Program Analysis and Verification

0368-4479

http://www.cs.tau.ac.il/~maon/teaching/2013-2014/paav/paav1314b.html

Noam Rinetzky

Lecture 9: Abstract Interpretation II

Slides credit: Roman Manevich, Mooly Sagiv, Eran Yahav

From verification to analysis

- Manual program verification
  - Verifier provides assertions
    - Loop invariants

- Program analysis
  - Automatic program verification
  - Tool automatically synthesize assertions
    - Finds loop invariants

Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis

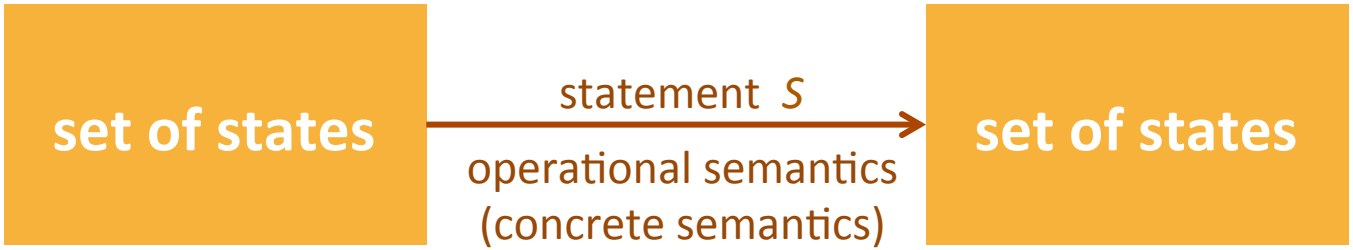
Abstract Interpretation [Cousot'77]

- Mathematical framework for approximating semantics (aka abstraction)
  - Allows designing sound static analysis algorithms
    - Usually compute by iterating to a fixed-point
  - Computes (loop) invariants
    - Can be interpreted as axiomatic verification assertions
    - Generalizes Hoare Logic & WP / SP calculus

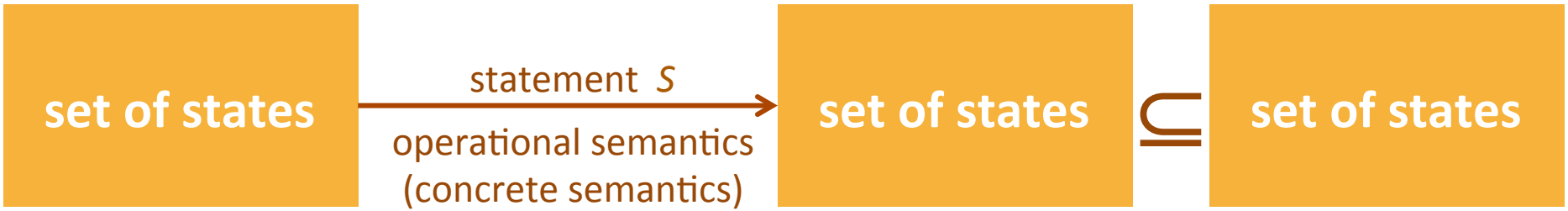
Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis
  - Abstract domains
    - Abstract states ~ Assertions
    - Join (⊔) ~ Weakening
  - Transformer functions
    - Abstract steps ~ Axioms
  - Chaotic iteration
    - Structured Programs ~ Control-flow graphs
    - Abstract computation ~ Loop invariants

