order
  - ($\sqcap$): Join operation: find least upper lower bound
  - ($\sqcup$): Meet operation: find greatest lower bound (not usually necessary)
  - $\top$: Top-most element of complete lattice
  - $\bot$: Bottom-most element of complete lattice
- **Product Lattices**: $L_1 \times L_2$ forms a lattice if L_1 and L_2 are lattices

Monotone Frameworks

Monotone Framework	**Lattice**
Abstract Domain	$L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
$join_b(x_1, \ldots, x_n)$	$x_1 \sqcup \ldots \sqcup x_n$
	$x \sqcap y$
'Unknown' start value	$\bot$
'Could be anything' end value	$\top$

Monotone Frameworks

Monotone Framework	**Lattice**
Abstract Domain	$L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
$join_b(x_1, \ldots, x_n)$	$x_1 \sqcup \ldots \sqcup x_n$
	$x \sqcap y$ (Not needed)
'Unknown' start value	$\bot$
'Could be anything' end value	$\top$

Monotone Frameworks

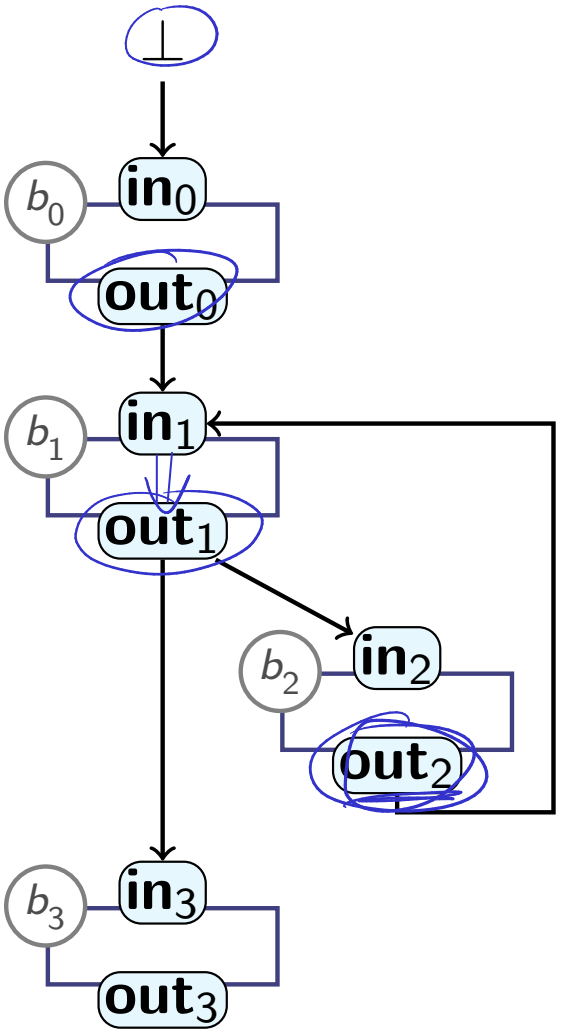
Monotone Framework	**Lattice**
Abstract Domain	$L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
$join_b(x_1, \ldots, x_n)$	$x_1 \sqcup \ldots \sqcup x_n$
	$x \sqcap y$ (Not needed)
'Unknown' start value	$\bot$
'Could be anything' end value	$\top$

- *Monotone Frameworks* (Killdall '77):
  - Lattice L of *finite height*
    (= satisfies Ascending Chain Condition)
  - Monotone $trans_b$
    - 'compatible' with semantics

$\Rightarrow$ Data flow analysis with Soundness and Termination

Formalising our Naïve Algorithm

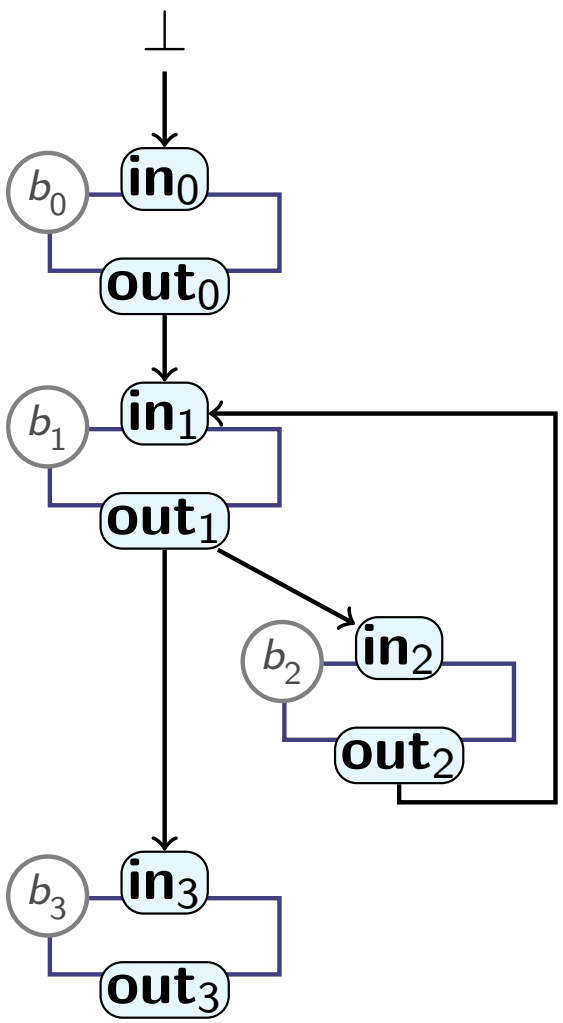
$$
\begin{aligned}
\mathbf{out}_0 &= \mathit{trans}_0(\bot) \\
\mathbf{out}_1 &= \mathit{trans}_1(\mathbf{out}_0 \sqcup \mathbf{out}_2) \\
\mathbf{out}_2 &= \mathit{trans}_2(\mathbf{out}_1) \\
\mathbf{out}_3 &= \mathit{trans}_3(\mathbf{out}_2)
\end{aligned}
$$



Formalising our Naïve Algorithm

$$
\begin{aligned}
\mathbf{out}_0 &= \mathit{trans}_0(\bot) \\
\mathbf{out}_1 &= \mathit{trans}_1(\mathbf{out}_0 \sqcup \mathbf{out}_2) \\
\mathbf{out}_2 &= \mathit{trans}_2(\mathbf{out}_1) \\
\mathbf{out}_3 &= \mathit{trans}_3(\mathbf{out}_2)
\end{aligned}
$$

- Lattices $\mathbf{out}_0 : L_0, \ldots, \mathbf{out}_3 : L_3$
- Can build:
  - $L_{0\ldots3} = L_0 \times L_1 \times L_2 \times L_3$: Product Lattice
  - $\bot_{0\ldots3} = \langle \bot_0, \bot_1, \bot_2, \bot_3 \rangle$

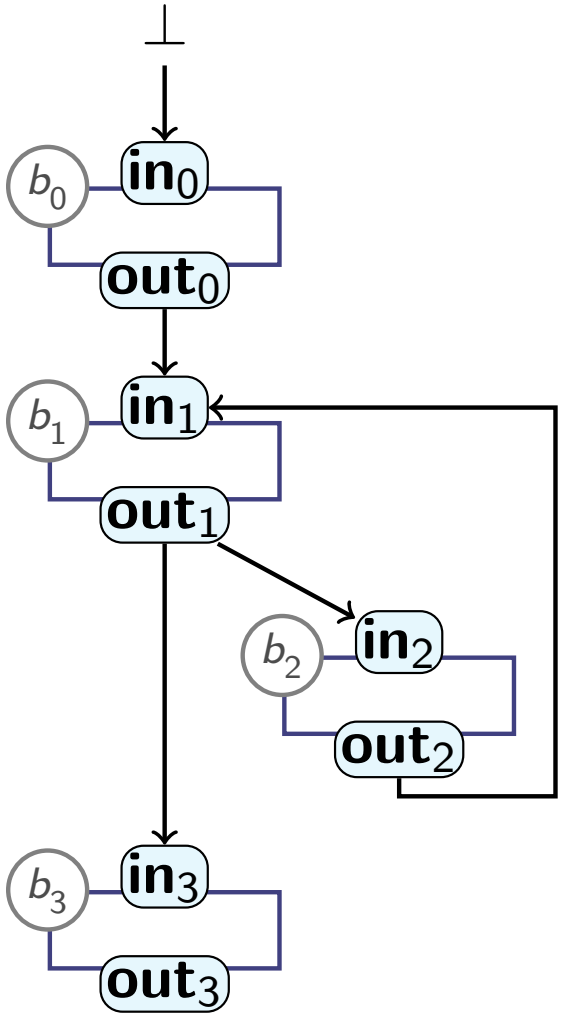


Formalising our Naïve Algorithm

$$
\begin{aligned}
\mathbf{out}_0 &= \mathit{trans}_0(\bot) \\
\mathbf{out}_1 &= \mathit{trans}_1(\mathbf{out}_0 \sqcup \mathbf{out}_2) \\
\mathbf{out}_2 &= \mathit{trans}_2(\mathbf{out}_1) \\
\mathbf{out}_3 &= \mathit{trans}_3(\mathbf{out}_2)
\end{aligned}
$$

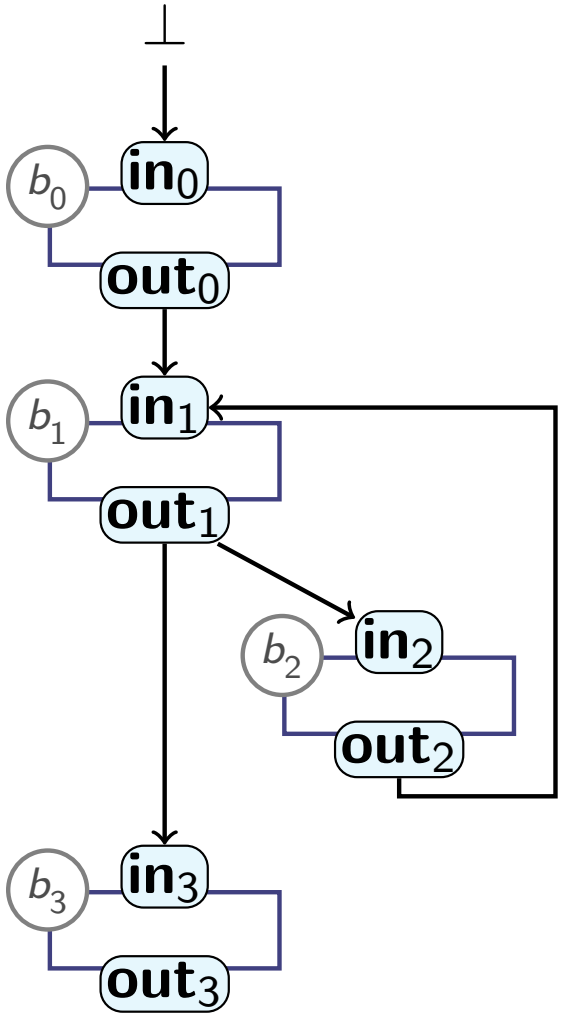
- Lattices $\mathbf{out}_0 : L_0, \ldots, \mathbf{out}_3 : L_3$
- Can build:
  - $L_{0\ldots3} = L_0 \times L_1 \times L_2 \times L_3$
  - $\bot_{0\ldots3} = \langle \bot_0, \bot_1, \bot_2, \bot_3 \rangle$
  - transfer function:

$$
\mathit{trans}_{0\ldots3}(\langle v_0, v_1, v_2, v_3 \rangle) = \left\langle \begin{array}{l} \mathit{trans}_0(v_0), \\ \mathit{trans}_1(v_0 \sqcup v_2), \\ \mathit{trans}_2(v_1), \\ \mathit{trans}_3(v_1) \end{array} \right\rangle
$$



Formalising our Naïve Algorithm

$$
\begin{aligned}
\mathbf{out}_0 &= \mathit{trans}_0(\bot) \\
\mathbf{out}_1 &= \mathit{trans}_1(\mathbf{out}_0 \sqcup \mathbf{out}_2) \\
\mathbf{out}_2 &= \mathit{trans}_2(\mathbf{out}_1) \\
\mathbf{out}_3 &= \mathit{trans}_3(\mathbf{out}_2)
\end{aligned}
$$



- Lattices $\mathbf{out}_0 : L_0, \ldots, \mathbf{out}_3 : L_3$
- Can build:
  - $L_{0\ldots3} = L_0 \times L_1 \times L_2 \times L_3$
  - $\bot_{0\ldots3} = \langle \bot_0, \bot_1, \bot_2, \bot_3 \rangle$
  - Monotone transfer function:

Analysis states at nodes

b_0
b_1
b_2
b_3

$$
\mathit{trans}_{0\ldots3}(\langle v_0, v_1, v_2, v_3 \rangle) = \left\langle \begin{array}{l} \mathit{trans}_0(v_0), \\ \mathit{trans}_1(v_0 \sqcup v_2), \\ \mathit{trans}_2(v_1), \\ \mathit{trans}_3(v_2) \end{array} \right\rangle
$$

Reaching a Solution

- In general:
  - Program P:
    - "Program Lattice" L_P
    - $\bot_P$
    - $trans_P$: Compute one step of naïve analysis
  - Repeat $trans_P$:

$$fp_\bot = trans_P^n(\bot_P)$$

Reaching a Solution

- In general:
  - Program P:
    - "Program Lattice" L_P
    - $\bot_P$
    - $trans_P$: Compute one step of naïve analysis
  - Repeat $trans_P$:

$$fp_\bot = trans_P^n(\bot_P)$$

- $fp_\bot$ is *Fixpoint* of $trans_P$:

$$fp_\bot = trans_P(fp_\bot)$$

- Fixpoint exists **iff** L_P satisfies Ascending Chain Condition

Reaching a Solution

- In general:
  - Program P:
    - "Program Lattice" L_P
    - $\bot_P$
    - $trans_P$: Compute one step of naïve analysis
  - Repeat $trans_P$:

$$fp_\bot = trans_P^n(\bot_P)$$

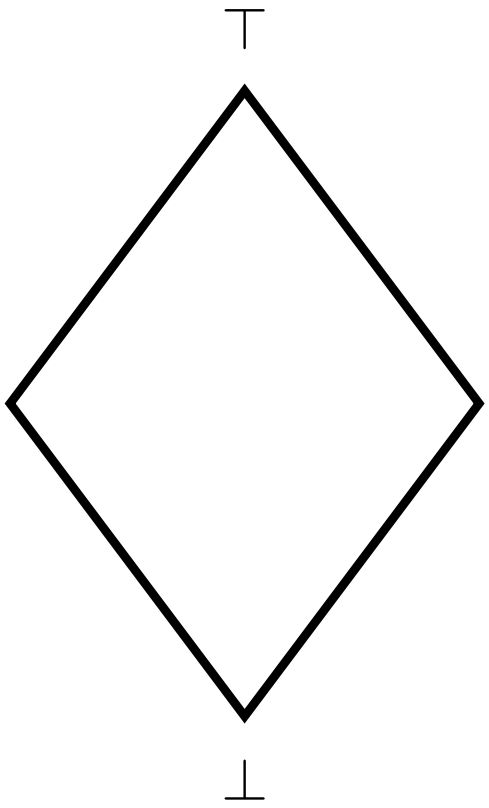
- $fp_\bot$ is *Fixpoint* of $trans_P$:

$$fp_\bot = trans_P(fp_\bot)$$

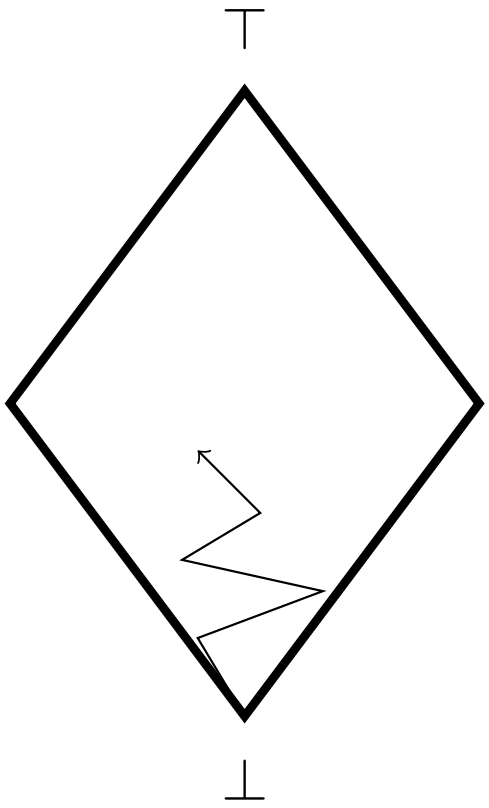
- Fixpoint exists **iff** L_P satisfies Ascending Chain Condition

Cousot & Cousot (1979), based on Kleene (1952), based on Knaster & Tarski (1933)

Fixpoints

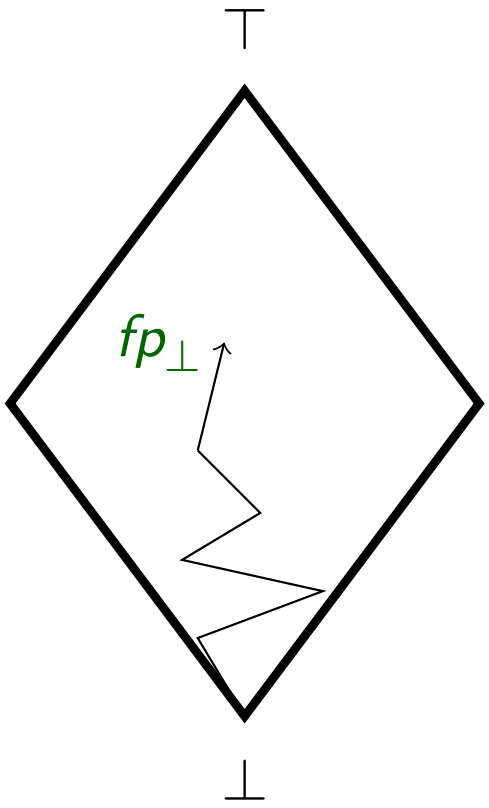


Fixpoints



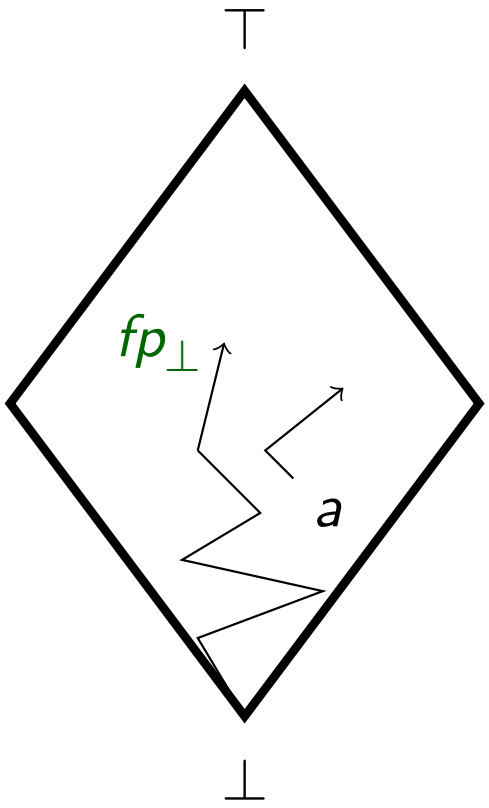
- Repeat $trans_P$

Fixpoints



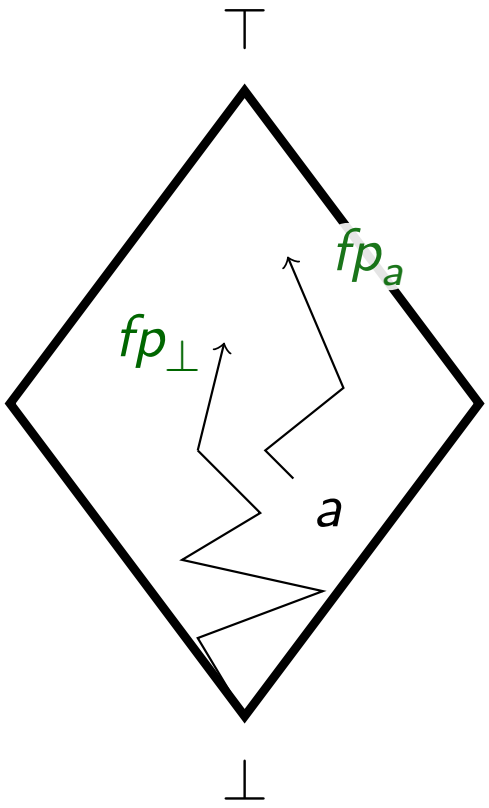
- Repeat $trans_P$ until fixpoint

Fixpoints



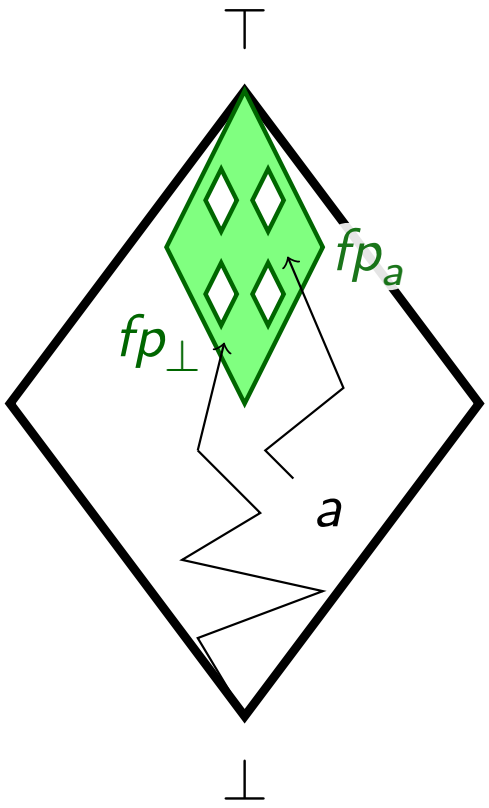
- Repeat $trans_P$ until fixpoint
- Can start from *any* point a

Fixpoints



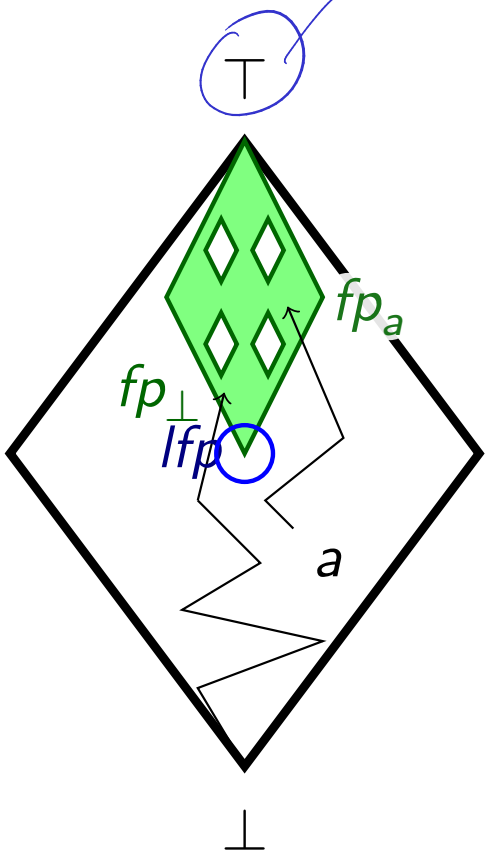
- Repeat $trans_P$ until fixpoint
- Can start from *any* point a
- Multiple fixpoints possible

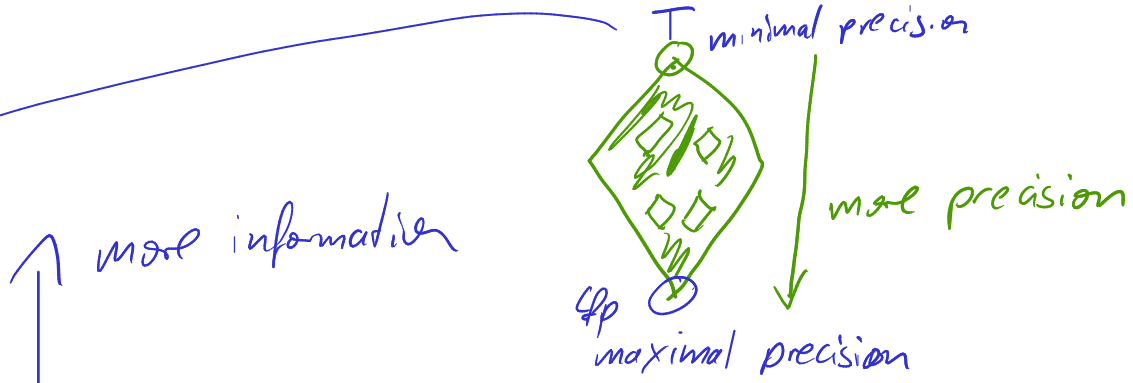
Fixpoints



- Repeat $trans_P$ until fixpoint
- Can start from *any* point a
- Multiple fixpoints possible
  - Each is a *sound* solution
  - Form a lattice
    (Knaster-Tarski, 1933)

