CSC410
Data Flow Analyses

AZADEH FARZAN

FALL 2023

First Structure

Partially Ordered Sets

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

Partially Ordered Sets

Information Order

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

Partially Ordered Sets

Information Order

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric

$x \sqsubseteq x$ (reflexivity).

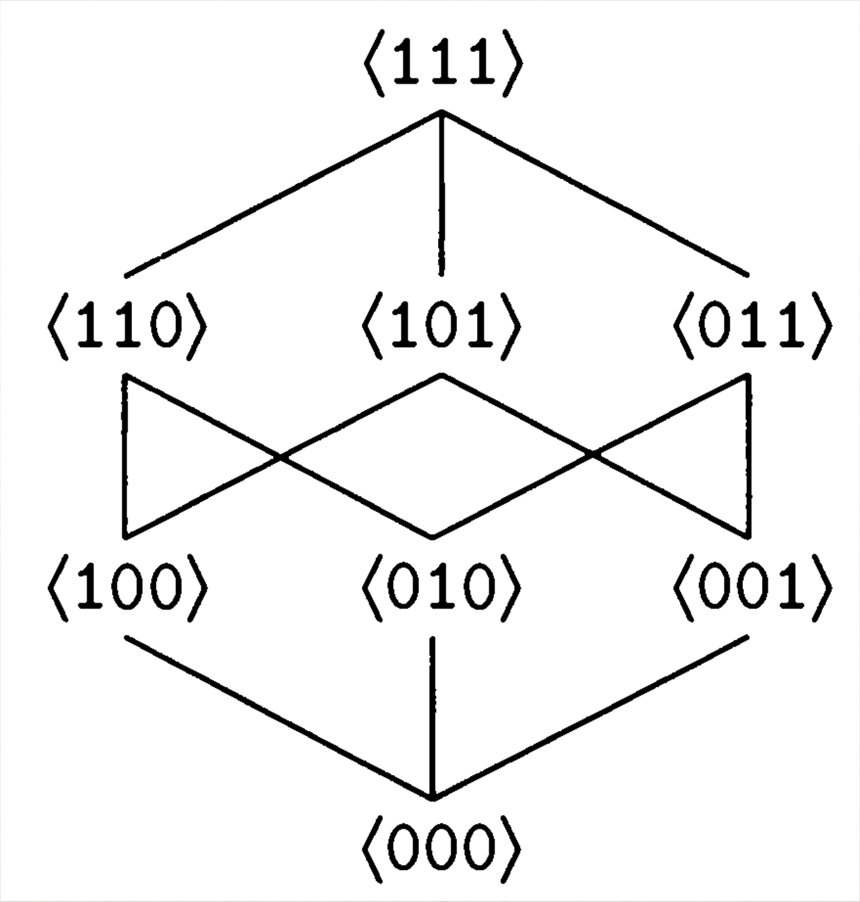
If $x \sqsubseteq y$ and $y \sqsubseteq z$, then $x \sqsubseteq z$ (transitivity).

If $x \sqsubseteq y$ and $y \sqsubseteq x$, then $x = y$ (antisymmetry).

Partially Ordered Sets

$(S, \sqsubseteq)$: a set S and a (partial) order relation $\sqsubseteq$

- $\sqsubseteq$ is reflexive, transitive, and anti-symmetric



Second Structure

Semi-Lattices

A (meet) *semi-lattice* $\mathbf{L} = (S, \sqcap)$ is a set S with a binary operation, called *meet* ($\sqcap$), that has the following properties:

(1) For all $x, y \in S$, there exist a unique $z \in S$ such that $x \sqcap y = z$ (CLOSURE).

(2) For all $x, y, z \in S$, we have

$$x \sqcap x = x \quad \text{(idempotence)}$$

$$x \sqcap y = y \sqcap x \quad \text{(commutativity)}$$

$$x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z \quad \text{(associativity)}$$

Complete Semi-Lattices

The unit for $\sqcap$ is $\top$:

$$\forall x : \; x \sqcap \top = \top \sqcap x = x$$

Complete Semi-Lattices

The unit for $\sqcap$ is $\top$:

$$\forall x : \;\; x \sqcap \top = \top \sqcap x = x$$

The meet semi-lattice is called complete if $\top \in \mathbb{L}$

The Connection

The Connection Between The Structures

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

$\sqsubseteq$ is provably a partial order relation.

$\sqcap$ is provably the greatest lower bound defined based on $\sqsubseteq$.

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

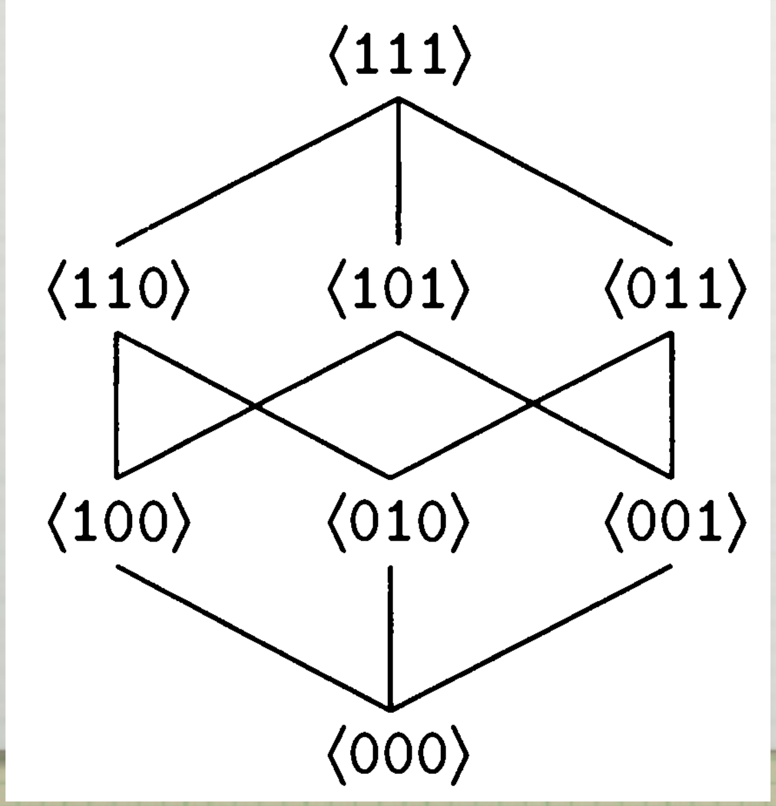
$(S, \sqcap)$ is provably a semi-lattice.

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

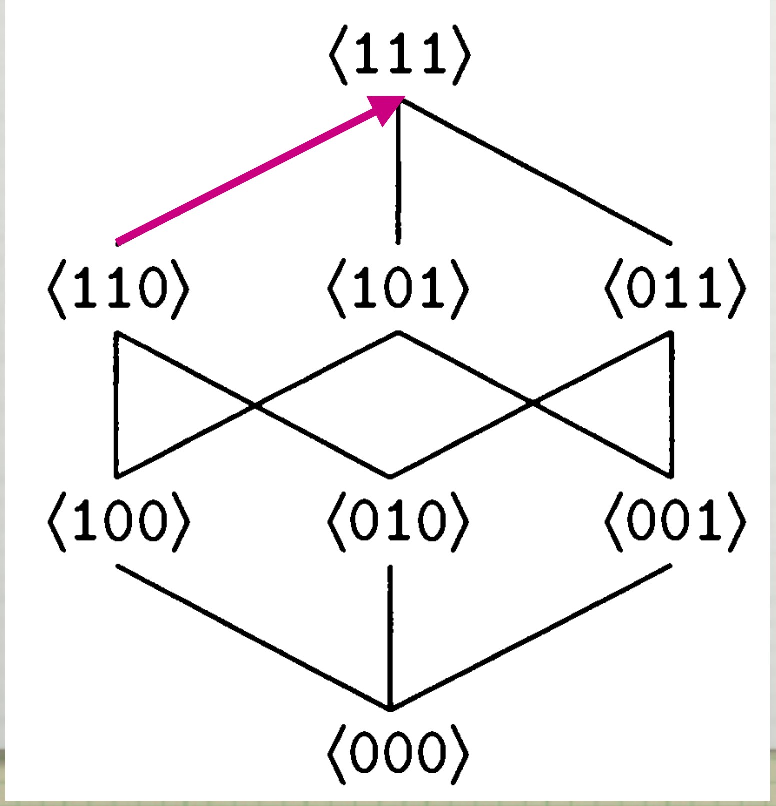


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

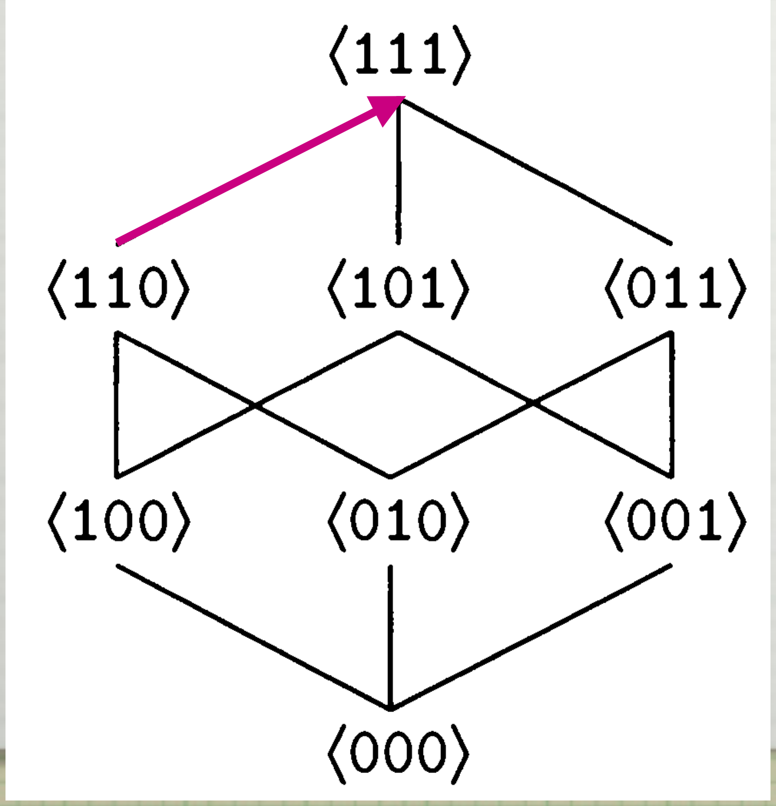


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.