Concrete Semantics

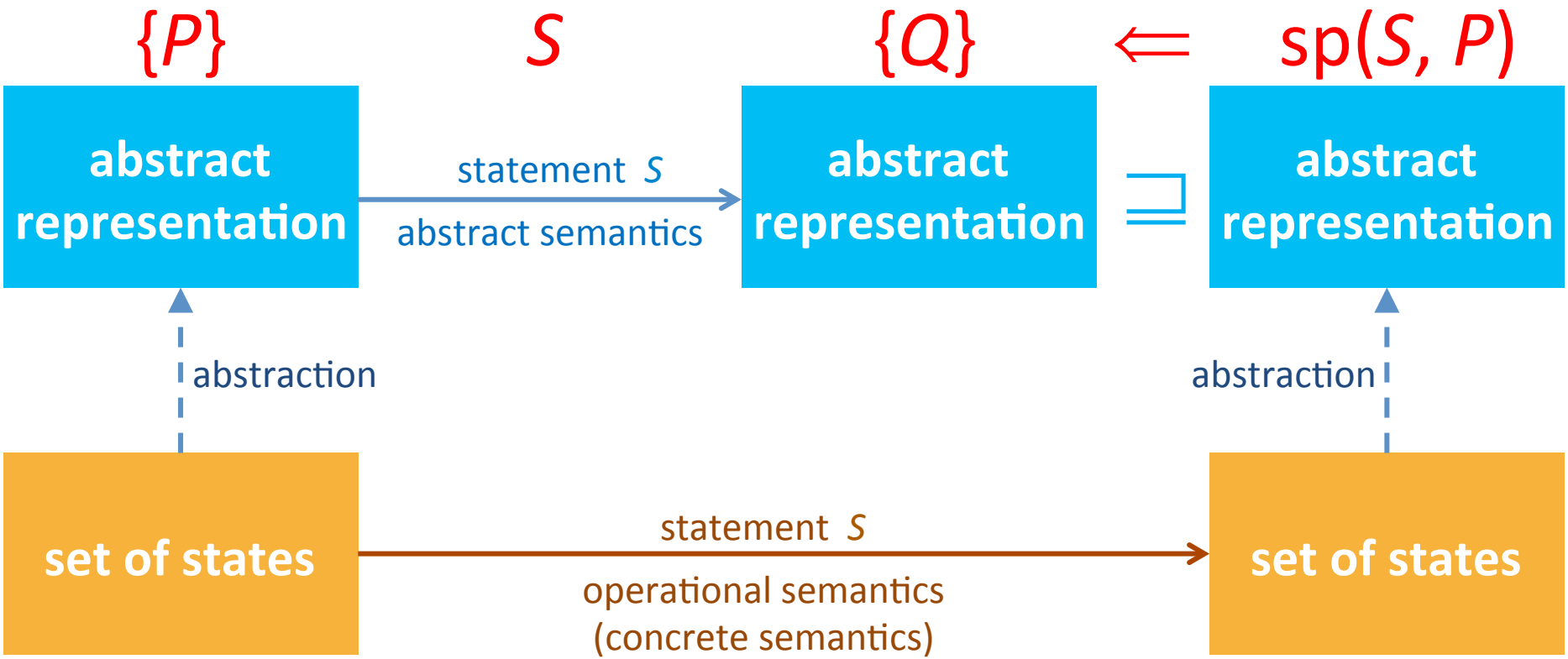


Conservative Semantics

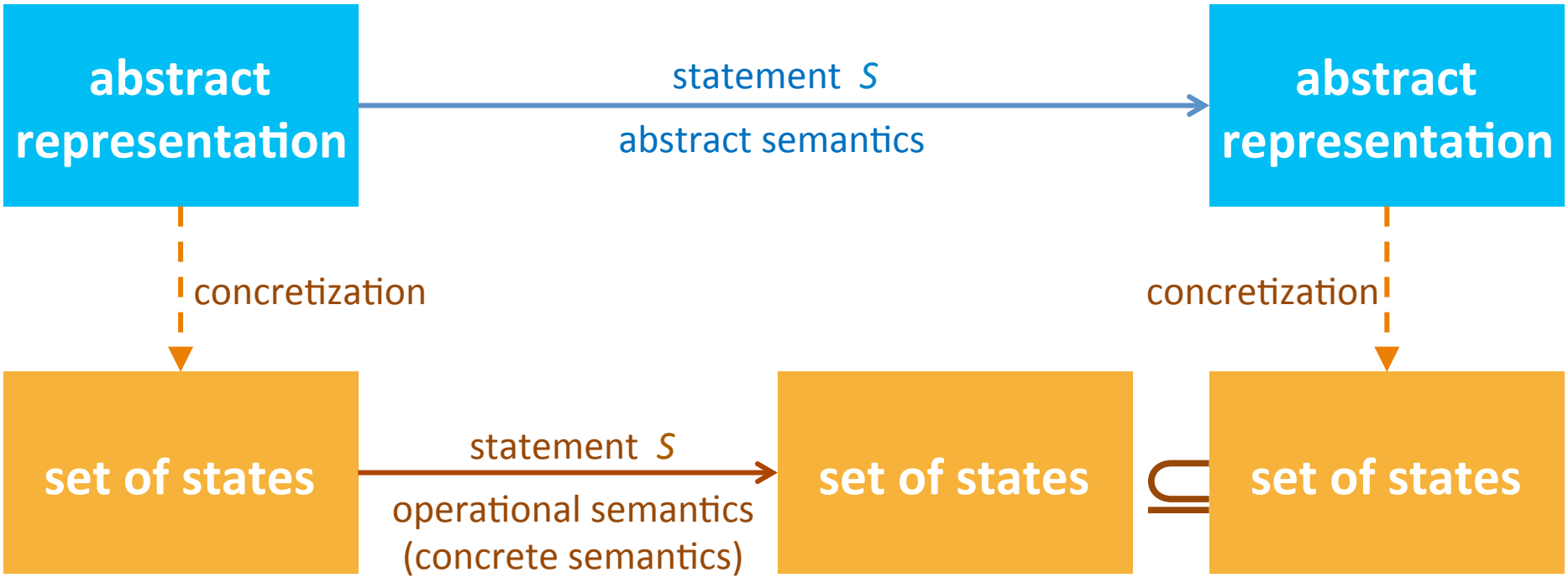


Abstract (conservative) interpretation

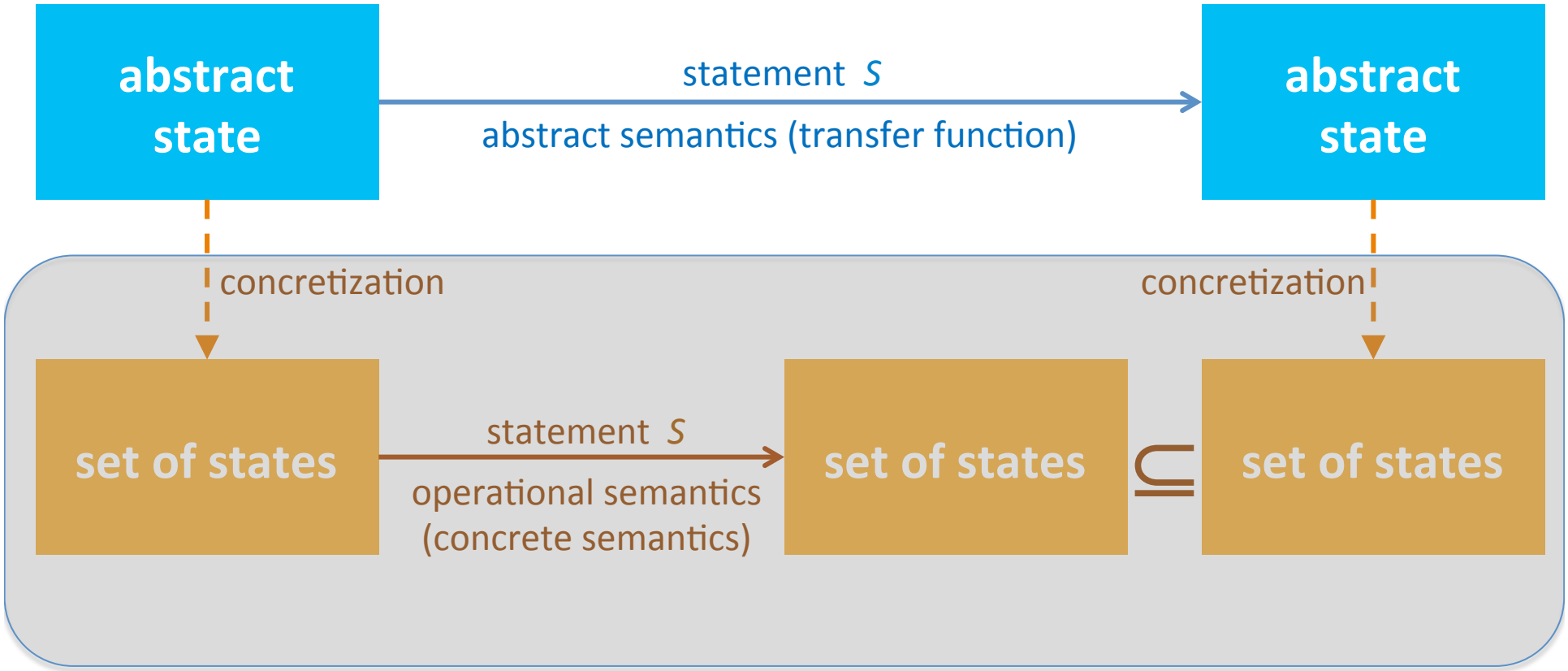
generalizes axiomatic verification



Abstract (conservative) interpretation



Abstract (conservative) interpretation



Abstract Interpretation [Cousot'77]