Fixpoints





- Repeat $trans_P$ until fixpoint
- Can start from *any* point a
- Multiple fixpoints possible
  - Each is a *sound* solution
  - Form a lattice
    (Knaster-Tarski, 1933)
- Least Fixpoint: Highest Precision

Value Range Analysis

'Find *value range* (interval of possible values) for x'

[1,1] [1,4]

Python

```
x = 1
while ...:
  if ...:
    x = 4
  else:
    x = 7
```

Value Range Analysis

'Find *value range* (interval of possible values) for x'

Python

```
x = 1
while ...:
  if ...:
    x = 4
  else:
    x = 7
```

- Multiple possible *sound* solutions:
  - $\top$
  - $[-99, 99]$
  - $[1, 10]$
  - $[1, 7]$
- All of these values are fixpoints

Value Range Analysis

'Find *value range* (interval of possible values) for x'

Python

```
x = 1
while ...:
  if ...:
    x = 4
  else:
    x = 7
```

- Multiple possible *sound* solutions:
  - $\top$
  - $[-99, 99]$
  - $[1, 10]$
  - $[1, 7]$
- All of these values are fixpoints
- $[1, 7]$ is *least fixpoint*

Summary

- **Monotone Frameworks**:
  - Combine:
    - Monotone transfer functions $trans_b$
    - Finite-Height Lattices

$$join_b(v_1, \ldots, v_k) = v_1 \sqcup \ldots \sqcup v_k$$

  - Guarantee:
    - Termination
    - Soundness
- With Monotone Frameworks, iterating $trans_b$ and $join_b$ produces **Fixpoint** (or *Fixed Point*)
  - Works from *any* starting point, possibly different fixpoint
    - Fixpoints form **Fixpoint Lattice**
    - **Least Fixpoint** (Bottom element) is *most precise solution*
- (Soundness only if $trans_b$ are *compatible*)

An Algorithm for Fixpoints

- So far: naïve algorithm for computing fixpoint
  - Produces a fixpoint
  - Keeps iterating *all* $trans_b$ / $join_b$ functions, even if nothing changed

An Algorithm for Fixpoints

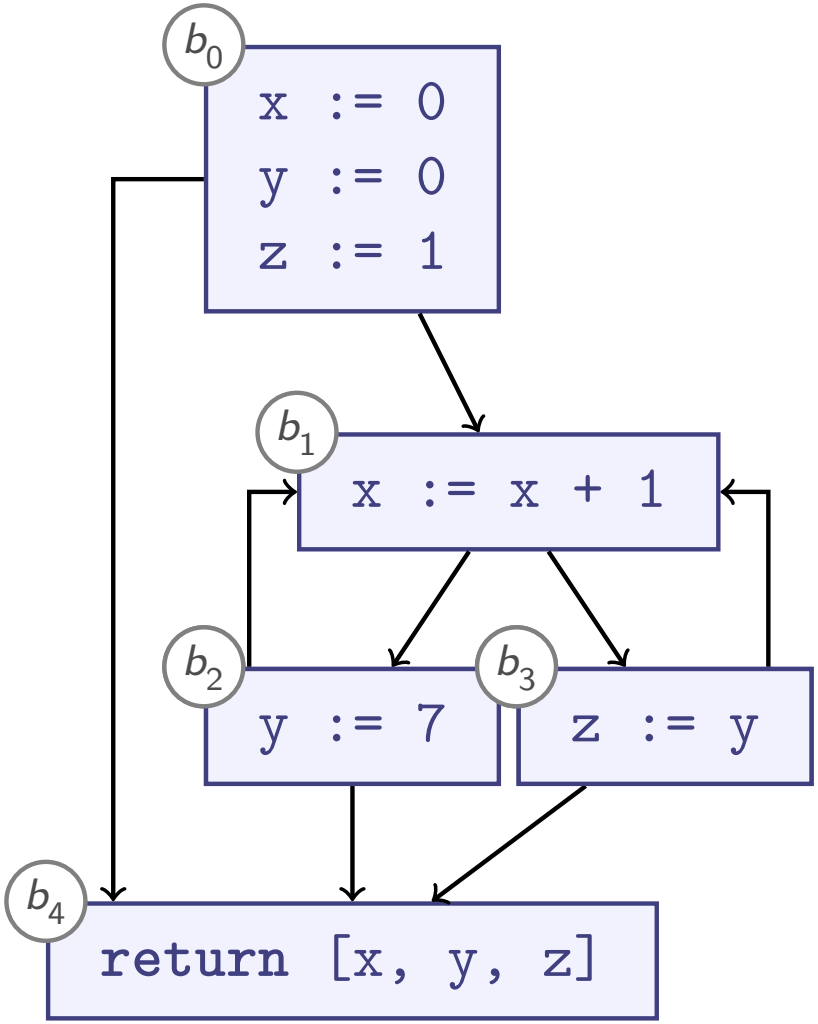
- So far: naïve algorithm for computing fixpoint
  - Produces a fixpoint
  - Keeps iterating *all* $trans_b$ / $join_b$ functions, even if nothing changed
- Optimise processing with *worklist*
  - Set-like datastructure:
    - **add** element (if not already present)
    - **contains** test: is element present?
    - **pop** element: remove and return one element
  - Tracks *what's left to be done*

An Algorithm for Fixpoints

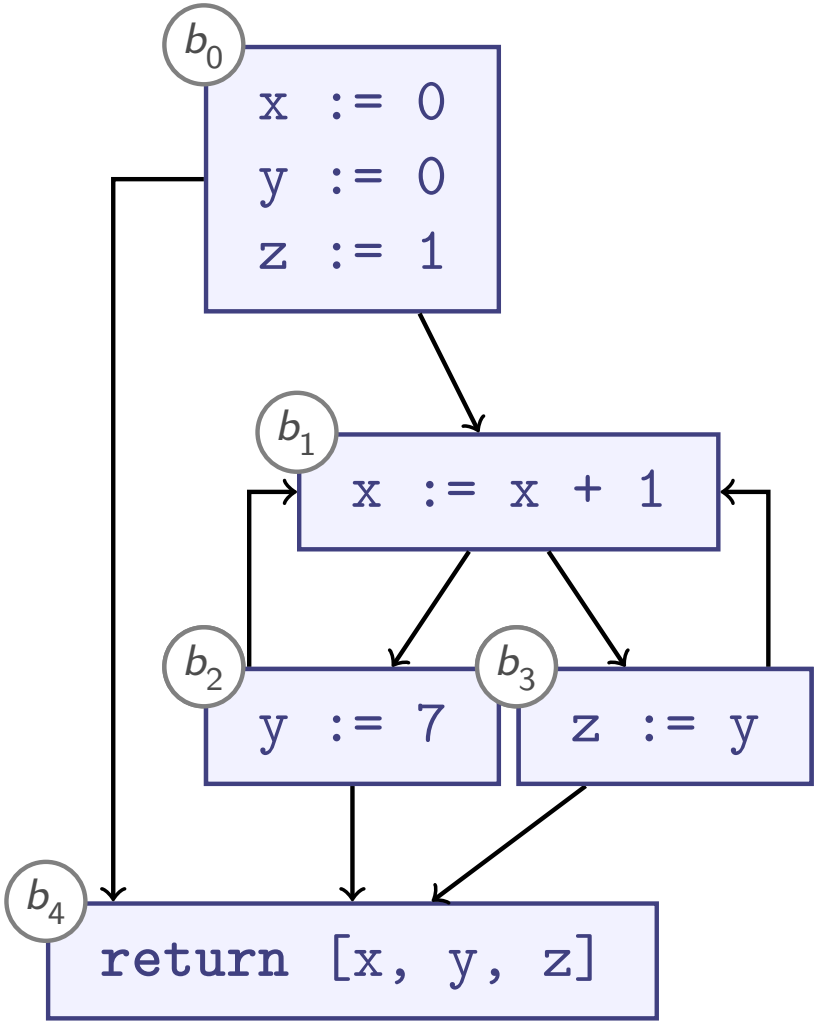
- So far: naïve algorithm for computing fixpoint
  - Produces a fixpoint
  - Keeps iterating *all* $trans_b$ / $join_b$ functions, even if nothing changed
- Optimise processing with *worklist*
  - Set-like datastructure:
    - **add** element (if not already present)
    - **contains** test: is element present?
    - **pop** element: remove and return one element
  - Tracks *what's left to be done*

$\Rightarrow$ "MFP" (Minimal Fixed Point) Algorithm
(*Does not always produce least fixpoint!*)

Example:



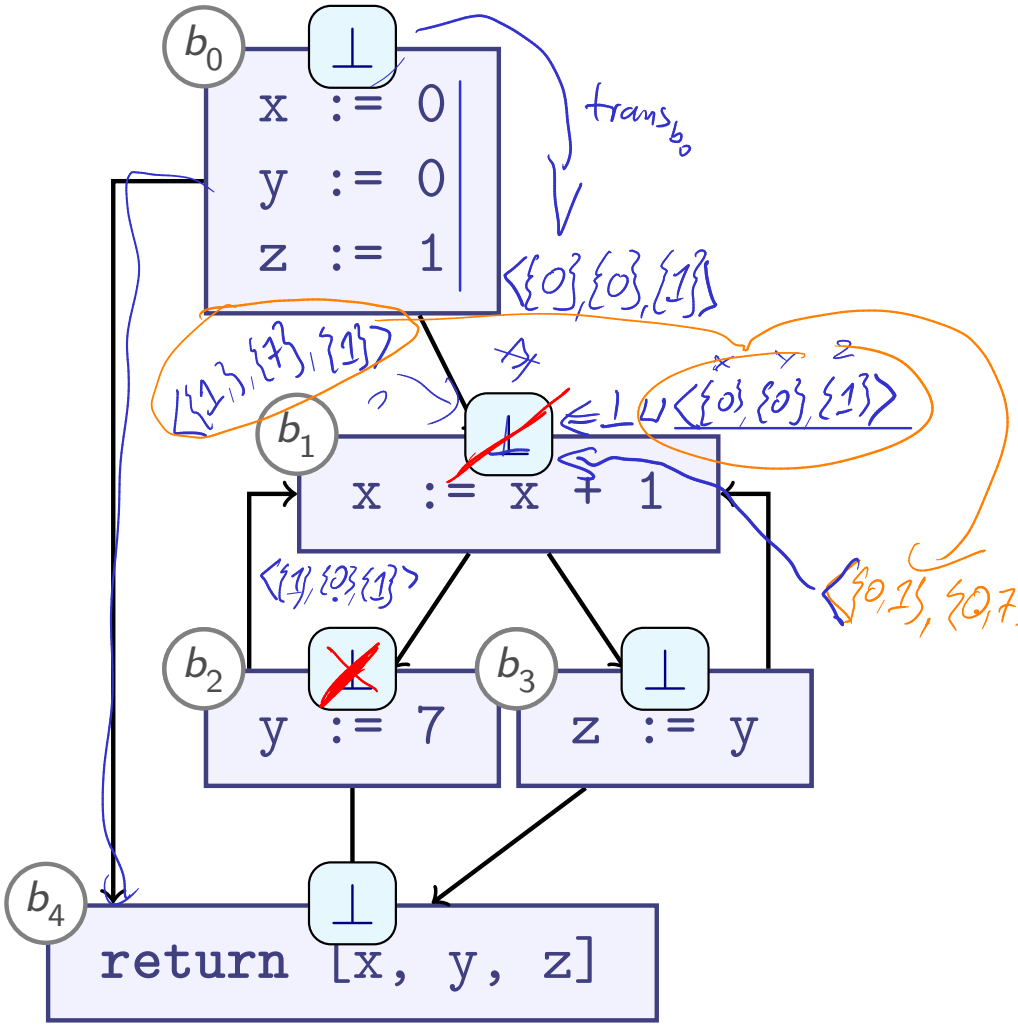
Example:



b	inputs	$trans_b$		
		x	y	z
b_0	$\emptyset$	0	0	1
b_1	$\{b_0, b_2, b_3\}$	$x+1$	y	z
b_2	$\{b_1\}$	x	7	z
b_3	$\{b_1\}$	x	y	y
b_4	$\{b_0, b_2, b_3\}$	x	y	z

$$join_{b_i}(\langle v_{x_1}, v_{y_1}, v_{z_1}\rangle, \langle v_{x_2}, v_{y_2}, v_{z_2}\rangle) = \langle v_{x_1} \cup v_{x_2}, v_{y_1} \cup v_{y_2}, v_{z_1} \cup v_{z_2}\rangle$$