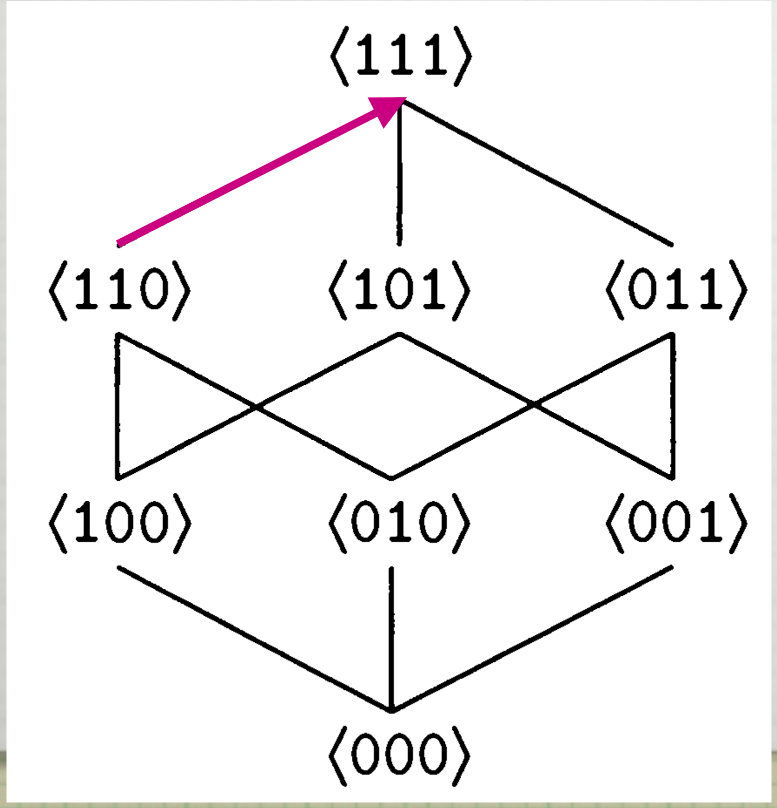
$$\sqcap = \wedge$$

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.



$$\sqcap = \wedge$$

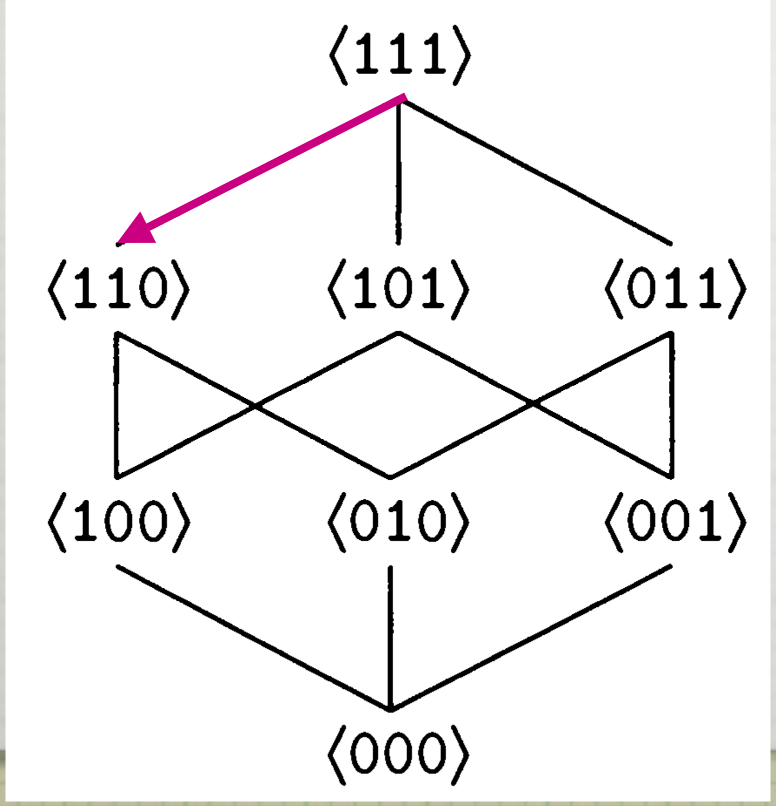
$$\langle 110\rangle \sqcap \langle 011\rangle = \langle 110\rangle \wedge \langle 011\rangle = \langle 010\rangle$$

The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.

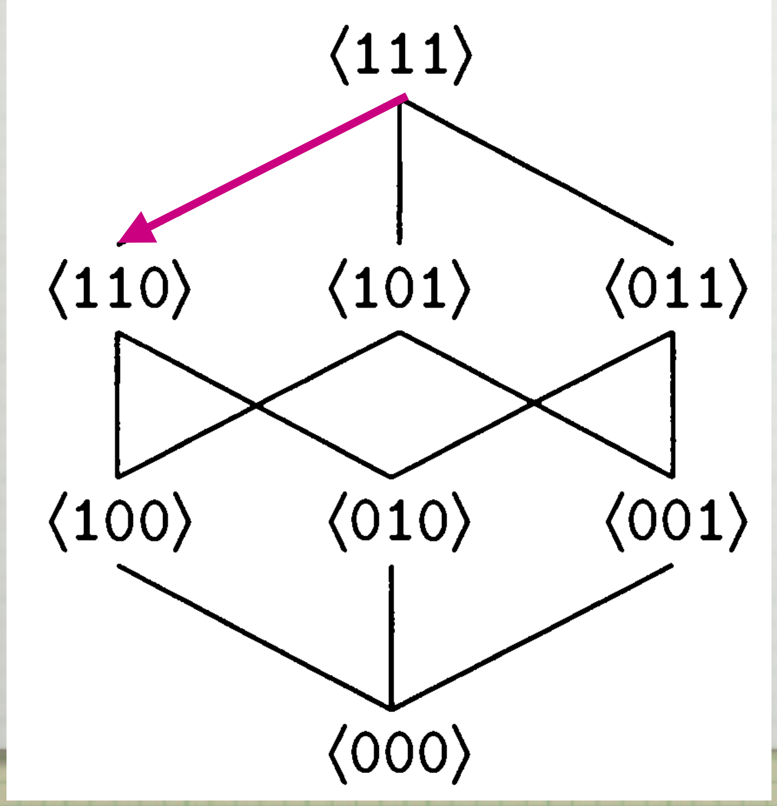


The Converse

Given a partially ordered set $(S, \sqsubseteq)$, where the greatest lower bound of every pair of elements is defined, let:

$$x \sqcap y = \text{the greatest lower bound according to } \sqsubseteq$$

$(S, \sqcap)$ is provably a semi-lattice.



$$\sqcap = \vee$$

$$\langle 110 \rangle \sqcap \langle 011 \rangle = \langle 110 \rangle \vee \langle 011 \rangle = \langle 111 \rangle$$

Theorem 1

Given a semi-lattice and define a binary operation $\sqsubseteq$:

$$x \sqsubseteq y \text{ if and only if } x \sqcap y = x$$

$\sqsubseteq$ is provably a partial order relation.

(proof on the board)

Theorem 2

$\sqcap$ is provably the greatest lower bound defined based on $\sqsubseteq$.