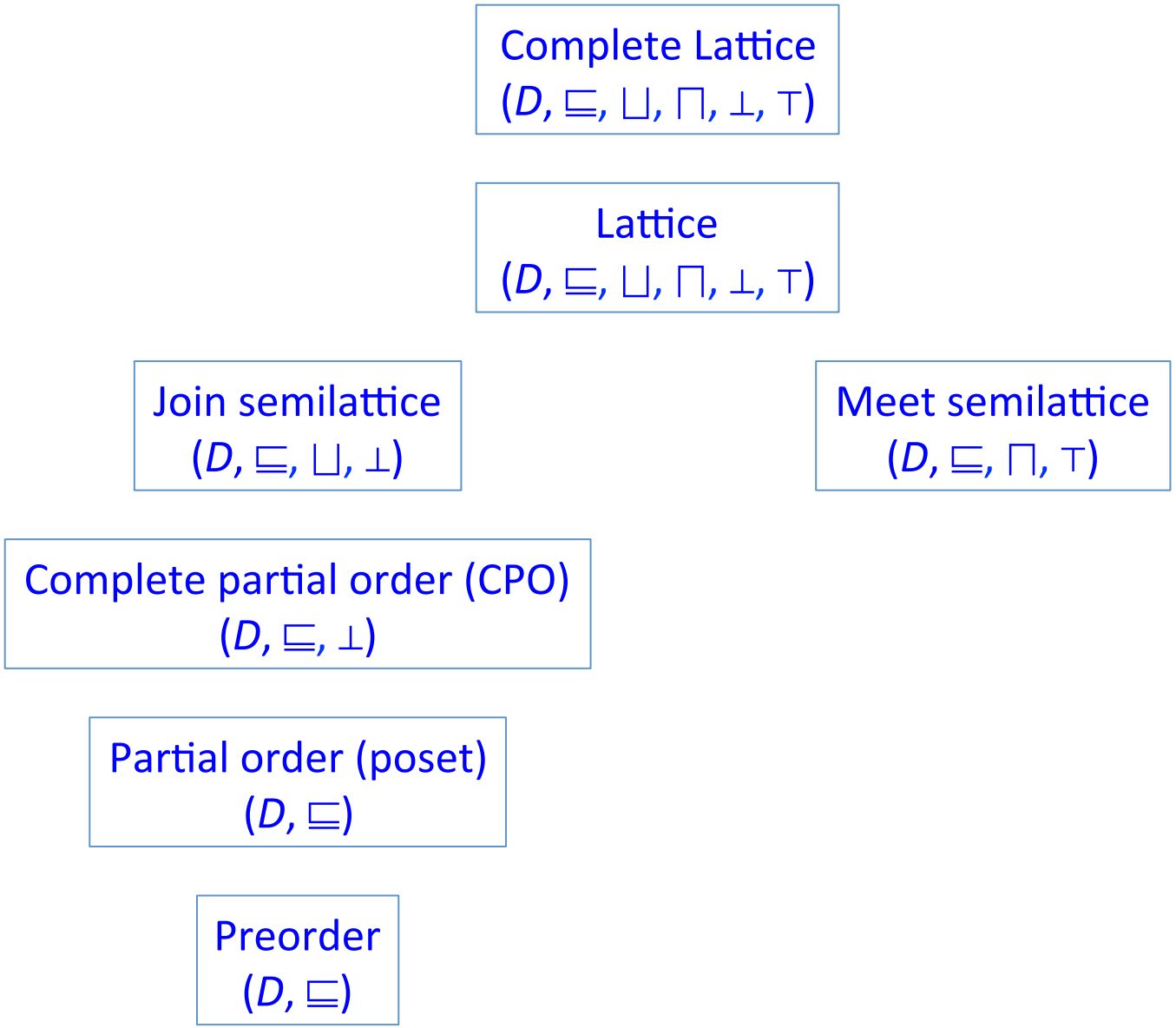
- Mathematical foundation of static analysis
  - Abstract domains
    - Abstract states ~ Assertions
    - Join (⊔) ~ Weakening
  - Transformer functions
    - Abstract steps ~ Axioms
  - Chaotic iteration
    - Abstract computation ~ Loop invariants
    - Structured Programs ~ Control-flow graphs

Lattices
$(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$

Monotonic functions

Fixpoints

A taxonomy of semantic domain types



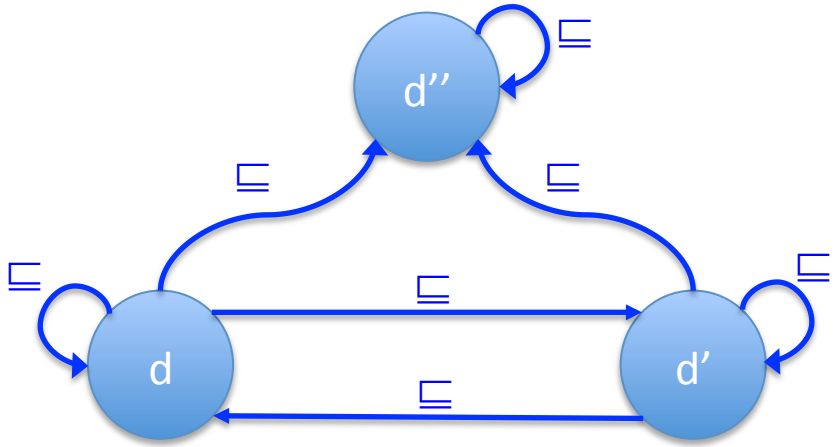
Preorder

- We say that a binary order relation $\sqsubseteq$ over a set D is a preorder if the following conditions hold for every d, d', d'' $\in$ D
  - Reflexive: $d \sqsubseteq d$
  - Transitive: $d \sqsubseteq d'$ and $d' \sqsubseteq d''$ implies $d \sqsubseteq d''$

Preorder

- We say that a binary order relation $\sqsubseteq$ over a set D is a preorder if the following conditions hold for every d, d', d'' $\in$ D
  - Reflexive: d $\sqsubseteq$ d
  - Transitive: d $\sqsubseteq$ d' and d' $\sqsubseteq$ d'' implies d $\sqsubseteq$ d''

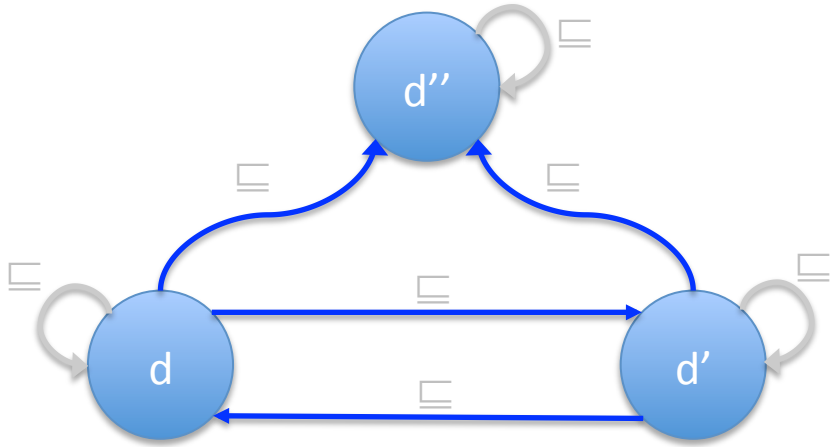
Hasse Diagram



Preorder

- We say that a binary order relation $\sqsubseteq$ over a set D is a preorder if the following conditions hold for every d, d', d'' $\in$ D
  - Reflexive: d $\sqsubseteq$ d
  - Transitive: d $\sqsubseteq$ d' and d' $\sqsubseteq$ d'' implies d $\sqsubseteq$ d''

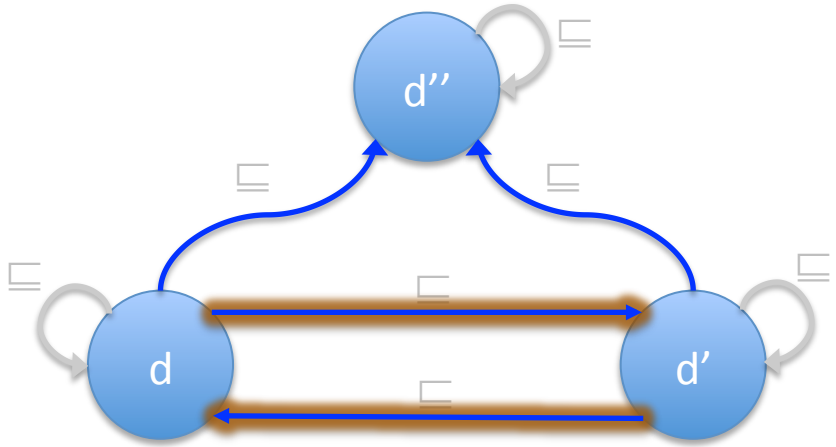
Hasse Diagram



Partial order

- We say that a binary order relation $\sqsubseteq$ over a set D is a preorder if the following conditions hold for every d, d', d'' $\in$ D
  - Reflexive: d $\sqsubseteq$ d
  - Transitive: d $\sqsubseteq$ d' and d' $\sqsubseteq$ d'' implies d $\sqsubseteq$ d''
  - Anti-symmetric: d $\sqsubseteq$ d' and d' $\sqsubseteq$ d implies d = d'