Example:



b	inputs	$trans_b$		
		x	y	z
b_0	$\emptyset$	0	0	1
b_1	$\{b_0, b_2, b_3\}$	$x+1$	y	z
b_2	$\{b_1\}$	x	7	z
b_3	$\{b_1\}$	x	y	y
b_4	$\{b_0, b_2, b_3\}$	x	y	z

$$join_{b_i}(\langle v_{x_1}, v_{y_1}, v_{z_1}\rangle, \langle v_{x_2}, v_{y_2}, v_{z_2}\rangle) = \langle v_{x_1} \cup v_{x_2}, v_{y_1} \cup v_{y_2}, v_{z_1} \cup v_{z_2}\rangle$$

Worklist

~~$b_0 \to b_1$~~
$b_0 \to b_4$
$b_1 \to b_2$
$b_1 \to b_3$
$b_2 \to b_4$
~~$b_2 \to b_1$~~
$b_3 \to b_4$
$b_3 \to b_1$

The MFP Algorithm

```
Procedure MFP(⊥, ⊔, ⊑, CFG, trans_, is-backward):
begin
  if is-backward then reverse edges(CFG);
  worklist := edges(CFG); -- edges that we need to look at
  foreach n ∈ nodes(CFG) do
    ~~out~~ in[n] = ⊥;            -- state of the analysis
  done
  while not empty(worklist) do
    ⟨n, n′⟩ := pop(worklist);   -- Edge n → n′
    if trans_n(~~out~~ in[n]) ⋢ ~~out~~ in[n′] then begin
      ~~out~~ in[n′] := ~~out~~ in[n′] ⊔ trans_n(~~out~~ in[n]);
      foreach n″ ∈ successor-nodes(CFG, n′) do
        push(worklist, ⟨n′, n″⟩);
      done
    end
  done
  return out;
end
```

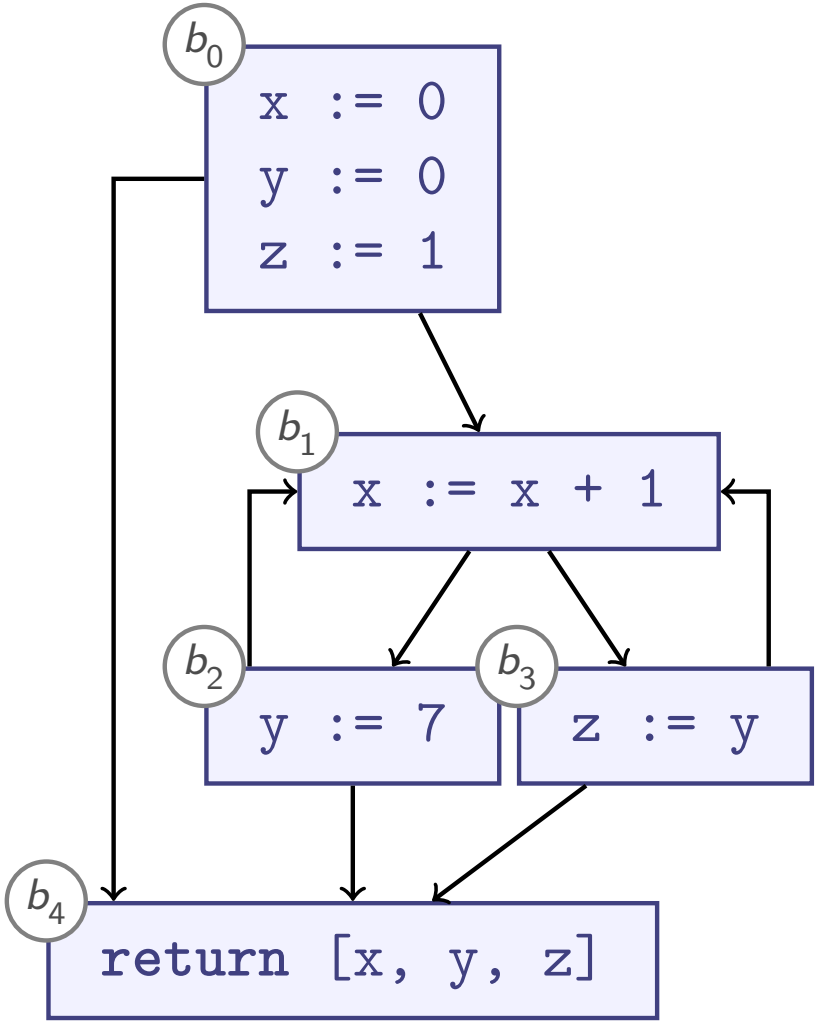
Worklist allows focussing effort!

Summary: MFP Algorithm

- **Product Lattice** allows analysing multiple variables at once
- Compute data flow analysis:
  - Initialise all nodes with $\top$ $\bot$
  - Repeat until nothing changes any more:
    - Apply transfer function
    - Propagate changes along control flow graph
    - Apply $\sqcap$ $\sqcup$
- Compute ~~maximal~~ a **fixpoint**
- Use **worklist** to increase efficiency
- Distinction: Forward/Backward analyses

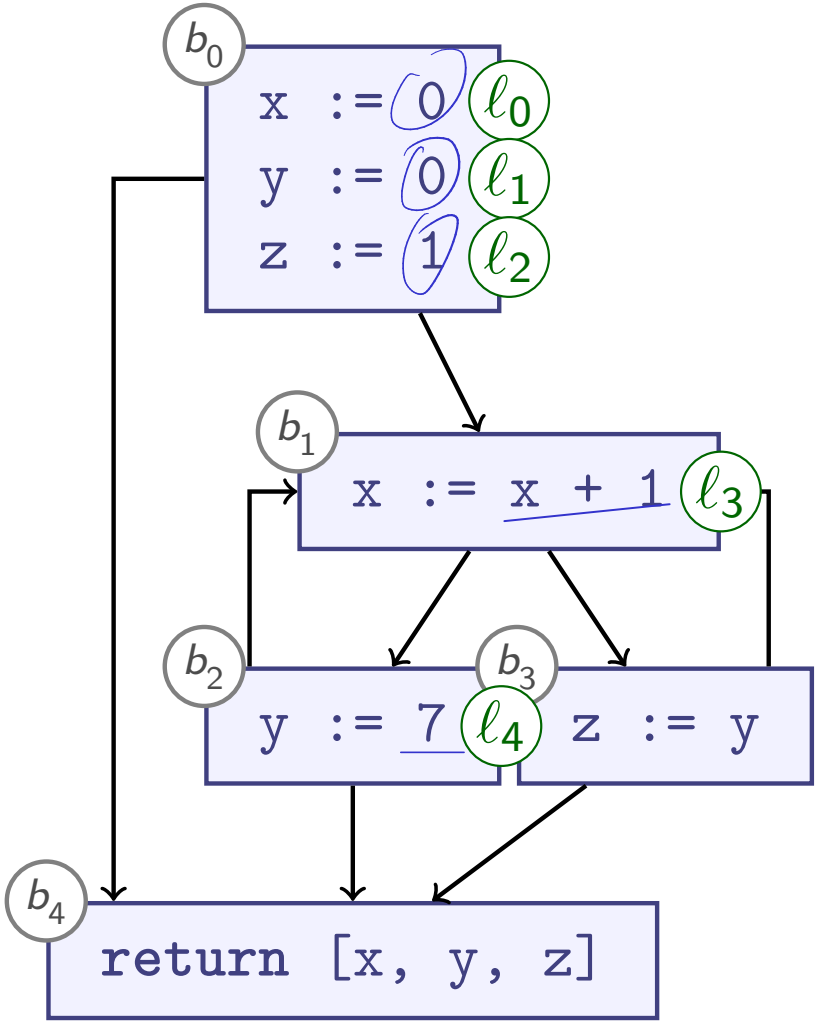
MFP revisited

Consider **Reaching Definitions** again, with different lattice:



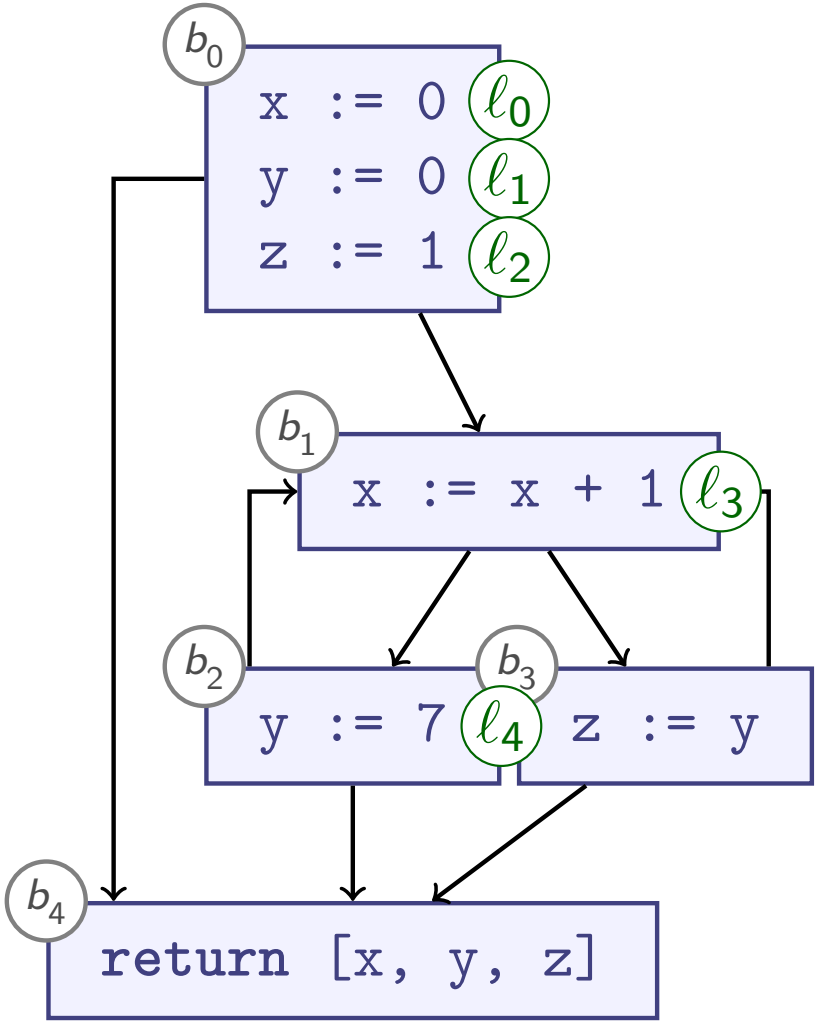
MFP revisited

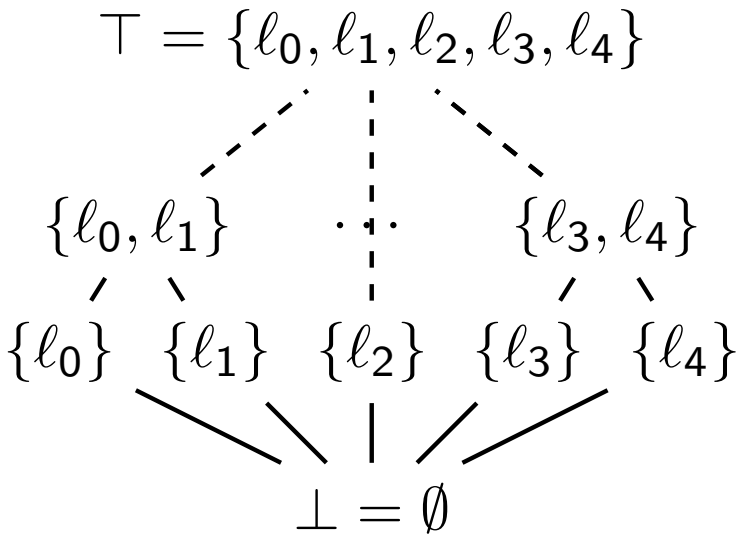
Consider **Reaching Definitions** again, with different lattice:



MFP revisited

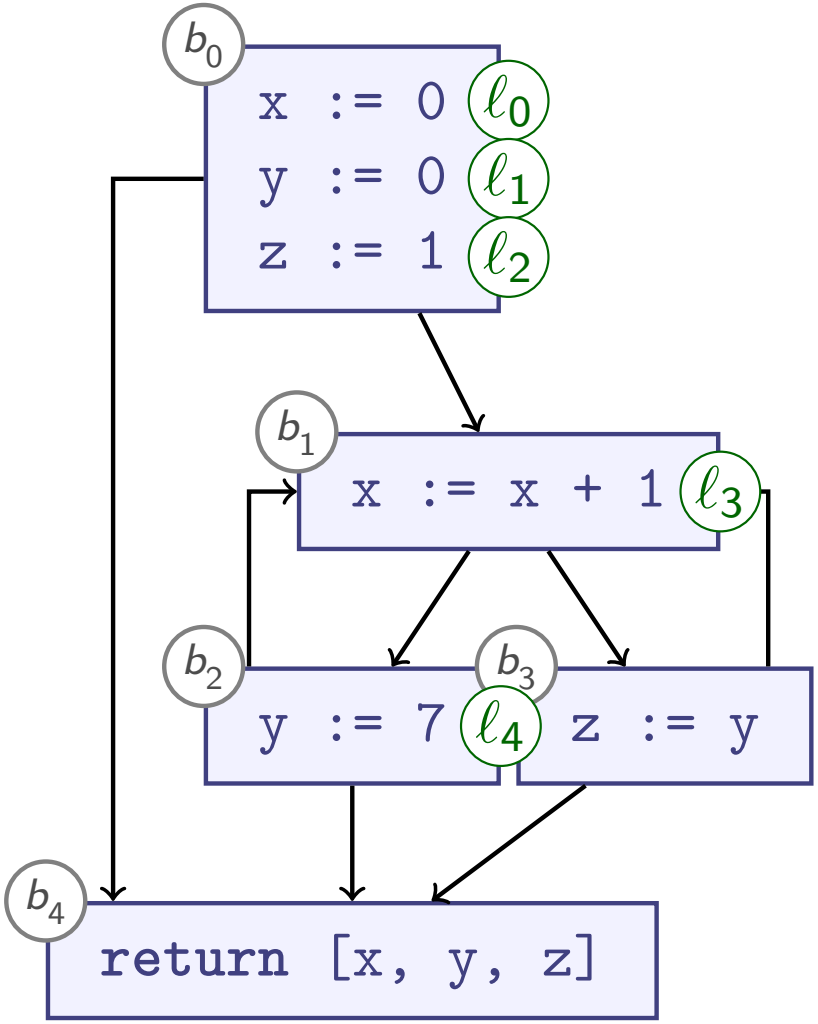
Consider **Reaching Definitions** again, with different lattice:





- All subsets of $\{\ell_0, \ldots, \ell_4\}$
- Finite height
- $\sqcup = \cup$

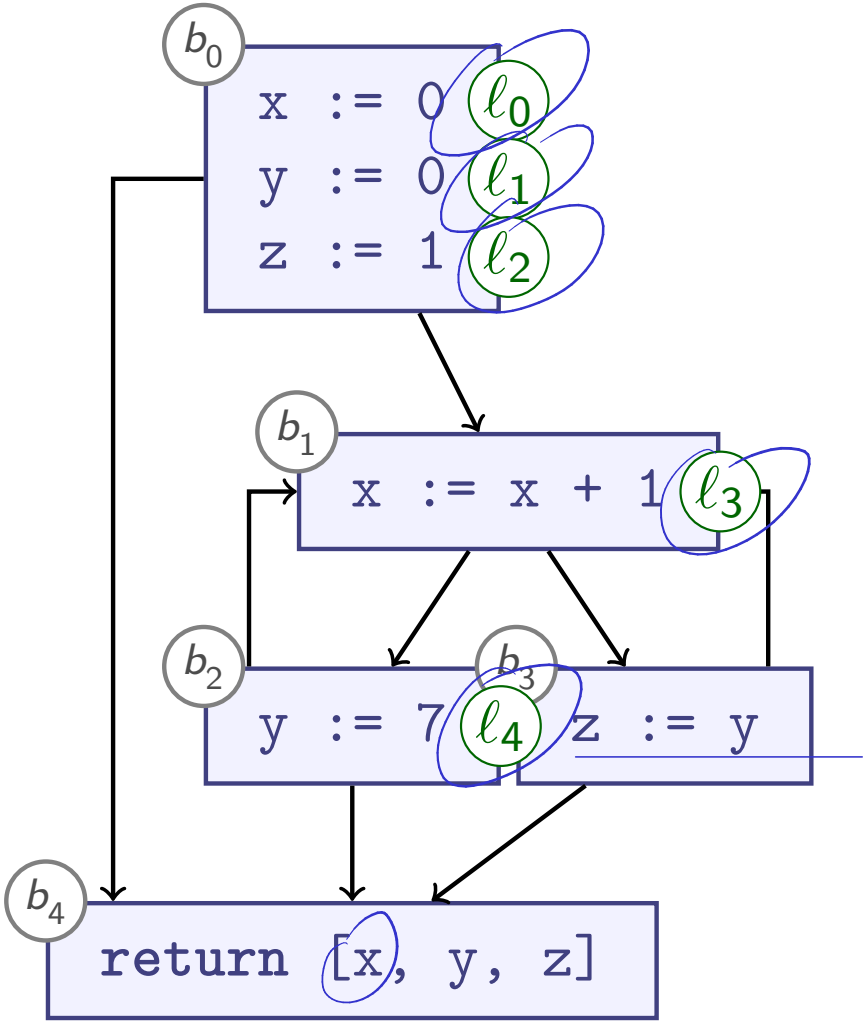
MFP revisited: Transfer Functions



$$
\begin{aligned}
trans_{b_0} = [\ & x \mapsto \{\ell_0\}, \\
& y \mapsto \{\ell_1\}, \\
& z \mapsto \{\ell_2\}] \\
trans_{b_1} = [\ & x \mapsto \{\ell_3\}] \\
trans_{b_2} = [\ & y \mapsto \{\ell_4\}] \\
trans_{b_3} = [\ & z \mapsto y]
\end{aligned}
$$

MFP solution

MFP revisited: Transfer Functions



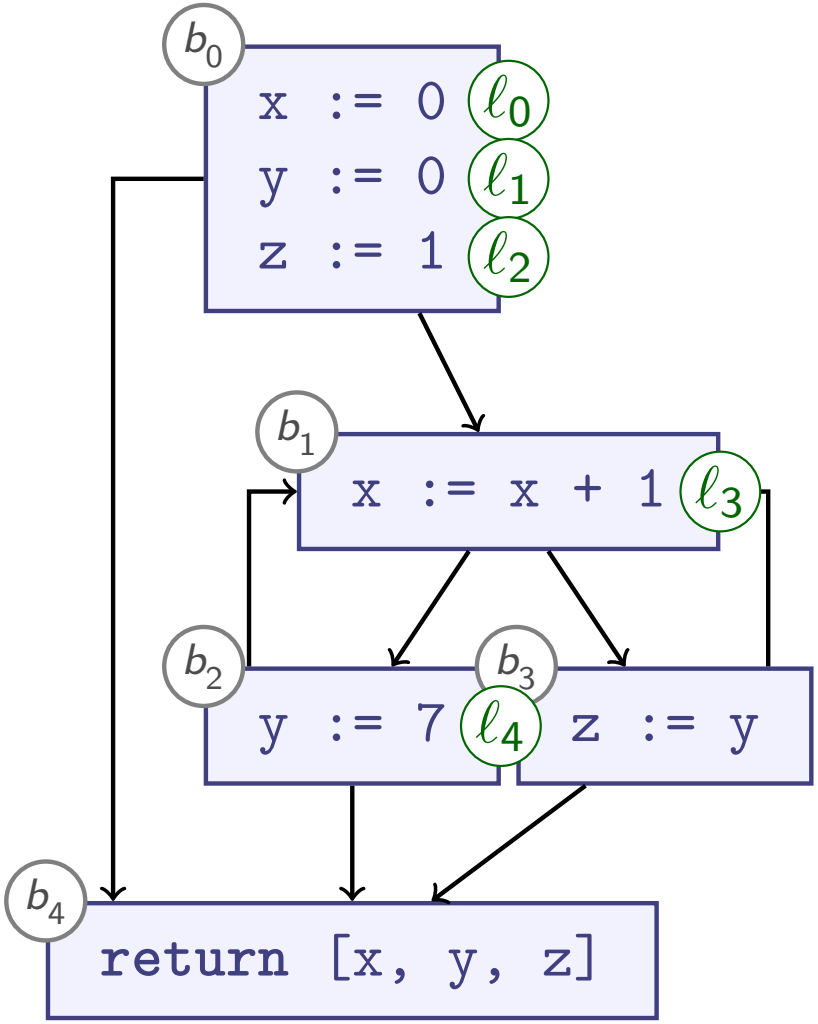
$$
\begin{aligned}
trans_{b_0} &= [\ x \mapsto \{\ell_0\},\\
&\quad\ \ y \mapsto \{\ell_1\},\\
&\quad\ \ z \mapsto \{\ell_2\}]\\
trans_{b_1} &= [x \mapsto \{\ell_3\}]\\
trans_{b_2} &= [y \mapsto \{\ell_4\}]\\
trans_{b_3} &= [z \mapsto y]
\end{aligned}
$$

MFP solution

$$
\begin{aligned}
x &\mapsto \{\ell_0, \ell_3\}\\
y &\mapsto \{\ell_1, \ell_4\}\\
z &\mapsto \{\ell_1, \ell_2, \ell_4\}
\end{aligned}
$$

- Least Fixpoint!

MFP revisited: Transfer Functions



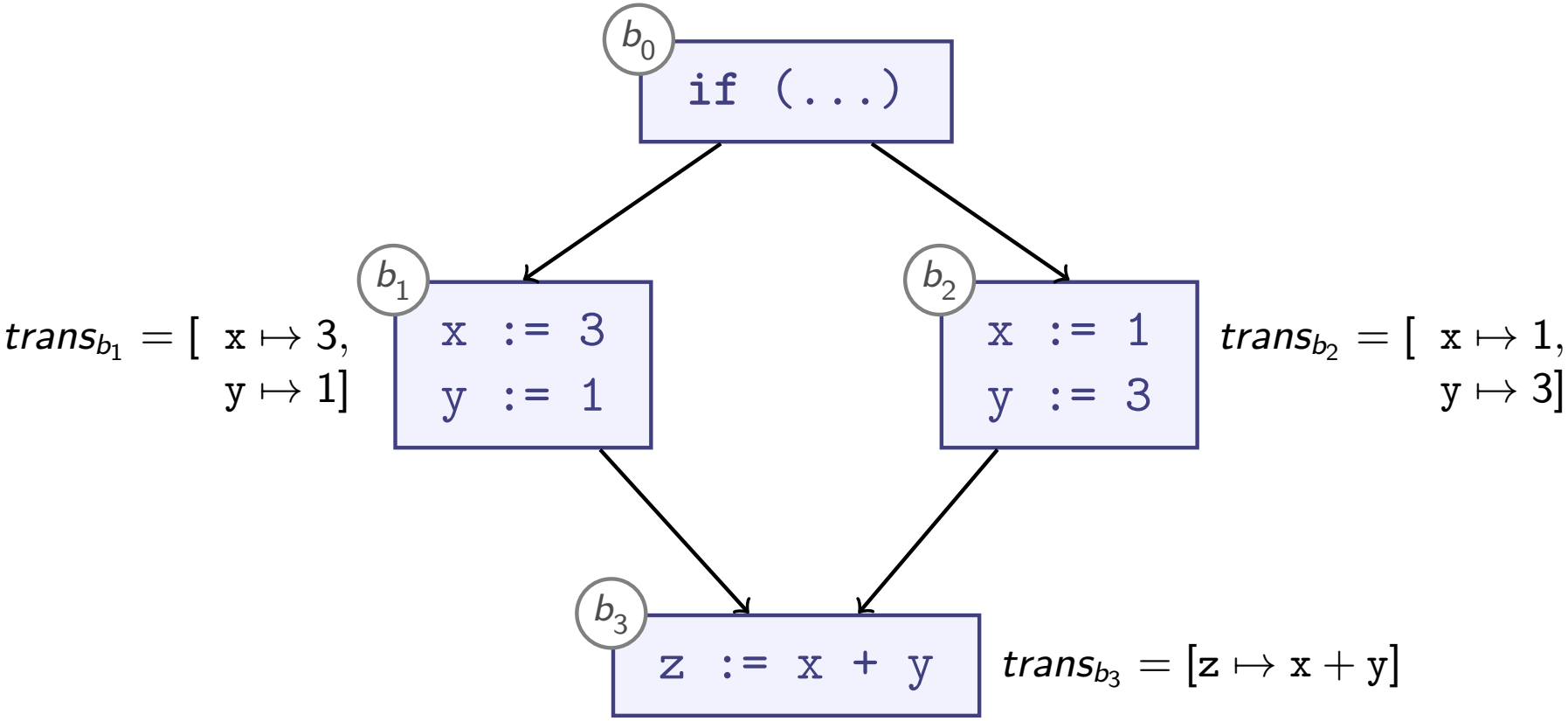
$$
\begin{aligned}
trans_{b_0} = [\ & x \mapsto \{\ell_0\}, \\
& y \mapsto \{\ell_1\}, \\
& z \mapsto \{\ell_2\}] \\
trans_{b_1} = [& x \mapsto \{\ell_3\}] \\
trans_{b_2} = [& y \mapsto \{\ell_4\}] \\
trans_{b_3} = [& z \mapsto y]
\end{aligned}
$$