(proof on the board)

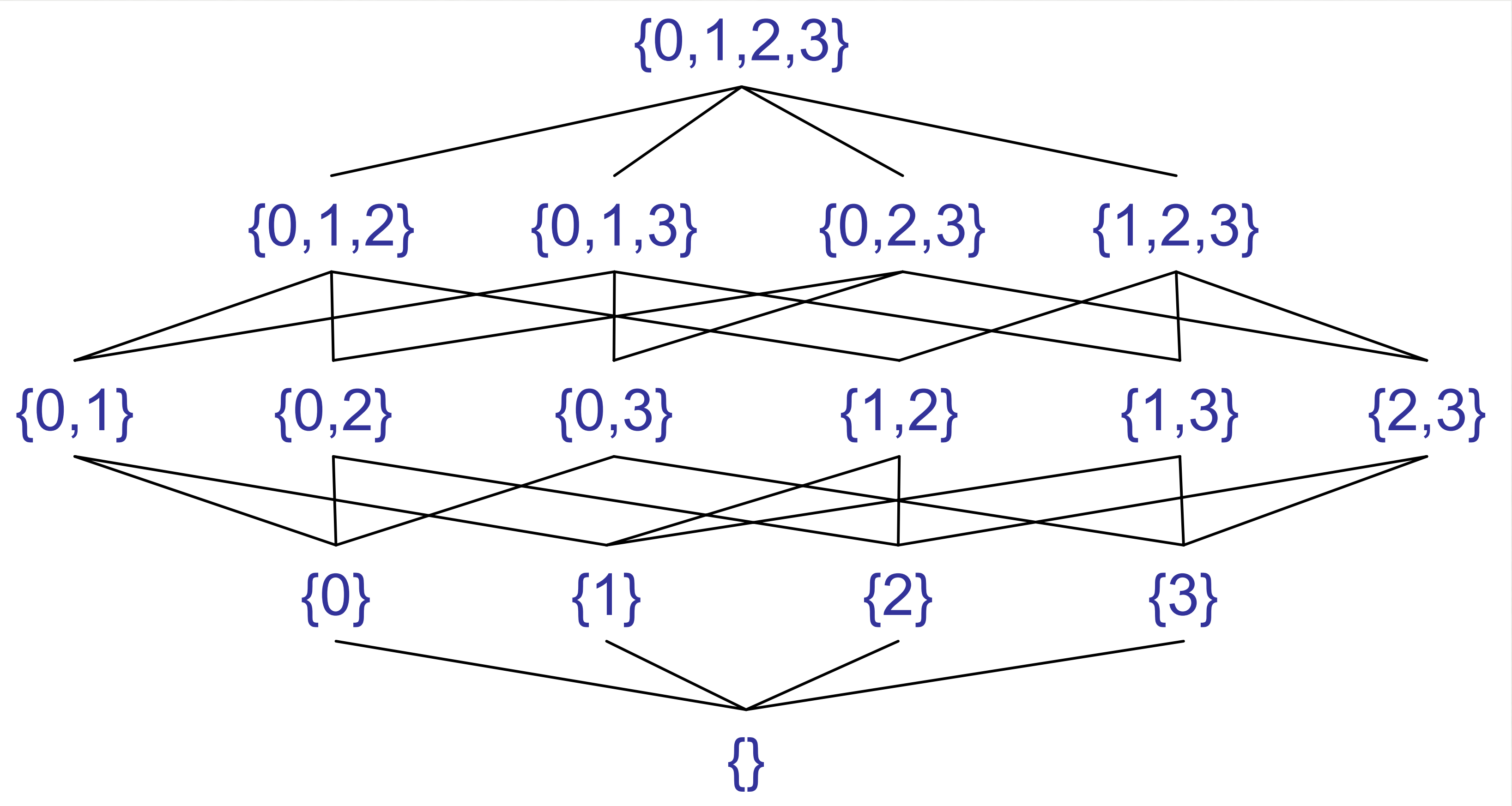
Theorem 3

Let $(S, \sqsubseteq)$ be a partially ordered set such that for all $x, y \in S$ the greatest lower bound of x and y is always defined (and in S). Prove $(S, \sqcap)$ to be a semi-lattice if:

$$x \sqcap y = glb(x, y)$$

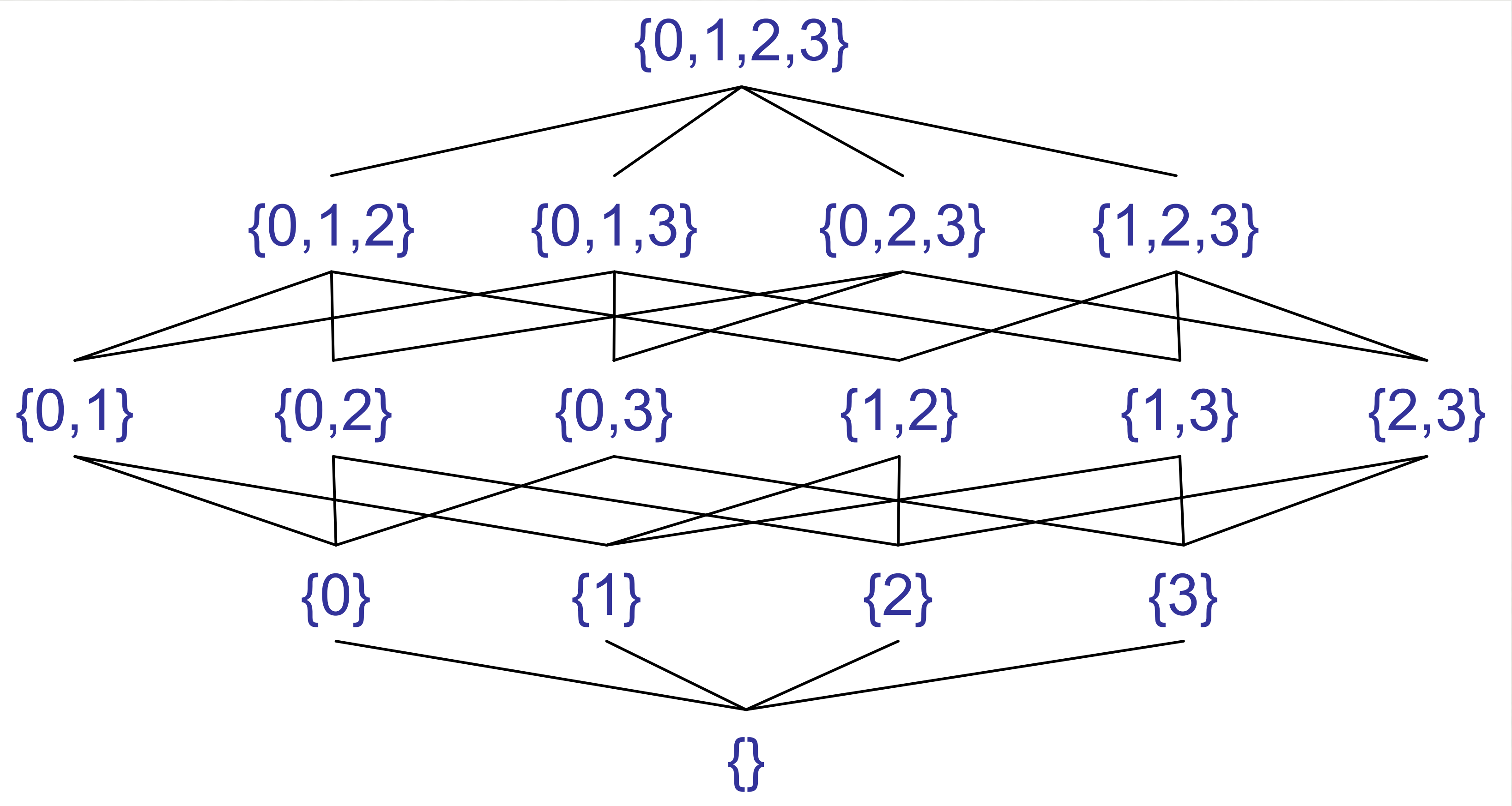
(proof on the board)

Example: Subset SemiLattice



$(\mathcal{P}(\mathcal{S}), \cap)$ is a complete meet semi-lattice.