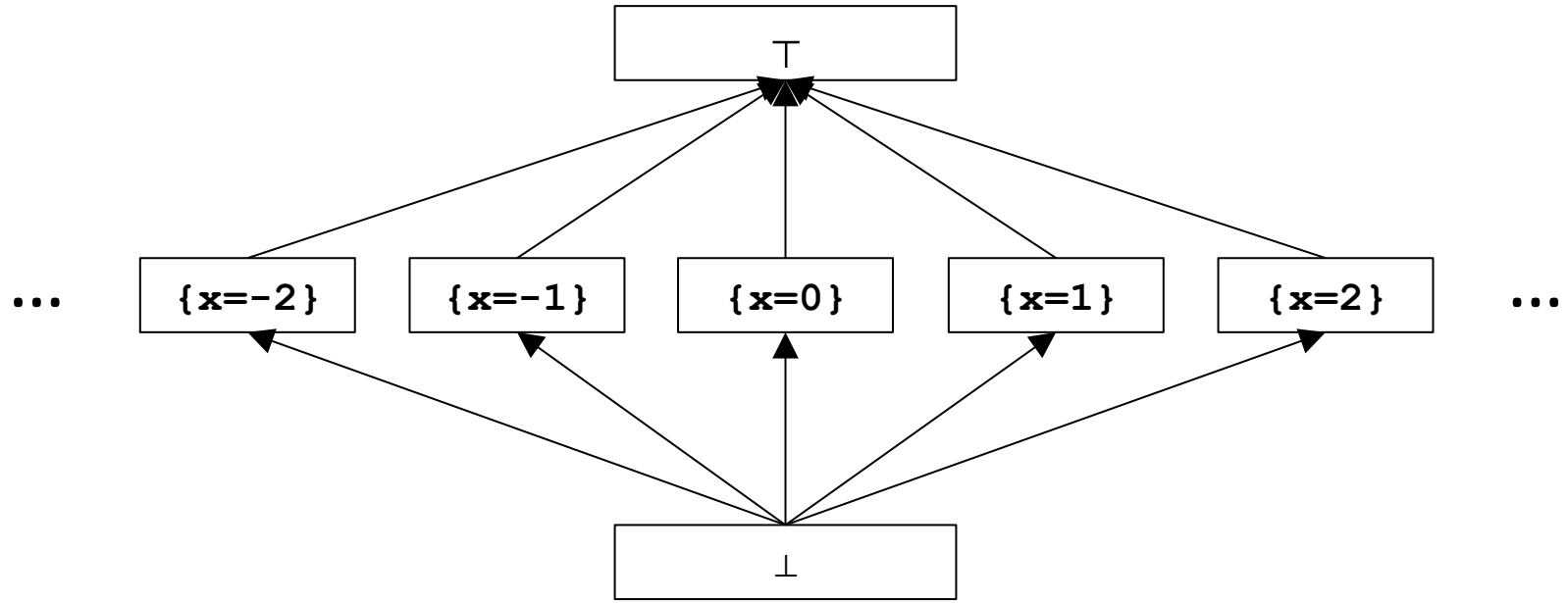
Hasse Diagram



Chains

- $d \sqsubset d'$ means $d \sqsubseteq d'$ and $d \neq d'$
- An ascending chain is a sequence
  $x_1 \sqsubset x_2 \sqsubset \ldots \sqsubset x_k \ldots$
- A descending chain is a sequence
  $x_1 \sqsupset x_2 \sqsupset \ldots \sqsupset x_k \ldots$
- The height of a poset $(D, \sqsubseteq)$ is the length of the maximal ascending chain in D

poset Hasse diagram (for CP)



Some posets-related terminology

- If $x \sqsubseteq y$ (alt $y \sqsupseteq x$) we can say
  - x is *lower* than y
  - x is *more precise* than y
  - x is *more concrete* than y
  - x *under-approximates* y

  - y is *greater* than x
  - y is *less precise* than x
  - y is *more abstract* than x
  - y *over-approximates* x

Least upper bound (LUB)

- $(D, \sqsubseteq)$ is a poset

- $b \in D$ is an upper bound of $A \subseteq D$ if $\forall a \in A: a \sqsubseteq b$

- $b \in D$ is the least upper bound of $A \subseteq D$ if
  - b is an upper bound of A
  - If b' is an upper bound of A then $b \sqsubseteq b'$
  - Join: $\bigsqcup X$ = LUB of X
    - $x \sqcup y = \bigsqcup\{x, y\}$