MFP solution

$$
\begin{aligned}
x &\mapsto \{\ell_0, \ell_3\} \\
y &\mapsto \{\ell_1, \ell_4\} \\
z &\mapsto \{\ell_1, \ell_2, \ell_4\}
\end{aligned}
$$

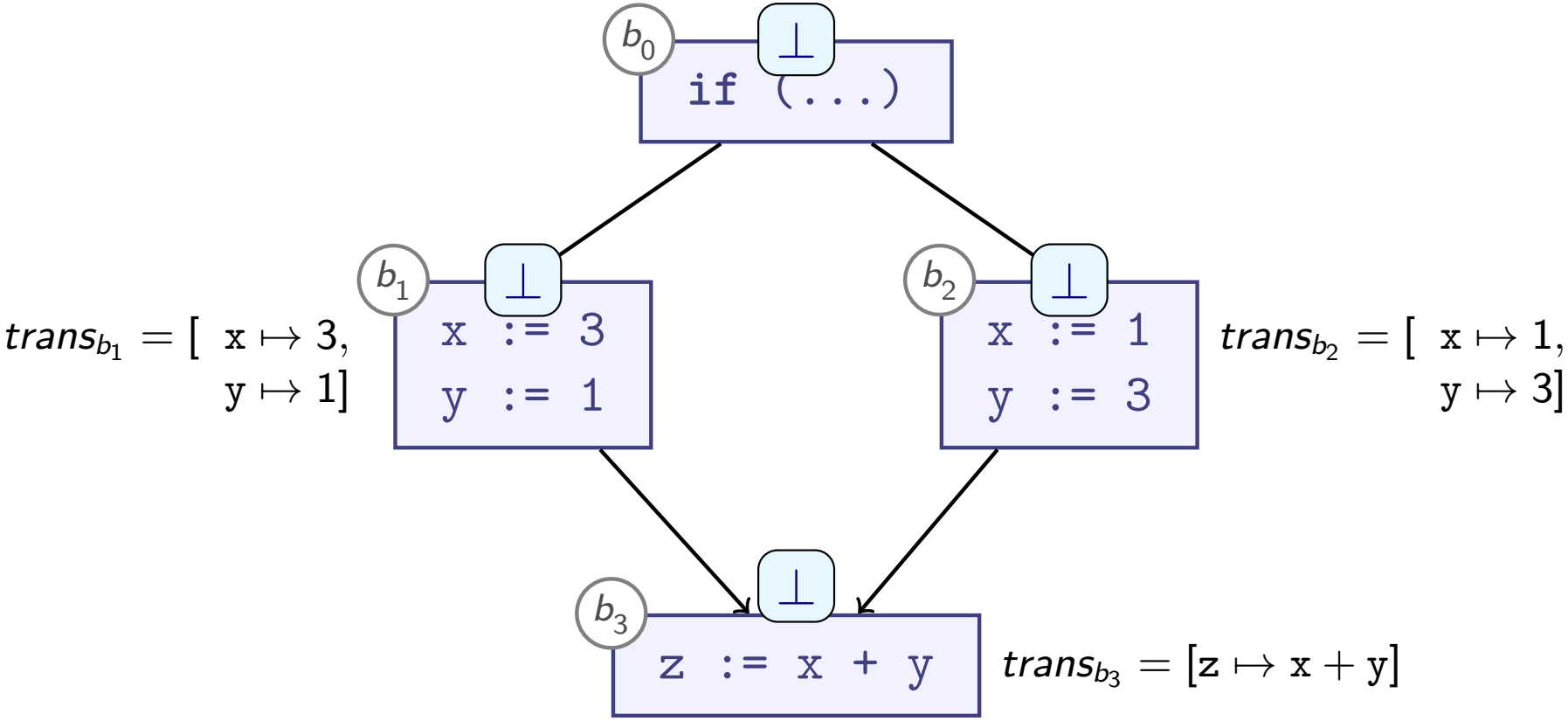
- Least Fixpoint!
- Do we always get LFP from MFP?