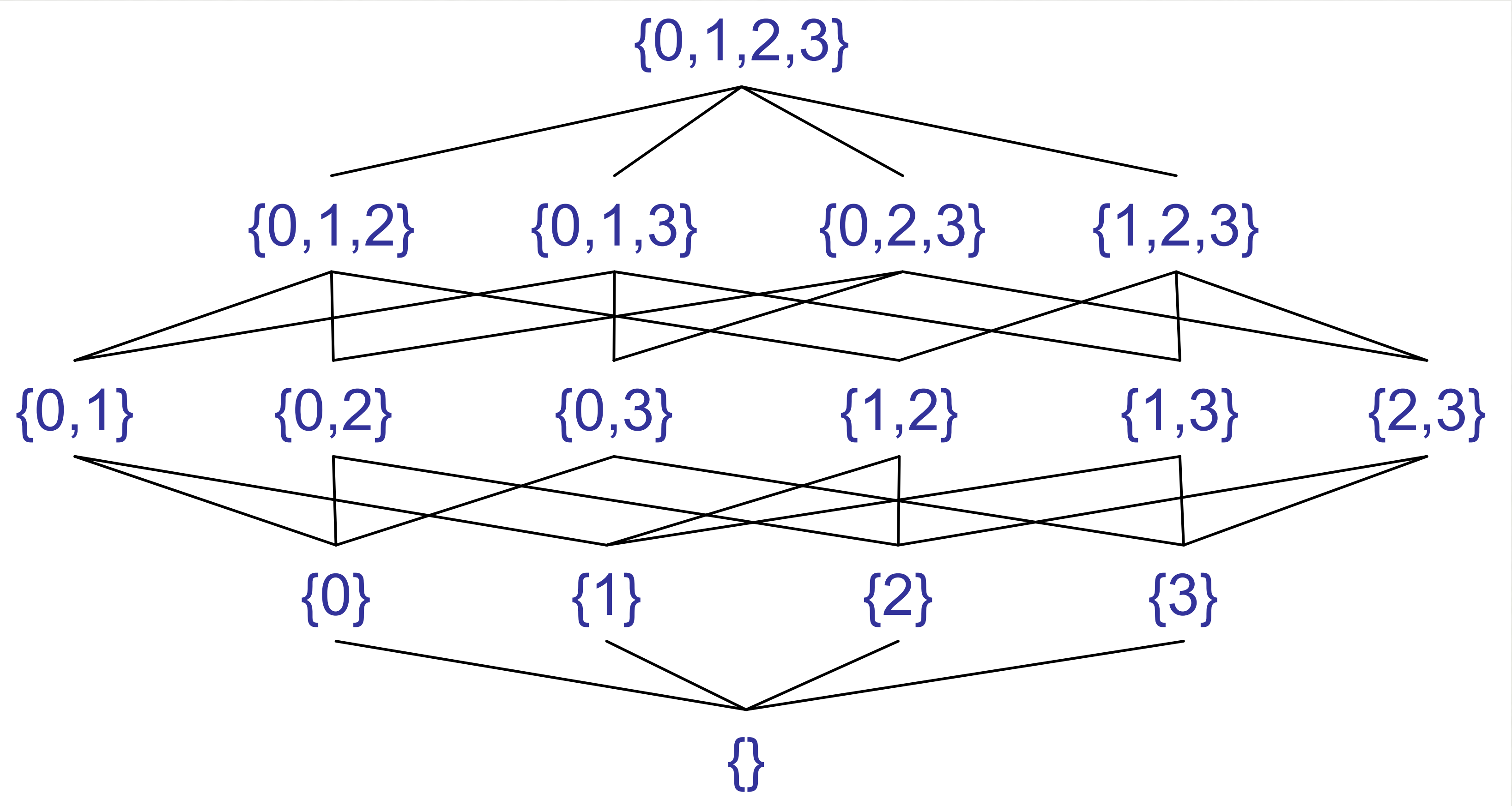
Example: Subset SemiLattice



$(\mathcal{P}(\mathcal{S}), \cap)$ is a complete meet semi-lattice.

$$\sqsubseteq = \subseteq$$

Example: Subset SemiLattice



$(\mathcal{P}(\mathcal{S}), \cap)$ is a complete meet semi-lattice.

$\sqsubseteq = \subseteq \qquad \top = S$

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LVe_{exit}(l)$$

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LVe_{exit}(l)$$

Our domain is a subset lattice where S is the set of all variables!

Descending Chains

A descending chain is a sequence of elements related by the order:

$$x_1 \sqsupseteq x_2 \sqsupseteq \cdots \sqsupseteq x_n$$

Descending Chains

A descending chain is a sequence of elements related by the order:

$$x_1 \sqsupseteq x_2 \sqsupseteq \cdots \sqsupseteq x_n$$

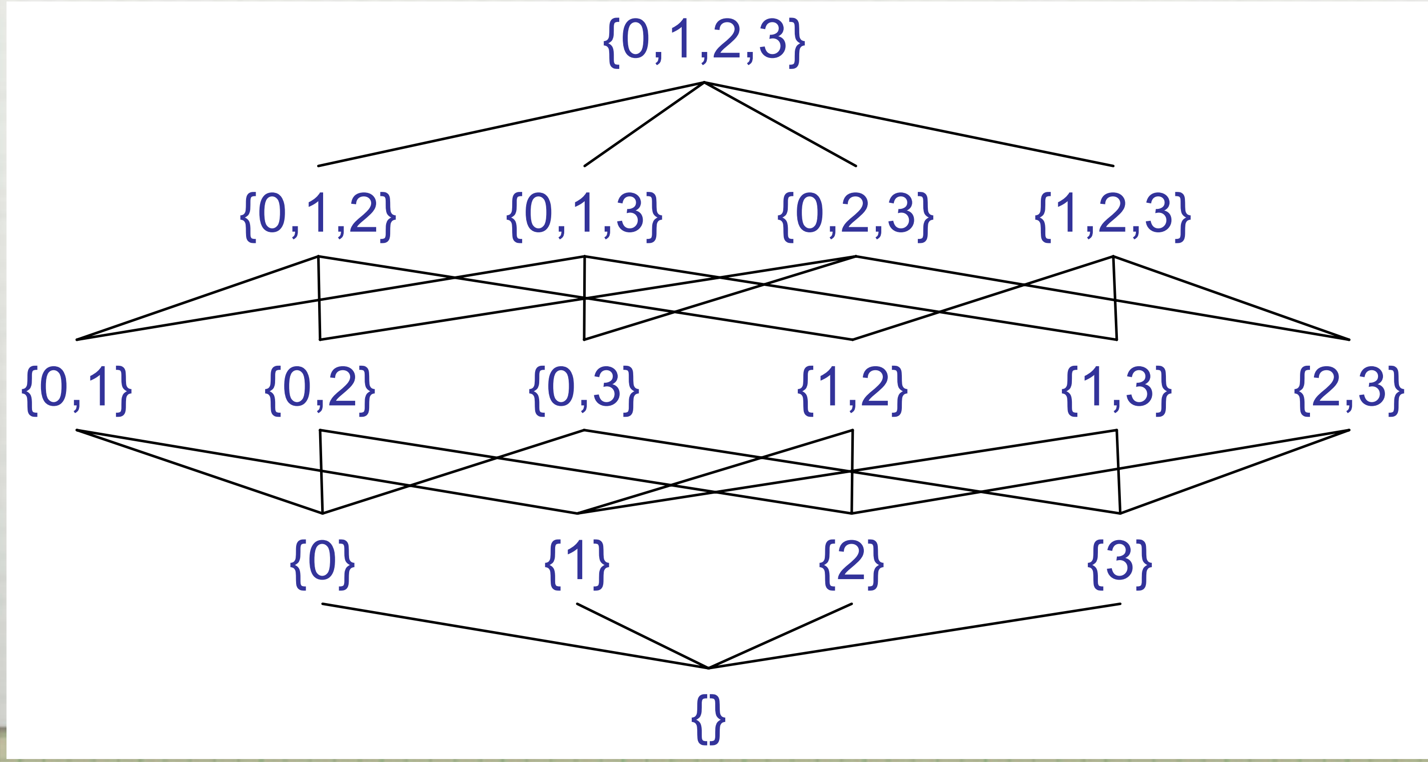
The height of a lattice (or semi-lattice) is the length of its longest descending chain.

Descending Chains

A descending chain is a sequence of elements related by the order:

$$x_1 \sqsupseteq x_2 \sqsupseteq \cdots \sqsupseteq x_n$$

The height of a lattice (or semi-lattice) is the length of its longest descending chain.



Descending Chains

A descending chain is a sequence of elements related by the order:

$$x_1 \sqsupseteq x_2 \sqsupseteq \cdots \sqsupseteq x_n$$

The height of a lattice (or semi-lattice) is the length of its longest descending chain.

Useful for Algorithmic convergence: a finite height!

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LV_{exit}(l)$$

Our domain is a subset lattice where S is the set of all variables!

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LV_{exit}(l)$$

Our domain is a subset lattice where S is the set of all variables!

What is the height of this lattice?

Recall Live Variable Analysis

Any solution $Live_{entry}$, $Live_{exit}$ to this system of equations overapproximates the true set of live variables.

$$x \text{ is live at } l \Rightarrow x \in LV_{exit}(l)$$

Our domain is a subset lattice where S is the set of all variables!

What is the height of this lattice?

The worklist algorithm terminates because this lattice has a finite height!

An Infinite Lattice with Finite Height

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

An Infinite Lattice with Finite Height

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

```
x := 2
y := 5
x := 1
z := 0
if (x <= 0) {
   z := x + 2
} else {
   z := y * y
}
x := z
```

An Infinite Lattice

The aim of the *Constant Propagation Analysis* is to determine

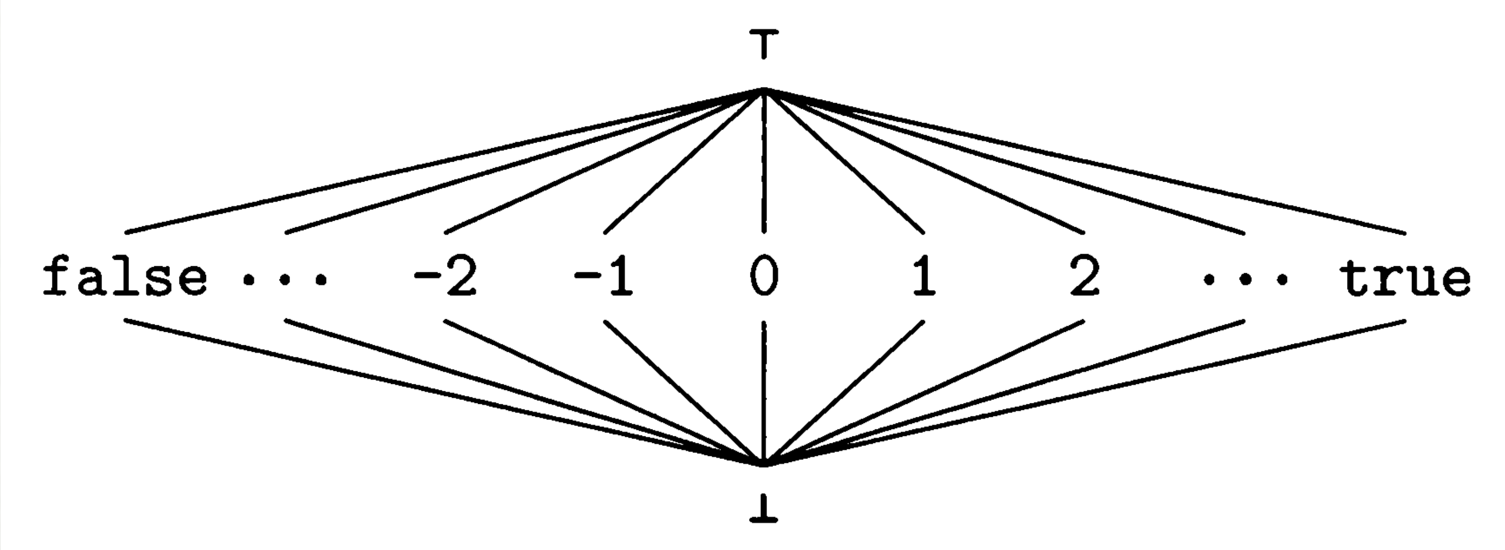
For each program point, whether or not a variable has a constant value whenever execution reaches that point.