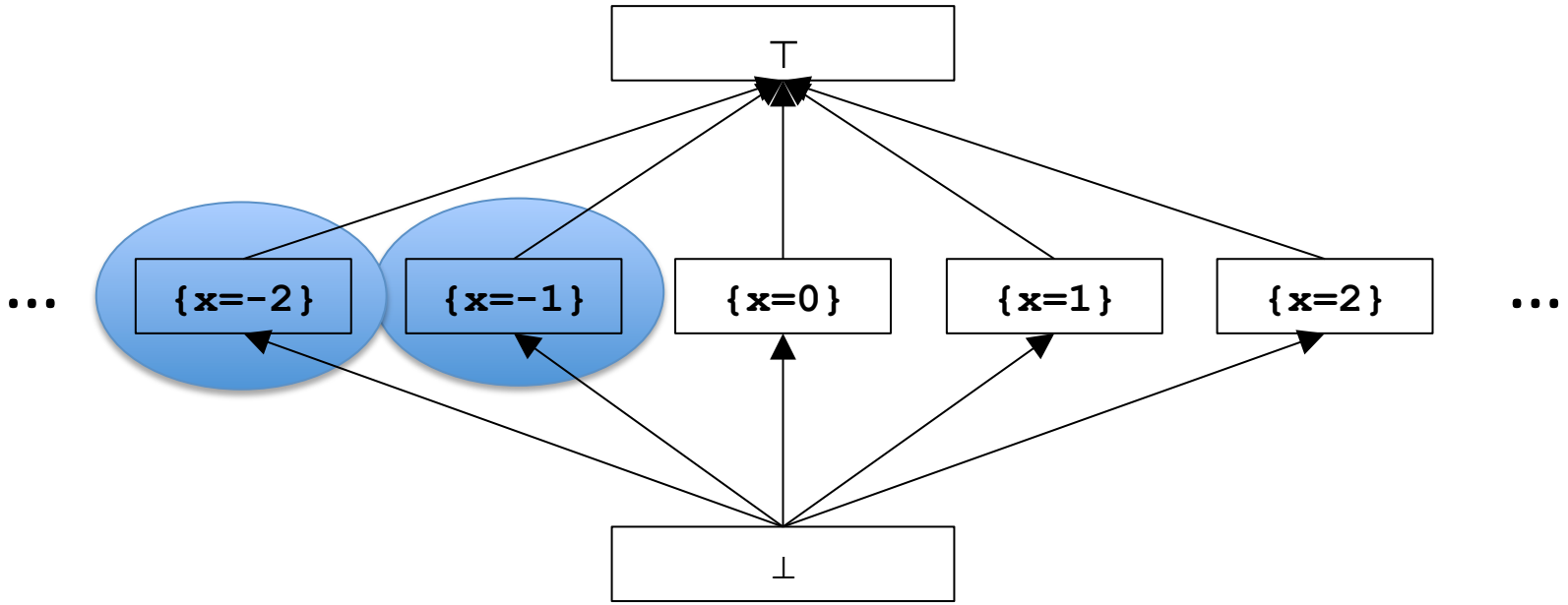
Join operator

- Properties of a join operator
  - Commutative: $x \sqcup y = y \sqcup x$
  - Associative: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
  - Idempotent: $x \sqcup x = x$

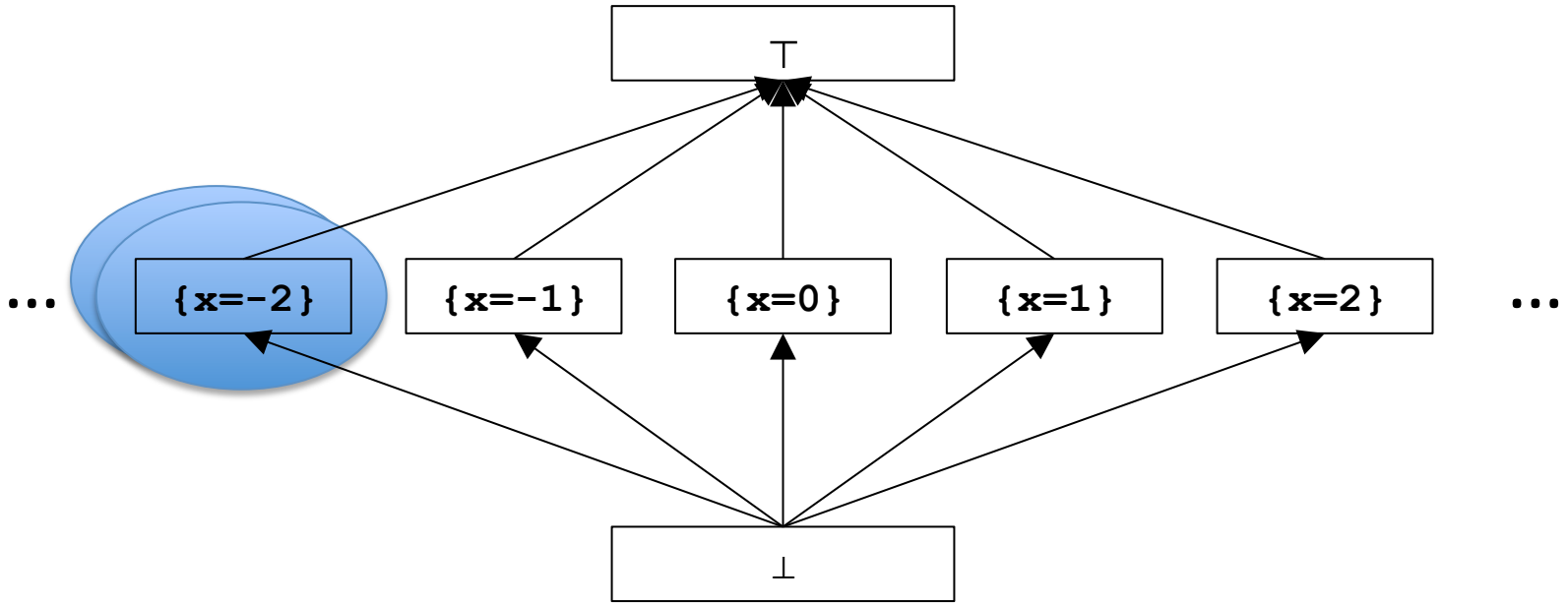
- A kind of abstract union (disjunction) operator

- Top element of $(D, \sqsubseteq)$ is $\top = \bigsqcup D$

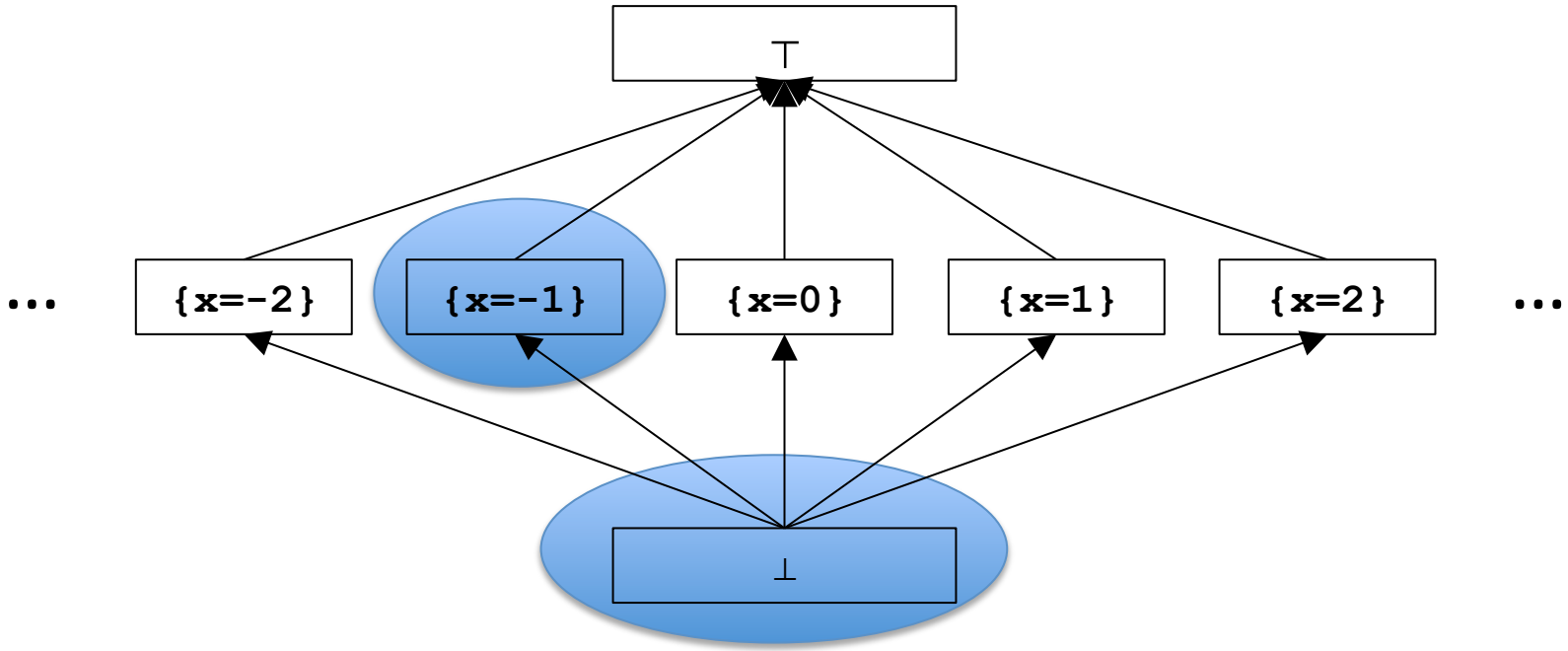
Join Example



Join Example



Join Example



Greatest lower bound (GLB)