Another Example



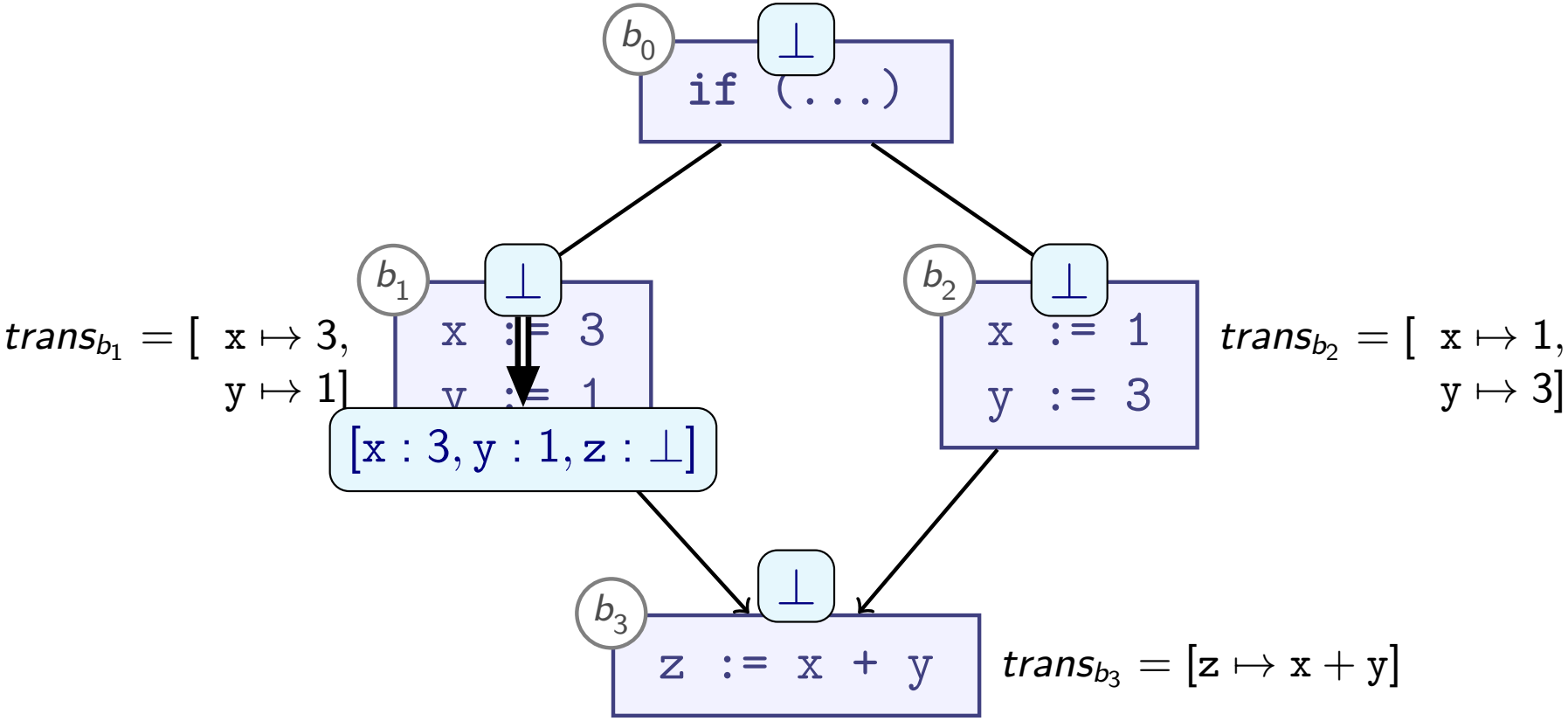
- Lattice: $\mathbb{Z}_\bot^\top$

Another Example



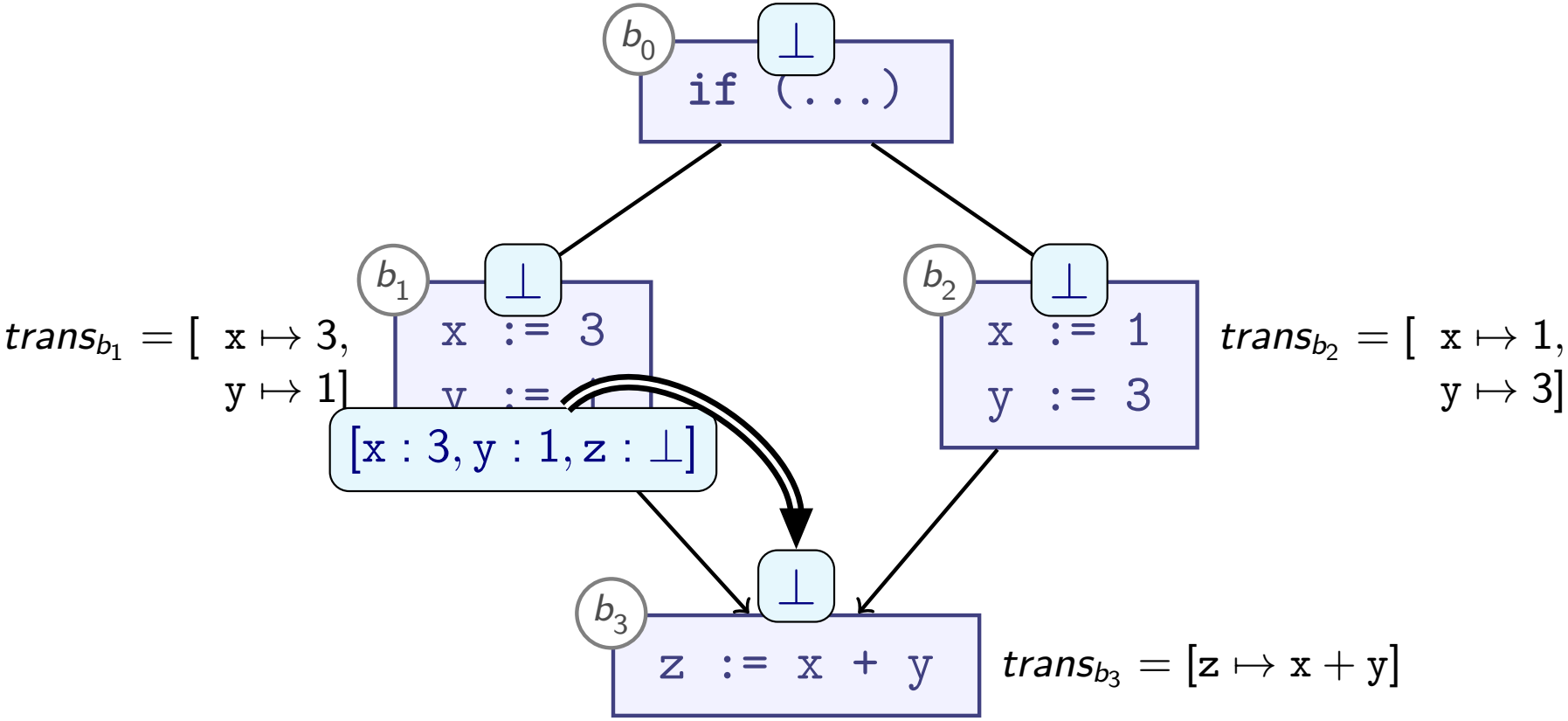
- Lattice: $\mathbb{Z}_{\bot}^{\top}$

Another Example



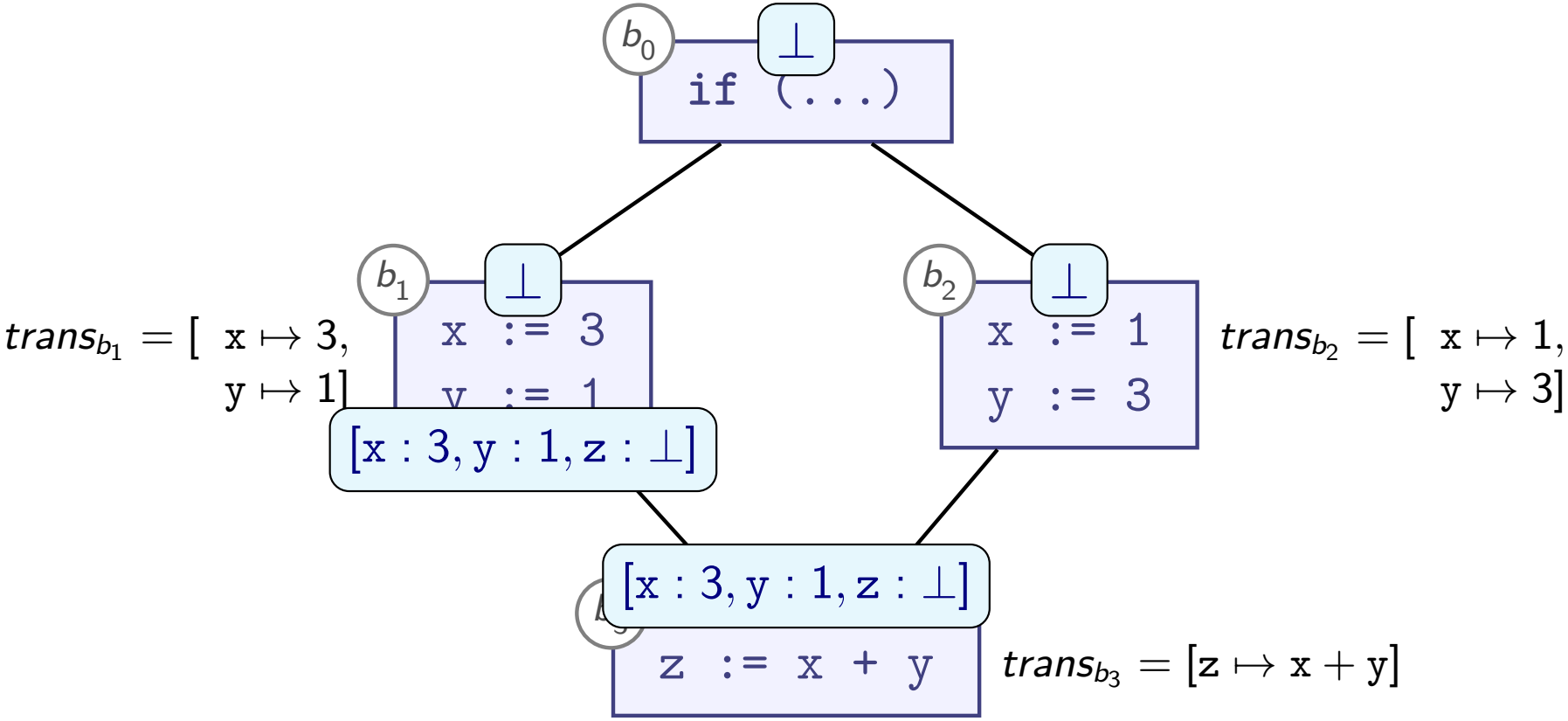
- Lattice: $\mathbb{Z}_\bot^\top$

Another Example



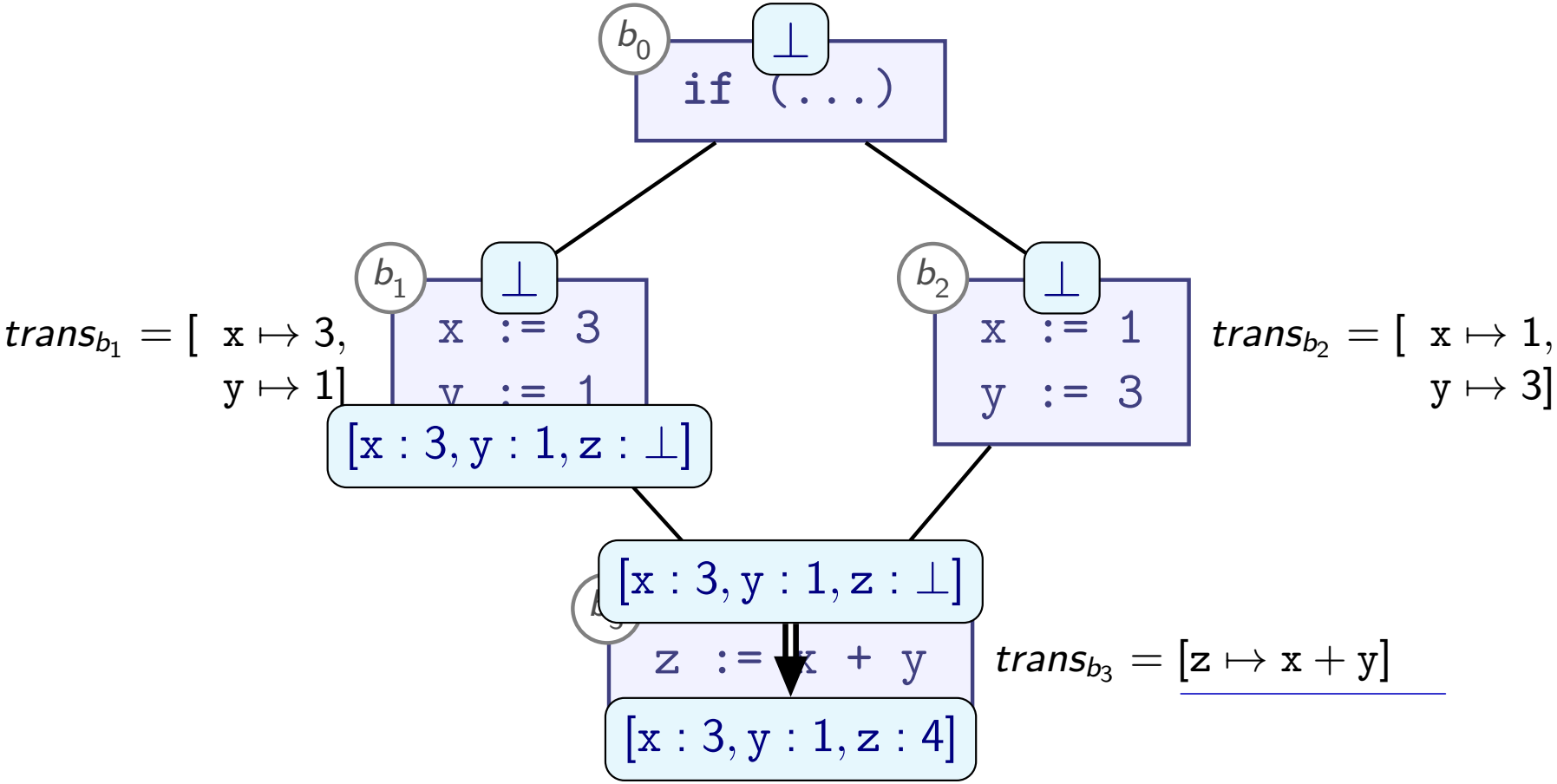
- Lattice: $\mathbb{Z}_{\bot}^{\top}$

Another Example



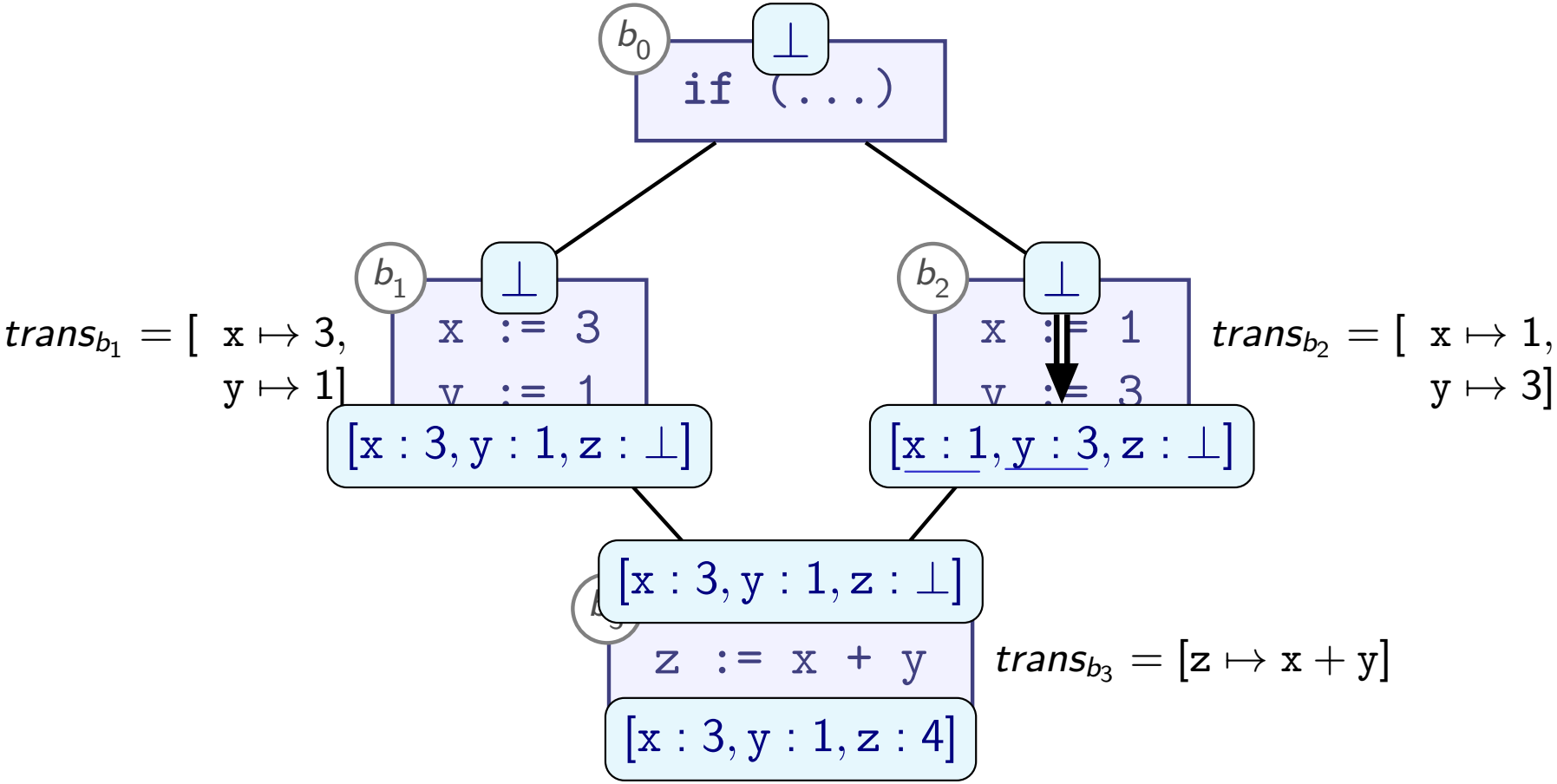
- Lattice: $\mathbb{Z}_\bot^\top$

Another Example



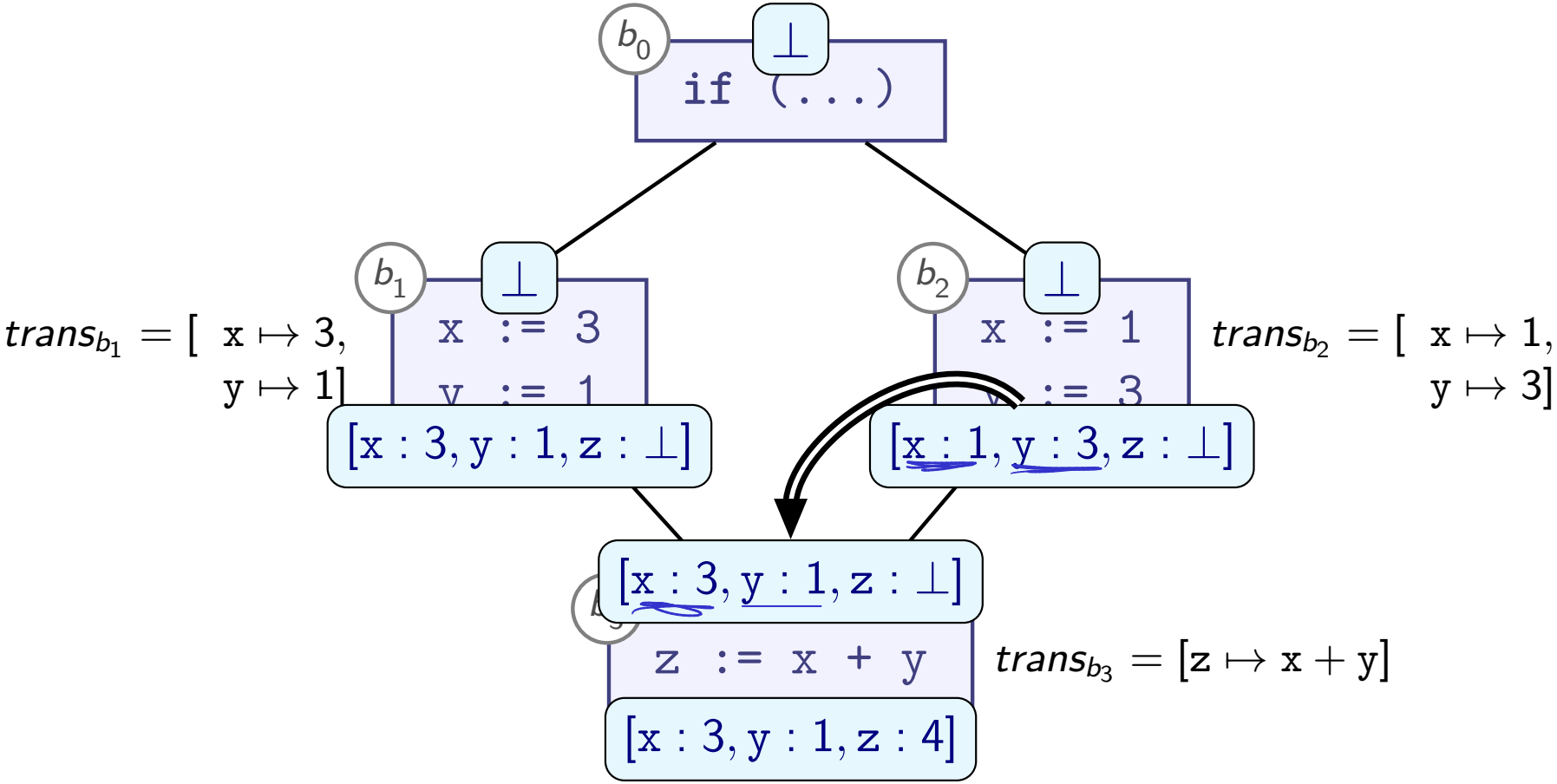
- Lattice: $\mathbb{Z}_\bot^\top$

Another Example



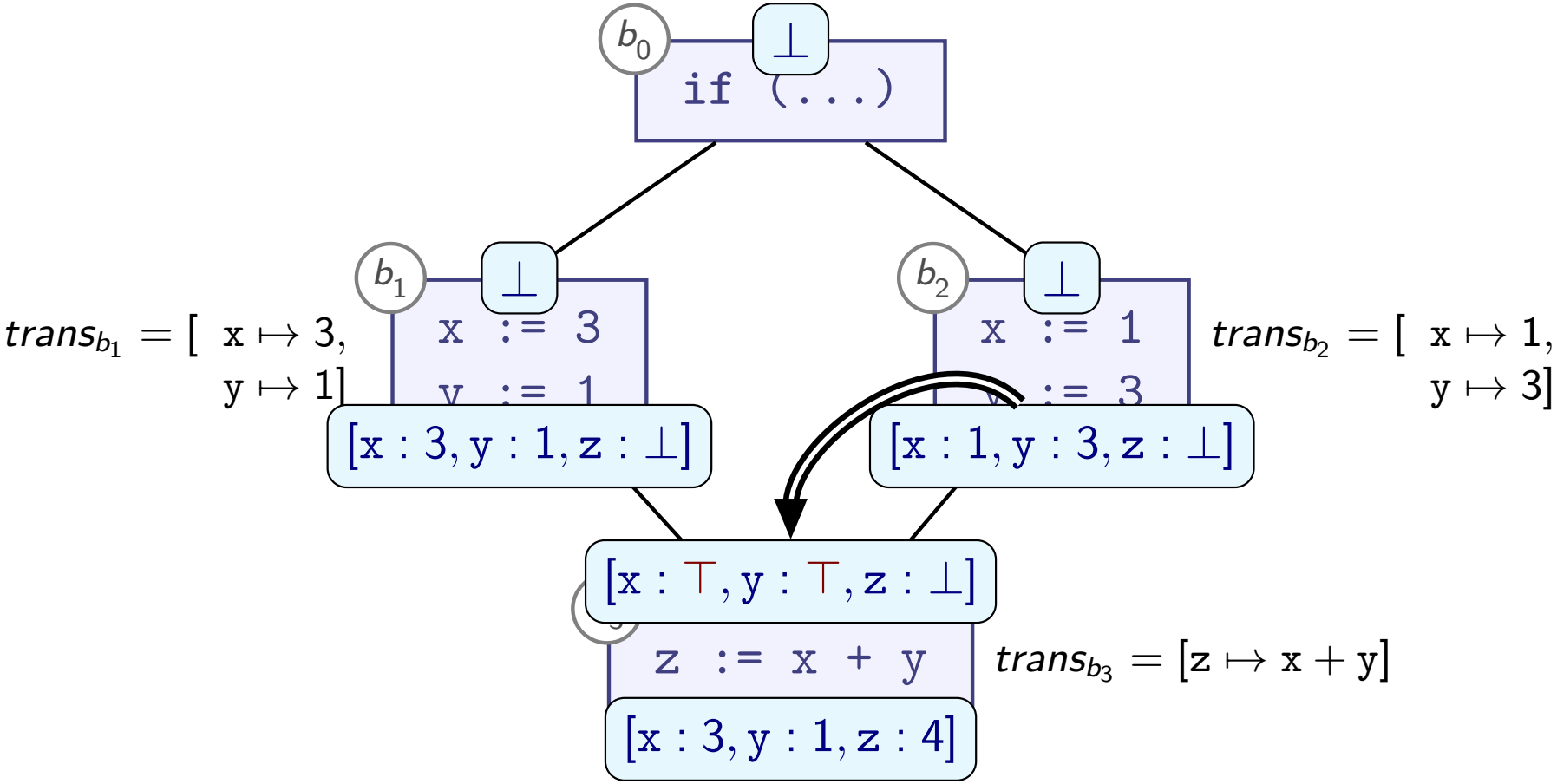
- Lattice: $\mathbb{Z}_{\bot}^{\top}$

Another Example



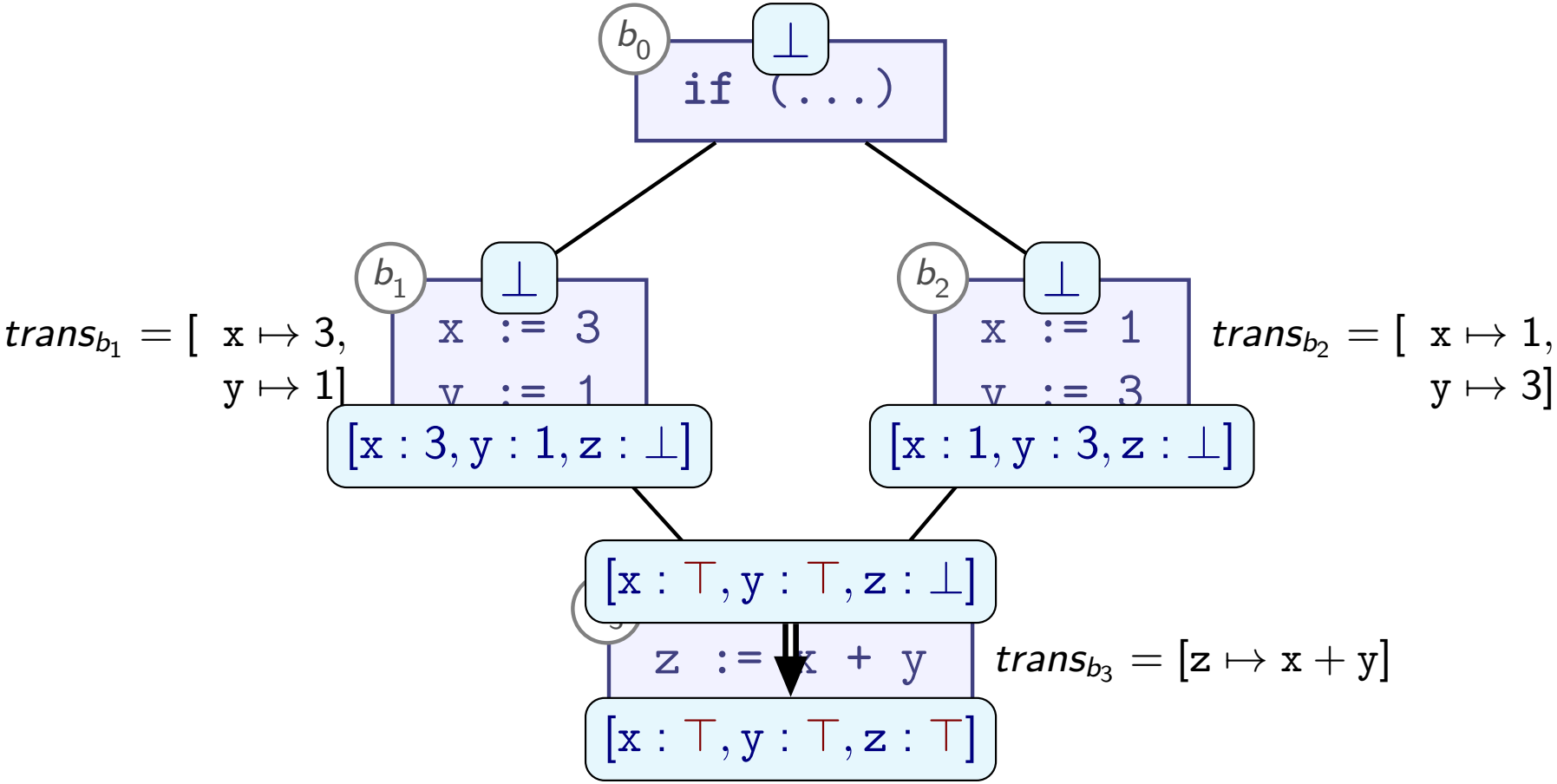
- Lattice: $\mathbb{Z}_{\bot}^{\top}$
  - $1 \sqcup 3 = \top = 3 \sqcup 1$

Another Example



- Lattice: $\mathbb{Z}_{\bot}^{\top}$
  - $1 \sqcup 3 = \top = 3 \sqcup 1$

Another Example



- Lattice: $\mathbb{Z}_{\bot}^{\top}$
  - $1 \sqcup 3 = \top = 3 \sqcup 1$
- **No**, MFP does not always compute the Least Fixpoint!

Distributive Frameworks

A <u>*Monotone Framework*</u> is:

- Lattice $L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
- L has finite height (Ascending Chain Condition)
- All $trans_b$ are monotonic

Distributive Frameworks

A <u>*Monotone Framework*</u> is:

- Lattice $L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
- L has finite height (Ascending Chain Condition)
- All $trans_b$ are monotonic