An Infinite Lattice

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

Integer Constant Propagation Lattice

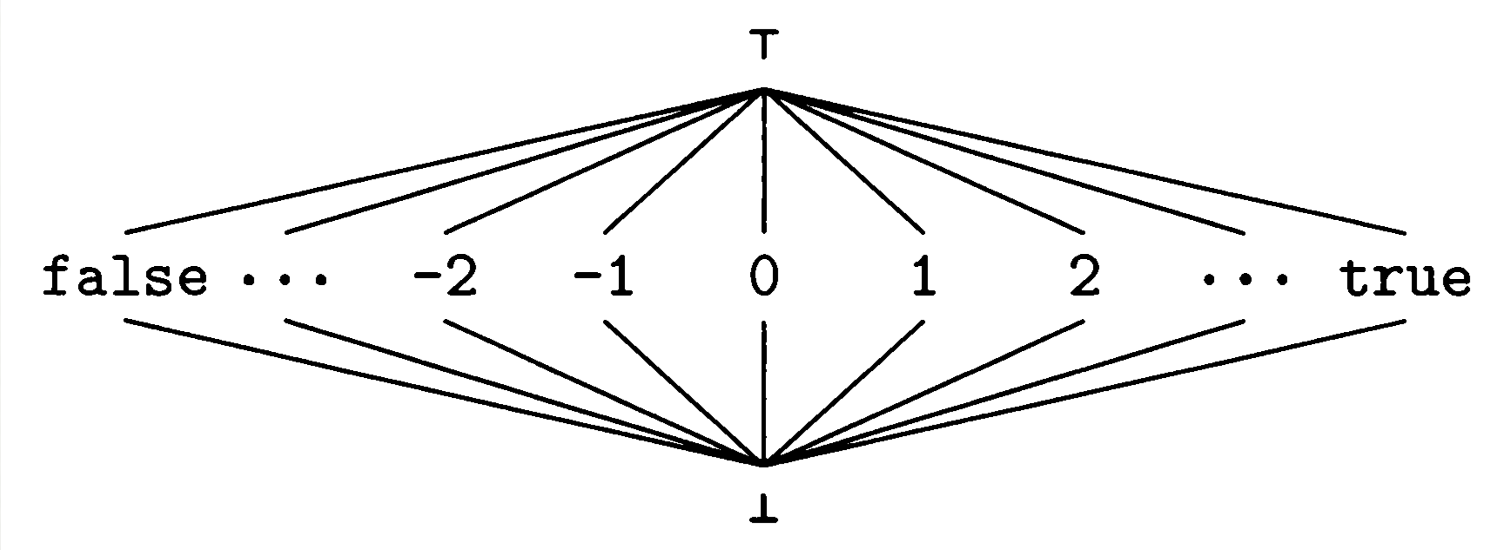


An Infinite Lattice

The aim of the *Constant Propagation Analysis* is to determine

For each program point, whether or not a variable has a constant value whenever execution reaches that point.