- $(D, \sqsubseteq)$ is a poset

- $b \in D$ is an lower bound of $A \subseteq D$ if $\forall a \in A: b \sqsubseteq a$

- $b \in D$ is the greatest lower bound of $A \subseteq D$ if
  - b is an lower bound of A
  - If b' is an lower bound of A then $b' \sqsubseteq b$
  - Meet: $\sqcap X$ = GLB of X
    - $x \sqcap y = \sqcap \{x, y\}$

Meet operator

- Properties of a meet operator
  - Commutative: $x \sqcap y = y \sqcap x$
  - Associative: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  - Idempotent: $x \sqcap x = x$

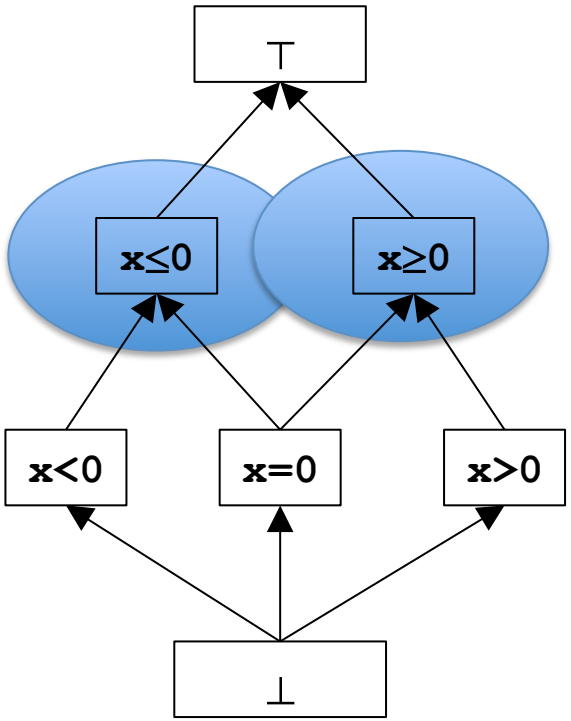
- A kind of abstract intersection (conjunction) operator

- Bottom element of $(D, \sqsubseteq)$ is $\bot = \sqcap D$

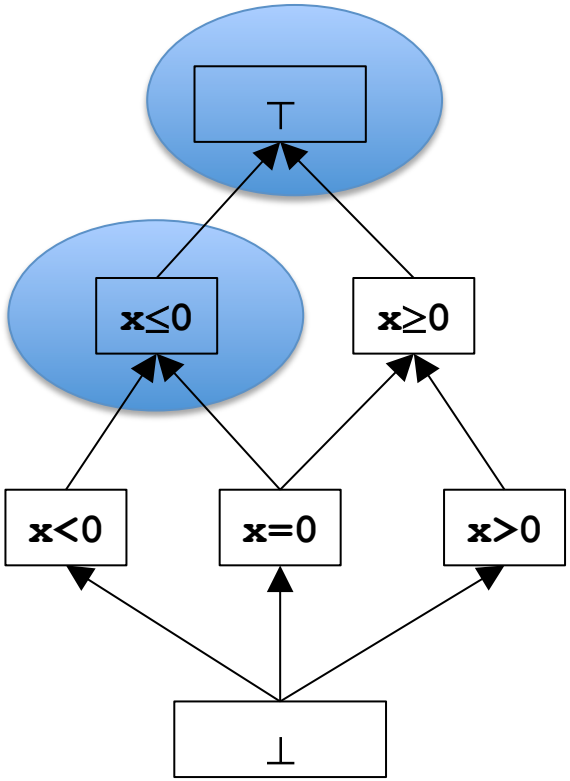
Complete partial order (CPO)

- A poset (D , $\sqsubseteq$) is a complete partial if every ascending chain $x_1 \sqsubset x_2 \sqsubset \ldots \sqsubset x_k \ldots$ has a LUB

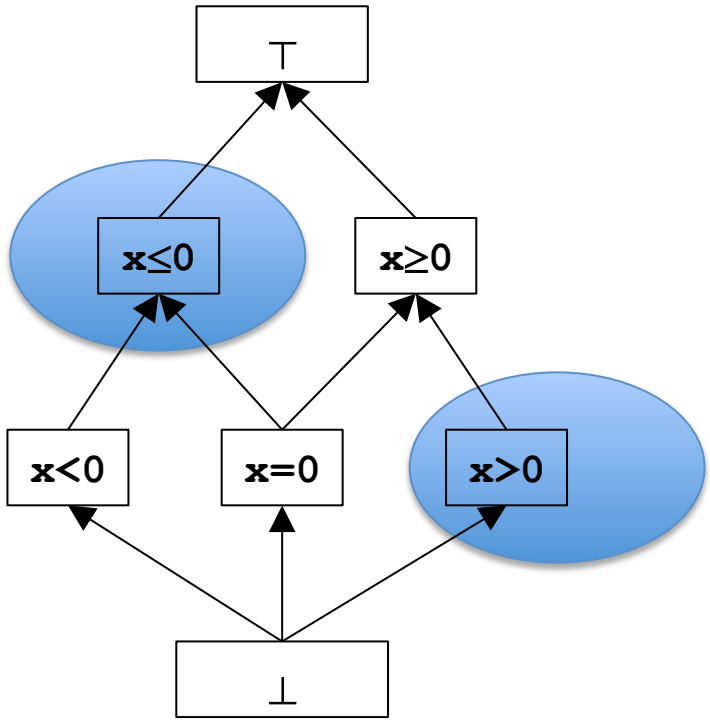
Meet Example



Meet Example



Meet Example



Complete partial order (CPO)

- A poset (D , $\sqsubseteq$) is a complete partial if every ascending chain $x_1 \sqsubset x_2 \sqsubset \ldots \sqsubset x_k \ldots$ has a LUB

Join semilattices

- $(D, \sqsubseteq, \sqcup, \top)$ is a join semilattice
  - $(D, \sqsubseteq)$ is a partial order
  - $\forall X \subseteq_{FIN} D \,.\, \sqcup X$ is defined
  - A top element $\top$

Meet semilattices

- $(D, \sqsubseteq, \sqcap, \bot)$ is a meet semilattice
  - $(D, \sqsubseteq)$ is a partial order
  - $\forall X \subseteq_{FIN} D \,.\, \sqcap X$ is defined
  - A bottom element $\bot$