- Guarantees *a* Fixpoint

Distributive Frameworks

A *Monotone Framework* is:

- Lattice $L = \langle \mathcal{L}, \sqsubseteq, \sqcap, \sqcup \rangle$
- L has finite height (Ascending Chain Condition)
- All $trans_b$ are monotonic
- Guarantees *a* Fixpoint

A *Distributive Framework* is:

- A Monotone Framework, where additionally:
- $trans_b$ distributes over $\sqcup$:

$$trans_b(x \sqcup y) = \underline{trans_b(x)} \sqcup \underline{trans_b(y)}$$

  for all programs and all x, y, b
- Guarantees that MFP gives Least Fixpoint

Distributive Problems

- Monotonic:

$$trans_b(x \sqcup y) \sqsupseteq trans_b(x) \sqcup trans_b(y)$$

- Distributive:

$$trans_b(x \sqcup y) = trans_b(x) \sqcup trans_b(y)$$

Distributive Problems

- Monotonic:

$$trans_b(x \sqcup y) \sqsupseteq trans_b(x) \sqcup trans_b(y)$$

- Distributive:

$$trans_b(x \sqcup y) = trans_b(x) \sqcup trans_b(y)$$

- Many analyses fit distributive framework
- Known *counter-example*: transfer functions on $\mathbb{Z}_{\perp}^{\top}$:
  - $[z \mapsto x + y]$
  - Generally:
    - depends on ≥ 2 independent inputs
    - can produce same output for different inputs

Summary

- **Distributive Frameworks** are *Monotone Frameworks* with additional property:

$$trans_b(x \sqcup y) = trans_b(x) \sqcup trans_b(y)$$

  for all programs and all x, y, b
- In Distributive Frameworks, MFP produces Least Fixpoint
- Some analyses (Gen/Kill analyses, discussed later) are always distributive