Integer Constant Propagation Lattice



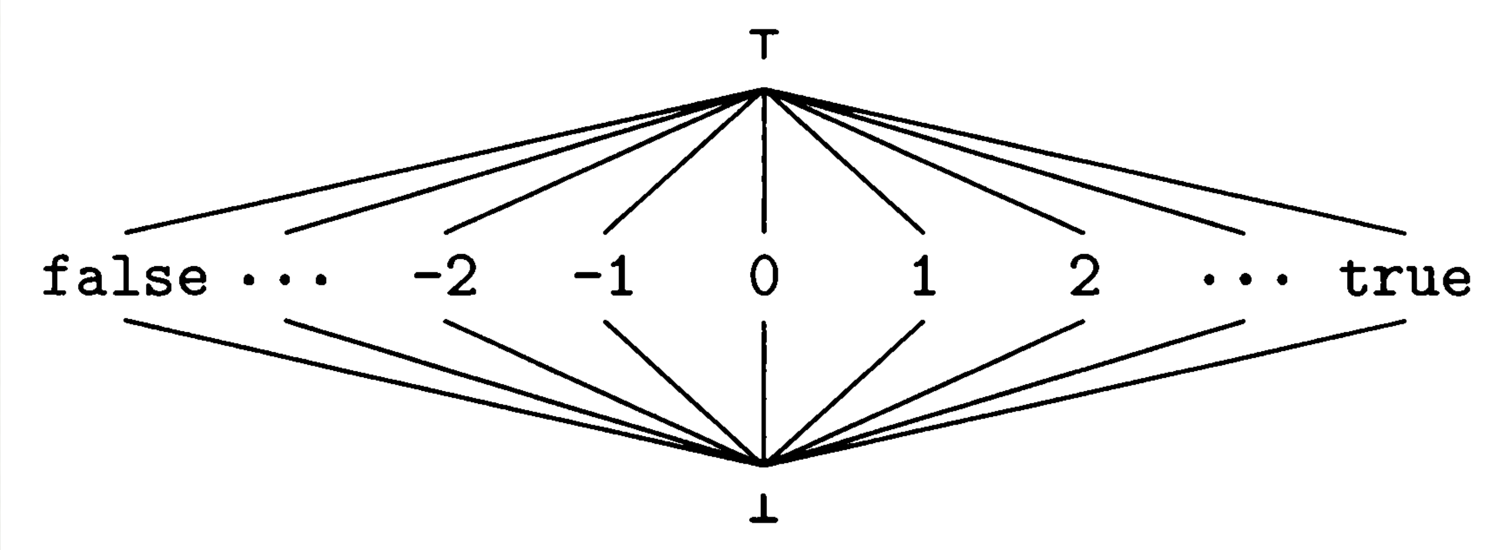
$\top$: non-constant

An Infinite Lattice

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

Integer Constant Propagation Lattice



$\top$: non-constant

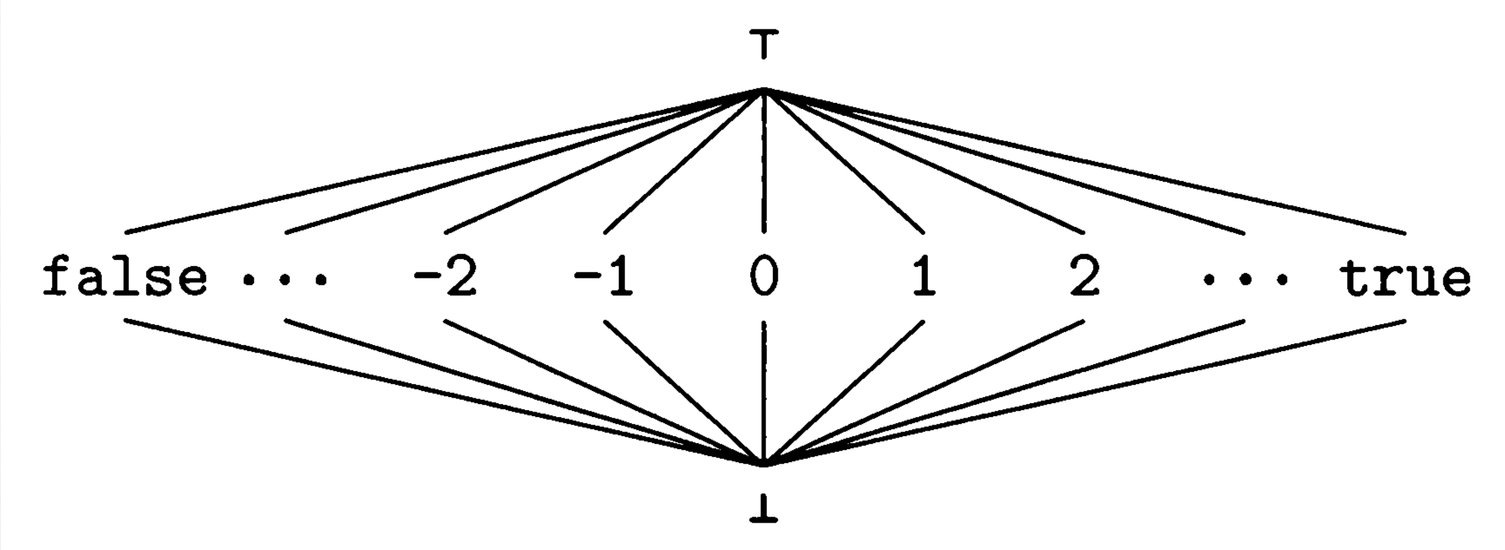
$\bot$: no-information

An Infinite Lattice

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

Integer Constant Propagation Lattice



$\top$: non-constant

$\bot$: no-information

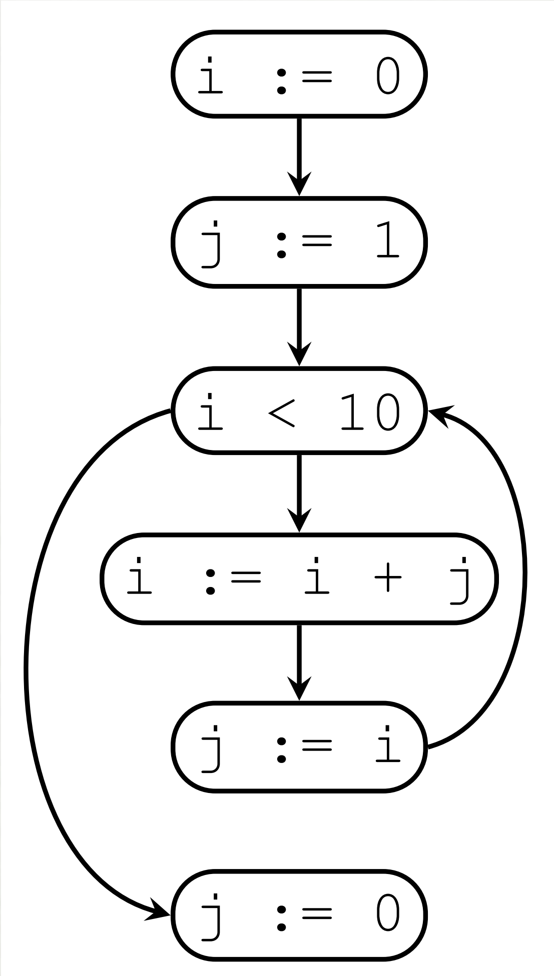
$$v_1 \sqcap v_2 = \begin{cases} v_1 & \text{if } v_1 = v_2 \\ \top & \text{if } v_1 \neq v_2 \end{cases}$$

A General Framework for Dataflow Analyses based on Basic Lattice Theory

Component 1:
Domains are (semi)-lattices of finite height!

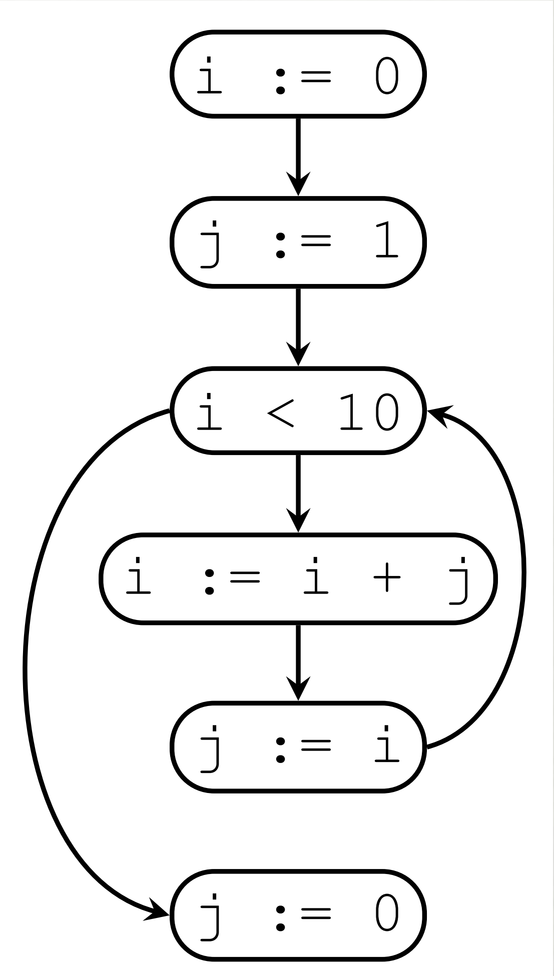
Example: Live Variable Analysis

The domain is a (complete) subset lattice $(\mathcal{P}(S), \cup, \emptyset, S)$ where S is the set of all program variables.



Example: Live Variable Analysis

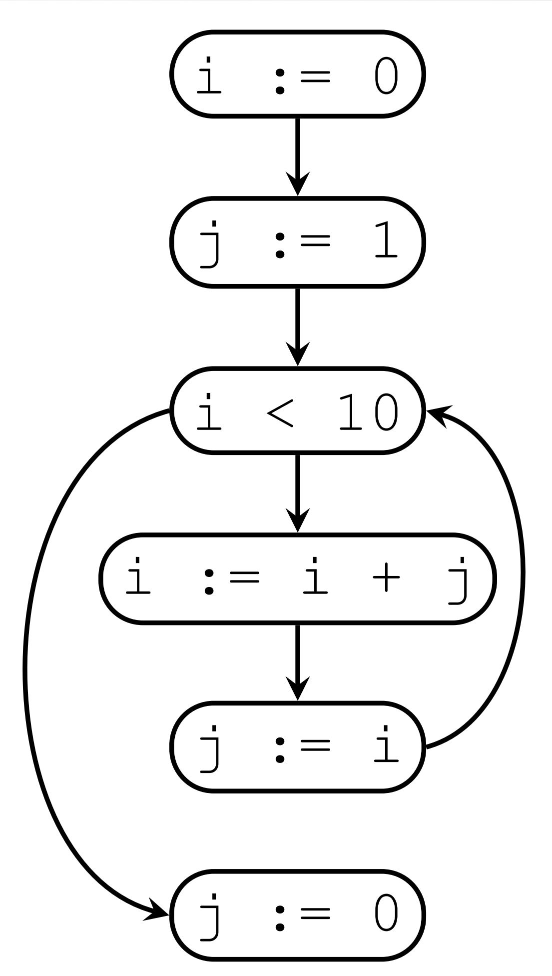
The domain is a (complete) subset lattice $(\mathcal{P}(S), \cup, \emptyset, S)$ where S is the set of all program variables.



Why $\cup$? Because a variable is live if it is live on some path from the node.

Example: Live Variable Analysis

The domain is a (complete) subset lattice $(\mathcal{P}(S), \cup, \emptyset, S)$ where S is the set of all program variables.



Why $\cup$? Because a variable is live if it is live on some path from the node.

By choosing to call variables live at exit, we also decide that this is a backward dataflow problem.

Example: Live Variable Analysis

To fully define the domain:

Example: Live Variable Analysis

To fully define the domain:

- Define the (semi) lattice: dataflow facts and how to combine them!

Example: Live Variable Analysis

To fully define the domain:

- Define the (semi) lattice: dataflow facts and how to combine them!

- Decide on the direction of the analysis: forward or backward!

Component 2:
Transfer Functions

Transfer Functions

A *transfer function* models, for a particular data flow analysis problem, the effect of the programming language constructs as a mapping from the lattice (used in the analysis) to itself).

$$\forall st \in \text{Statements},\ f_{st} : \mathbf{L} \mapsto \mathbf{L}$$

Transfer Functions

A *transfer function* models, for a particular data flow analysis problem, the effect of the programming language constructs as a mapping from the lattice (used in the analysis) to itself).

$$\forall st \in \text{Statements}, \; f_{st} : \mathbf{L} \mapsto \mathbf{L}$$

Example:

$$\begin{aligned} LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\ LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell') \end{aligned}$$

Transfer Functions

A *transfer function* models, for a particular data flow analysis problem, the effect of the programming language constructs as a mapping from the lattice (used in the analysis) to itself).

$$\forall st \in \text{Statements},\ f_{st} : \mathbf{L} \mapsto \mathbf{L}$$

Example:

backward: would reverse for forward!

$$\begin{aligned} LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\ LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell') \end{aligned}$$

Properties of Transfer Functions

Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Properties of Transfer Functions

Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Distributivity: $\forall x, y \in \mathbf{L} : f(x \sqcap y) = f(x) \sqcap f(y).$

Properties of Transfer Functions

Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Distributivity: $\forall x, y \in \mathbf{L} : f(x \sqcap y) = f(x) \sqcap f(y).$

Example:

$$\begin{aligned} LV_{entry}(\ell) &= LV_{exit}(\ell) \setminus write(\ell) \cup read(\ell) \\ LV_{exit}(\ell) &= \bigcup_{\ell \to \ell' \in E} LV_{entry}(\ell') \end{aligned}$$

Properties of Transfer Functions

Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Transfer functions in a dataflow analysis must be monotone!

Distributivity: $\forall x, y \in \mathbf{L} : f(x \sqcap y) = f(x) \sqcap f(y).$

Properties of Transfer Functions

Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Transfer functions in a dataflow analysis must be monotone!

Distributivity: $\forall x, y \in \mathbf{L} : f(x \sqcap y) = f(x) \sqcap f(y).$

But not necessarily distributive!