Lattices

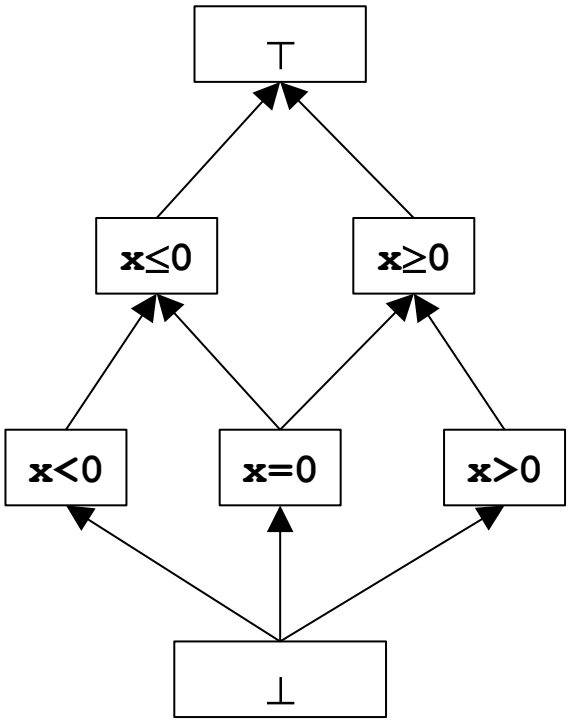
- $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is a lattice if
  - $(D, \sqsubseteq, \sqcup, \top)$ is a join semilattice
  - $(D, \sqsubseteq, \sqcap, \bot)$ is a meet semilattice

- A lattice $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is a complete lattice if
  - $\bigsqcup X$ and $\bigsqcap Y$ are defined for arbitrary sets

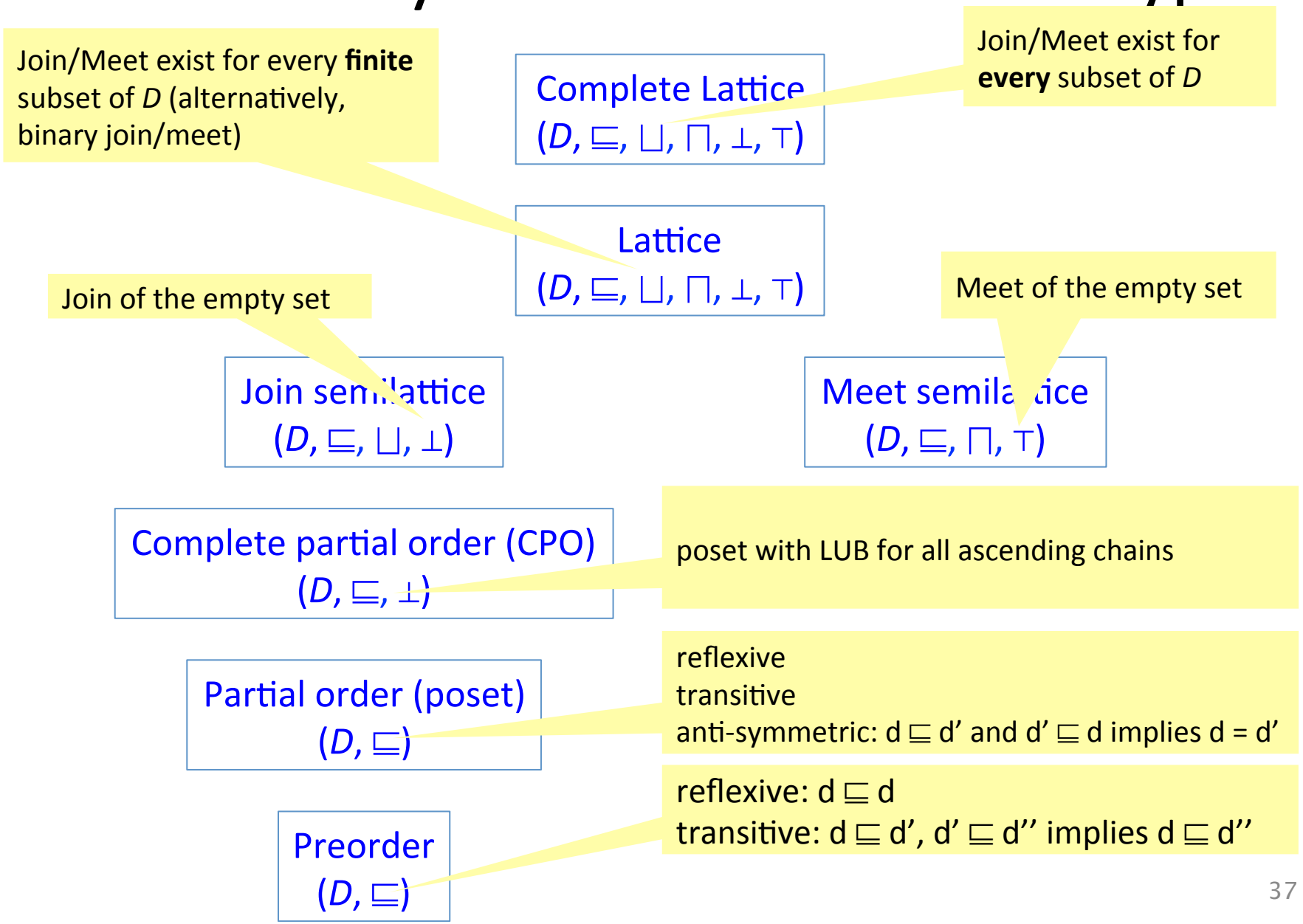
Example: Powerset lattices

- $(2^X, \subseteq, \cup, \cap, \varnothing, X)$ is the powerset lattice of X
  - A complete lattice