Properties of Transfer Functions

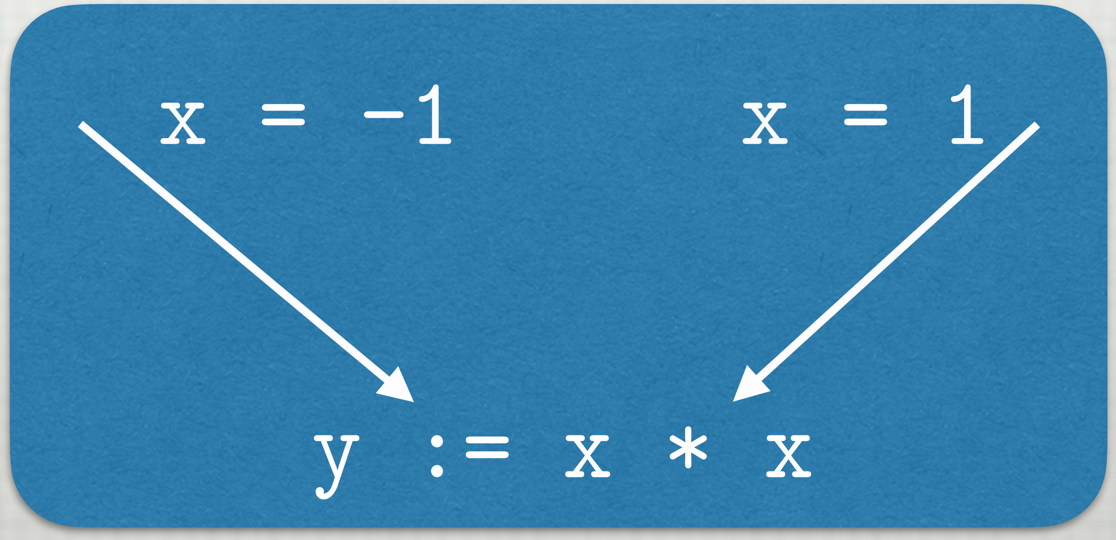
Monotonicity: $\forall x, y \in \mathbf{L} : x \sqsubseteq y \implies f(x) \sqsubseteq f(y).$

Transfer functions in a dataflow analysis must be monotone!

Distributivity: $\forall x, y \in \mathbf{L} : f(x \sqcap y) = f(x) \sqcap f(y).$

But not necessarily distributive!

Constant Propagation:



Component 3:
The Computation

The Goal

What is the *goal* of dataflow analyses?

The Goal

What is the *goal* of dataflow analyses?

meet-over-all-paths (MOP) solutions.

The Goal

What is the *goal* of dataflow analyses?

meet-over-all-paths (MOP) solutions.

- Start from the beginning (entry node, or exist note for backward flow problems) with some initial information.

The Goal

What is the *goal* of dataflow analyses?

meet-over-all-paths (MOP) solutions.

- Start from the beginning (entry node, or exist note for backward flow problems) with some initial information.

- Walk down a path and apply transfer functions along these paths to each node in the flow graph.

The Goal

What is the *goal* of dataflow analyses?

meet-over-all-paths (MOP) solutions.

- Start from the beginning (entry node, or exist note for backward flow problems) with some initial information.

- Walk down a path and apply transfer functions along these paths to each node in the flow graph.

- For each node, compute the *meet* of all paths to this point.

Formally

For a path $\pi = init \ldots l$

$$f_\pi = f_{init} \circ \cdots \circ f_l$$

$$MOP_\circ(l) = \textstyle\bigsqcap_{\pi \in Path(l)} f_\pi(\iota).$$

$$MOP_\bullet(l) = f_l(MOP_\circ(l)).$$

Formally

For a path $\pi = init \ldots l$

Transfer Function for location l

$$f_\pi = f_{init} \circ \cdots \circ f_l$$

Set of all paths to l

Initial information at "init"

$$MOP_\circ(l) = \bigsqcap_{\pi \in Path(l)} f_\pi(\iota).$$

$$MOP_\bullet(l) = f_l(MOP_\circ(l)).$$

Can this solution be computed effectively?

Bad News!

For an arbitrary data flow analysis problem where transfer functions are only monotone, one can show that there may be no algorithm to compute the MOP solution.

Bad News!

For an arbitrary data flow analysis problem where transfer functions are only monotone, one can show that there may be no algorithm to compute the MOP solution.

Lemma

The MOP solution for Constant Propagation is undecidable.

Proof: Let $u_1, \cdots, u_n$ and $v_1, \cdots, v_n$ be strings over the alphabet $\{1, \cdots, 9\}$; let $| u |$ denote the length of u; let $[\![u]\!]$ be the natural number denoted.

The Modified Post Correspondence Problem is to determine whether or not $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_n}$ for some sequence $i_1, \cdots, i_m$ with $i_1 = 1$.

x:=$[\![u_1]\!]$; y:=$[\![v_1]\!]$;
while [$\cdots$] do
  (if [$\cdots$] then x:=x * $10^{|u_1|}$ + $[\![u_1]\!]$; y:=y * $10^{|v_1|}$ + $[\![v_1]\!]$ else
   ⋮
   if [$\cdots$] then x:=x * $10^{|u_n|}$ + $[\![u_n]\!]$; y:=y * $10^{|v_n|}$ + $[\![v_n]\!]$ else skip)
[z:=abs((x-y)*(x-y))]$^\ell$

Then $MOP_\bullet(\ell)$ will map z to 1 if and only if the Modified Post Correspondence Problem has no solution. This is undecidable.

So, what do we do?

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Good News

Instead, compute the *maximal fixed point solution* (MFP).

in the meet lattice

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \bigsqcap_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \prod_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \prod_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

forward

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \bigsqcap_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

entry

forward

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

exit

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_{\circ}(l) = \begin{cases} \iota & l = init \\ \bigsqcap_{(l',l) \in flow} MFP_{\bullet}(l') & \text{otherwise} \end{cases}$$

exit

backward

$$MFP_{\bullet}(l) = f_l(MFP_{\circ}(l))$$

entry

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \bigsqcap_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

What is a solution to these set of constraints?

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \bigsqcap_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

What is a solution to these set of constraints?

A solution is a fixed point!

Good News

Instead, compute the *maximal fixed point solution* (MFP).

Consider the set of constraints below:

$$MFP_\circ(l) = \begin{cases} \iota & l = init \\ \prod_{(l',l)\in flow} MFP_\bullet(l') & \text{otherwise} \end{cases}$$

$$MFP_\bullet(l) = f_l(MFP_\circ(l))$$

What is a solution to these set of constraints?

A solution is a fixed point!

Is it unique?

Algebra brings it all together!

Fixed Point Solutions

Theorem: (Knaster-Tarski Fixpoint Theorem)

Fixed Point Solutions

Theorem: (Knaster-Tarski Fixpoint Theorem)

Let $\mathbf{L}$ be a complete lattice and $F : \mathbf{L} \to \mathbf{L}$ be a monotone function. Then, the set of fixpoints of F in $\mathbf{L}$ is also a complete lattice.

Fixed Point Solutions

Theorem: (Knaster-Tarski Fixpoint Theorem)

Let $\mathbf{L}$ be a complete lattice and $F : \mathbf{L} \to \mathbf{L}$ be a monotone function. Then, the set of fixpoints of F in $\mathbf{L}$ is also a complete lattice.

Corollary: We have a set of solutions (fixed points), with a guarantee for the existence of a maximal (also minimal) solution.

How do we compute it?

Fixed Point Solutions

Theorem: (Kleene Fixpoint Theorem)

Fixed Point Solutions

Theorem: (Kleene Fixpoint Theorem)

Let $\mathbf{L}$ be a complete lattice and $F : \mathbf{L} \to \mathbf{L}$ be a monotone function. The maximal fixpoint of $\mathbb{L}$ is the infimum of the descending chain $\top \sqsupseteq F(\top) \sqsupseteq F(F(\top)) \sqsupseteq \ldots$.

Fixed Point Solutions

Theorem: (Kleene Fixpoint Theorem)

Let $\mathbf{L}$ be a complete lattice and $F : \mathbf{L} \to \mathbf{L}$ be a monotone function. The maximal fixpoint of $\mathbb{L}$ is the infimum of the descending chain $\top \sqsupseteq F(\top) \sqsupseteq F(F(\top)) \sqsupseteq \ldots$.

We can start with the solution $\top$ and continually apply F to its own result until we eventually reach a fixed point which will be maximal.

Fixed Point Solutions

Theorem: (Kleene Fixpoint Theorem)

Let $\mathbf{L}$ be a complete lattice and $F : \mathbf{L} \to \mathbf{L}$ be a monotone function. The maximal fixpoint of $\mathbb{L}$ is the infimum of the descending chain $\top \sqsupseteq F(\top) \sqsupseteq F(F(\top)) \sqsupseteq \ldots$.

We can start with the solution $\top$ and continually apply F to its own result until we eventually reach a fixed point which will be maximal.

DFA Algorithm

$\forall k \in N \,.\, \text{IN}_k = \text{OUT}_k = \top$
repeat
  foreach $k \in N$ **do** {
    $\text{IN}_k = \sqcap\{\text{OUT}_p \mid p \in pred(k)\}$
    $\text{OUT}_k = F_k(\text{IN}_k)$
  }
while solution changes

The Coincidence

If transfer functions are monotone:

$$MOP_\circ(l) \sqsupseteq MFP_\circ(l) \qquad MOP_\bullet(l) \sqsupseteq MFP_\bullet(l)$$

Less Precise!

The Coincidence

If transfer functions are monotone:

$$MOP_\circ(l) \sqsupseteq MFP_\circ(l) \qquad MOP_\bullet(l) \sqsupseteq MFP_\bullet(l)$$

The Coincidence

If transfer functions are monotone:

$$MOP_\circ(l) \sqsupseteq MFP_\circ(l) \qquad MOP_\bullet(l) \sqsupseteq MFP_\bullet(l)$$

The fixpoint solution over-approximates the result!

The Coincidence

If transfer functions are monotone:

$$MOP_\circ(l) \sqsupseteq MFP_\circ(l) \qquad MOP_\bullet(l) \sqsupseteq MFP_\bullet(l)$$

The fixpoint solution over-approximates the result!

If transfer functions are distributive:

$$MOP_\circ(l) = MFP_\circ(l) \qquad MOP_\bullet(l) = MFP_\bullet(l)$$

The Coincidence

If transfer functions are monotone:

$$MOP_{\circ}(l) \sqsupseteq MFP_{\circ}(l) \qquad MOP_{\bullet}(l) \sqsupseteq MFP_{\bullet}(l)$$

Less Precise!

The fixpoint solution over-approximates the result!

If transfer functions are distributive:

$$MOP_{\circ}(l) = MFP_{\circ}(l) \qquad MOP_{\bullet}(l) = MFP_{\bullet}(l)$$

The two solutions coincide!

Let’s make another instance of our framework!

Very Busy Expressions

```
if [a>b]^1 then ([x:=b-a]^2; [y:=a-b]^3) else ([y:=b-a]^4; [x:=a-b]^5)
```

Very Busy Expressions

if [a>b]^1 then ([x:=b-a]^2; [y:=a-b]^3) else ([y:=b-a]^4; [x:=a-b]^5)

[t1:=b-a]^A; [t2:=b-a]^B;
if [a>b]^1 then ([x:=t1]^2; [y:=t2]^3) else ([y:=t1]^4; [x:=t2]^5)

Very Busy Expressions

if [a>b]^1 then ([x:=b-a]^2; [y:=a-b]^3) else ([y:=b-a]^4; [x:=a-b]^5)

[t1:=b-a]^A; [t2:=b-a]^B;
if [a>b]^1 then ([x:=t1]^2; [y:=t2]^3) else ([y:=t1]^4; [x:=t2]^5)

An expression is very busy at the exit from a label if, no matter what path is taken from the label, the expression is always used before any of the variables occurring in it are redefined.

Check List

Check List

- Define the semi-lattice: dataflow facts and how to combine them!

Check List

- Define the semi-lattice: dataflow facts and how to combine them!
    - Decide on the direction of the analysis: forward vs backward.

Check List

- Define the semi-lattice: dataflow facts and how to combine them!
  - Decide on the direction of the analysis: forward vs backward.
  - Sanity check: the corresponding order should make sense!

Check List

- Define the semi-lattice: dataflow facts and how to combine them!
  - Decide on the direction of the analysis: forward vs backward.
  - Sanity check: the corresponding order should make sense!
  - Decide on the initial values.

Check List

- Define the semi-lattice: dataflow facts and how to combine them!
    - Decide on the direction of the analysis: forward vs backward.
    - Sanity check: the corresponding order should make sense!
    - Decide on the initial values.
- Design the transfer functions:

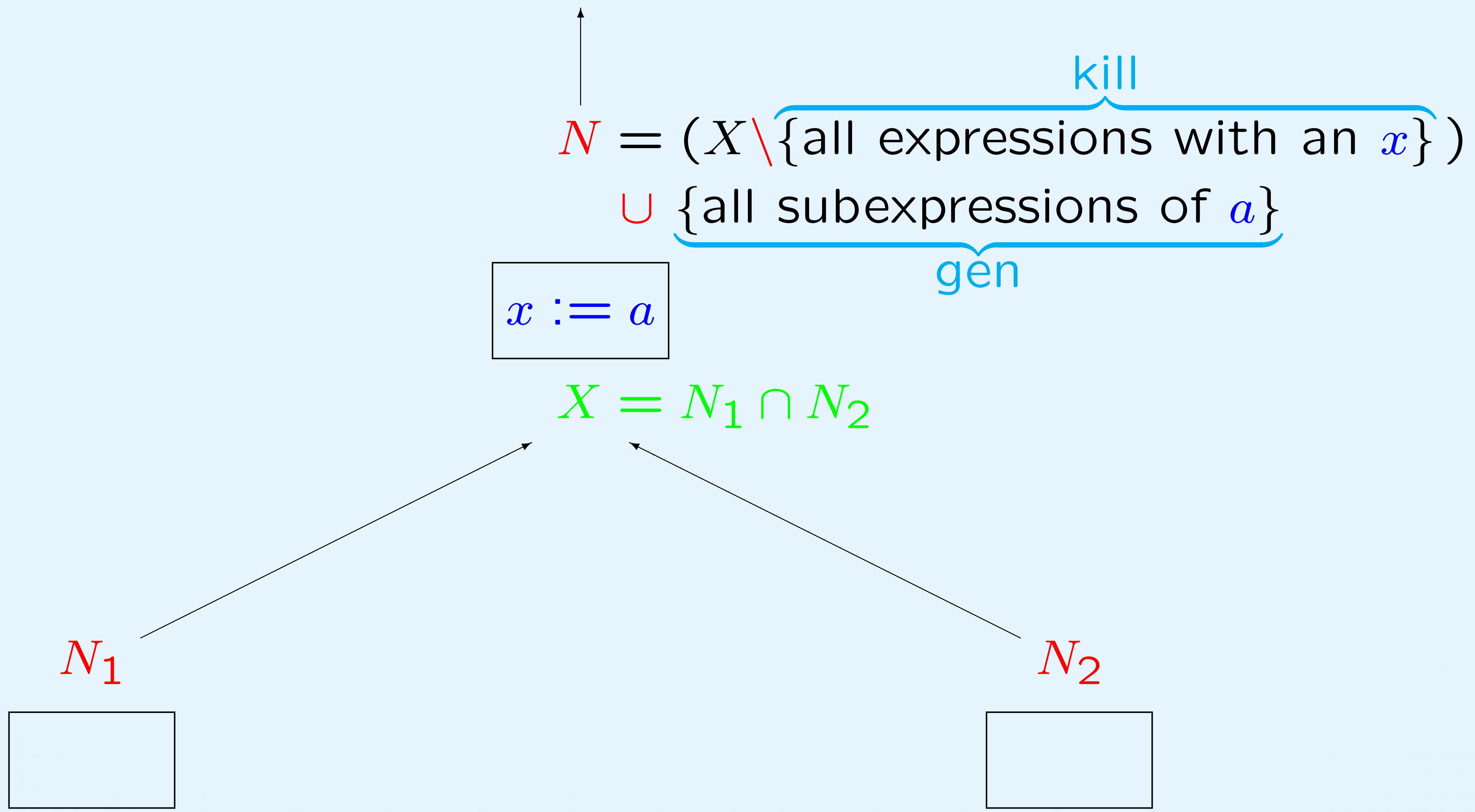
Check List

- Define the semi-lattice: dataflow facts and how to combine them!
    - Decide on the direction of the analysis: forward vs backward.
    - Sanity check: the corresponding order should make sense!
    - Decide on the initial values.
- Design the transfer functions:
    - How does each statement affect the dataflow facts?

Check List

- Define the semi-lattice: dataflow facts and how to combine them!
    - Decide on the direction of the analysis: forward vs backward.
    - Sanity check: the corresponding order should make sense!
    - Decide on the initial values.
- Design the transfer functions:
    - How does each statement affect the dataflow facts?
    - Sanity check: Monotonicity!

The Design Process



Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

Direction: Backward

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

Direction: Backward

Transfer Functions:

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

Direction: Backward

Transfer Functions:

$$VB_{\bullet}(l) = \begin{cases} \emptyset & l = exit \\ \bigcap_{(l,l') \in flow} VB_{\bullet}(l') & \text{otherwise} \end{cases}$$

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

Direction: Backward

Transfer Functions:

$$VB_\bullet(l) = \begin{cases} \emptyset & l = exit \\ \bigcap_{(l,l') \in flow} VB_\bullet(l') & \text{otherwise} \end{cases}$$

$$VB_\circ(l) = RD_\bullet(l) \setminus \{exp | var(exp) \cap write(l) \neq \emptyset\} \\ \cup \; computed(l)$$

Very Busy Expressions: Formal Setup

Dataflow Facts: $D = \mathcal{P}(Exp)$

Domain: complete meet semi-lattice $(D, \cap, D)$

unlike live variables: here we want the greatest fixed point!

Direction: Backward

Transfer Functions:

$$VB_{\bullet}(l) = \begin{cases} \emptyset & l = exit \\ \bigcap_{(l,l') \in flow} VB_{\bullet}(l') & \text{otherwise} \end{cases}$$

$$VB_{\circ}(l) = RD_{\bullet}(l) \setminus \{exp | var(exp) \cap write(l) \neq \emptyset\}$$
$$\cup \; computed(l)$$