Example: Sign lattice



A taxonomy of semantic domain types



Towards a recipe for static analysis

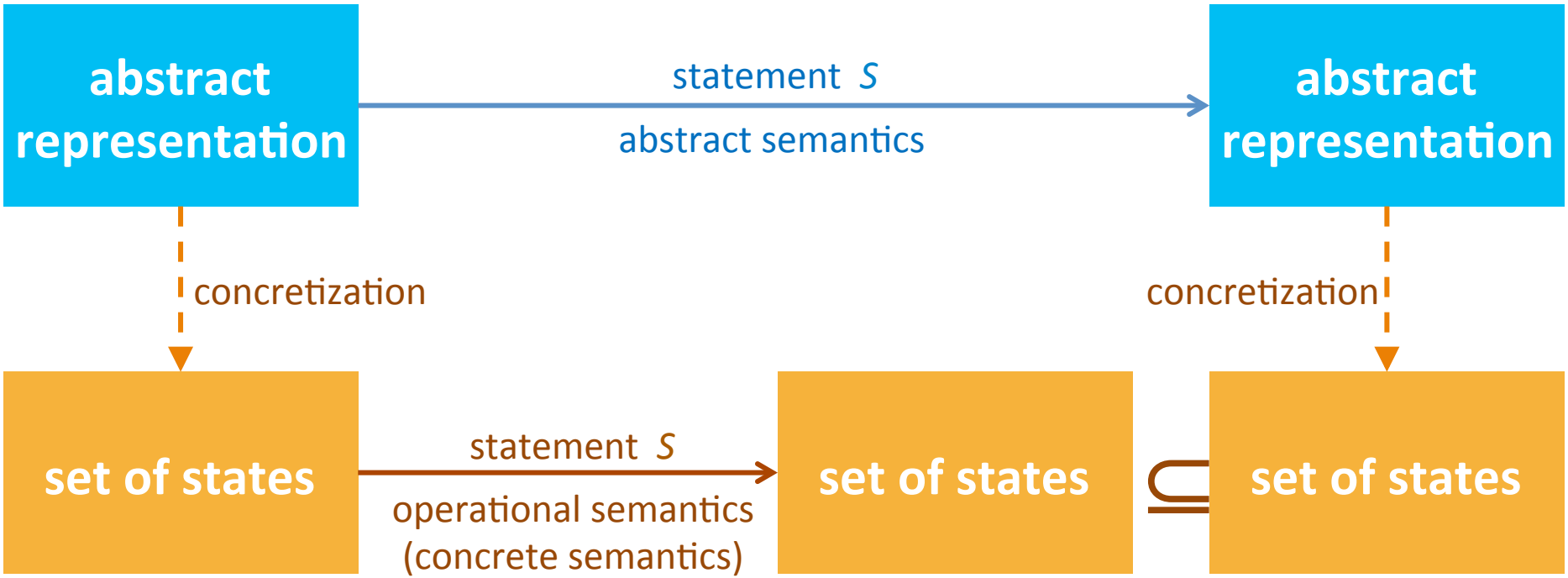
Collecting semantics

- For a set of program states **State**, we define the collecting lattice

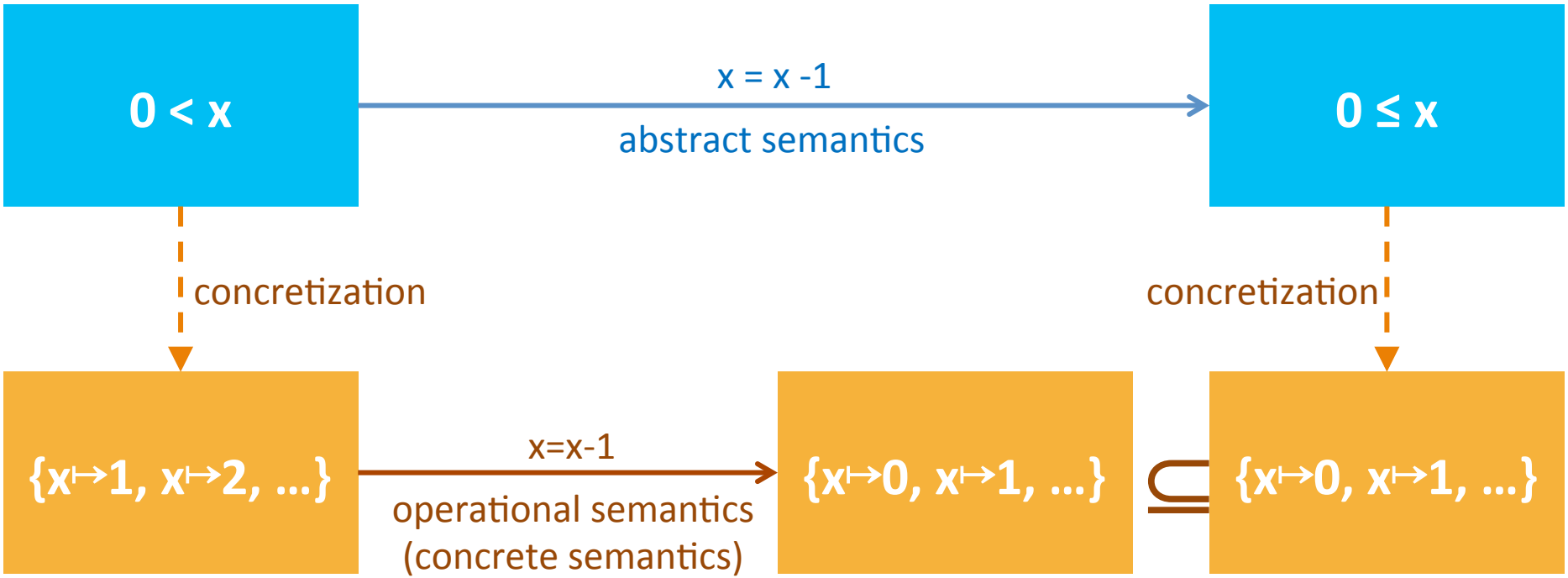
  $(2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$

- The collecting semantics accumulates the (possibly infinite) sets of states generated during the execution
  - Not computable in general

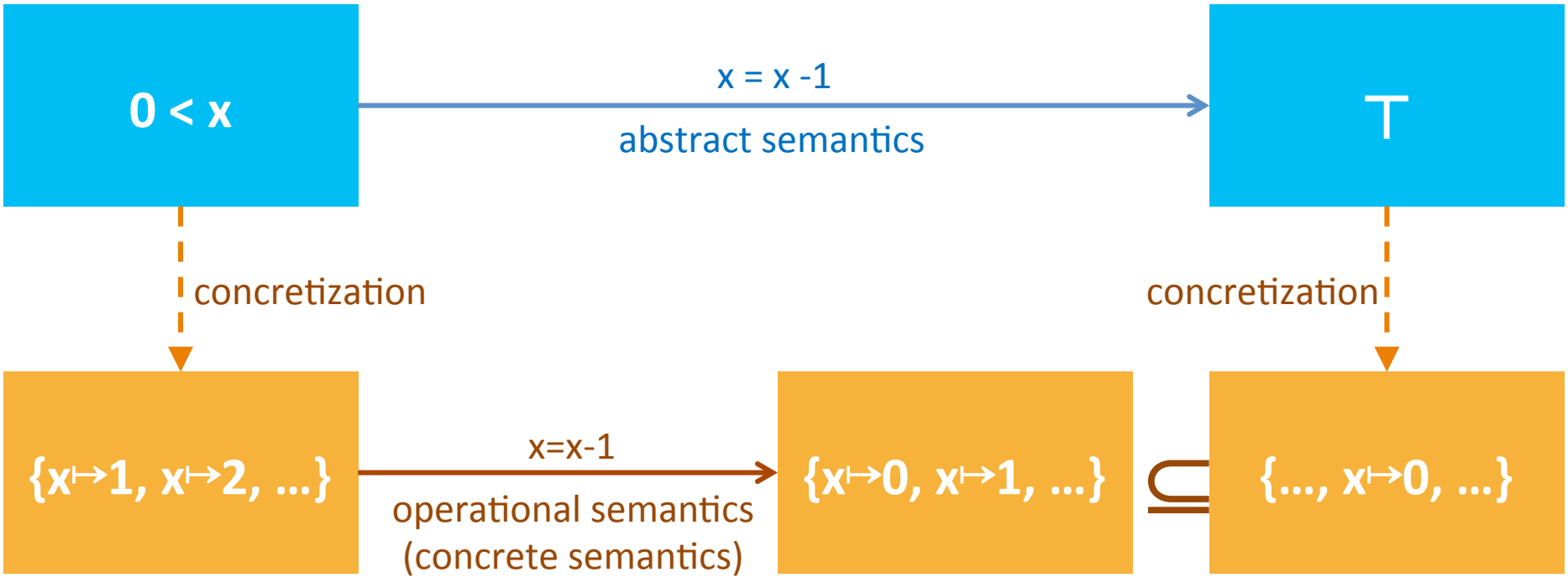
Abstract (conservative) interpretation



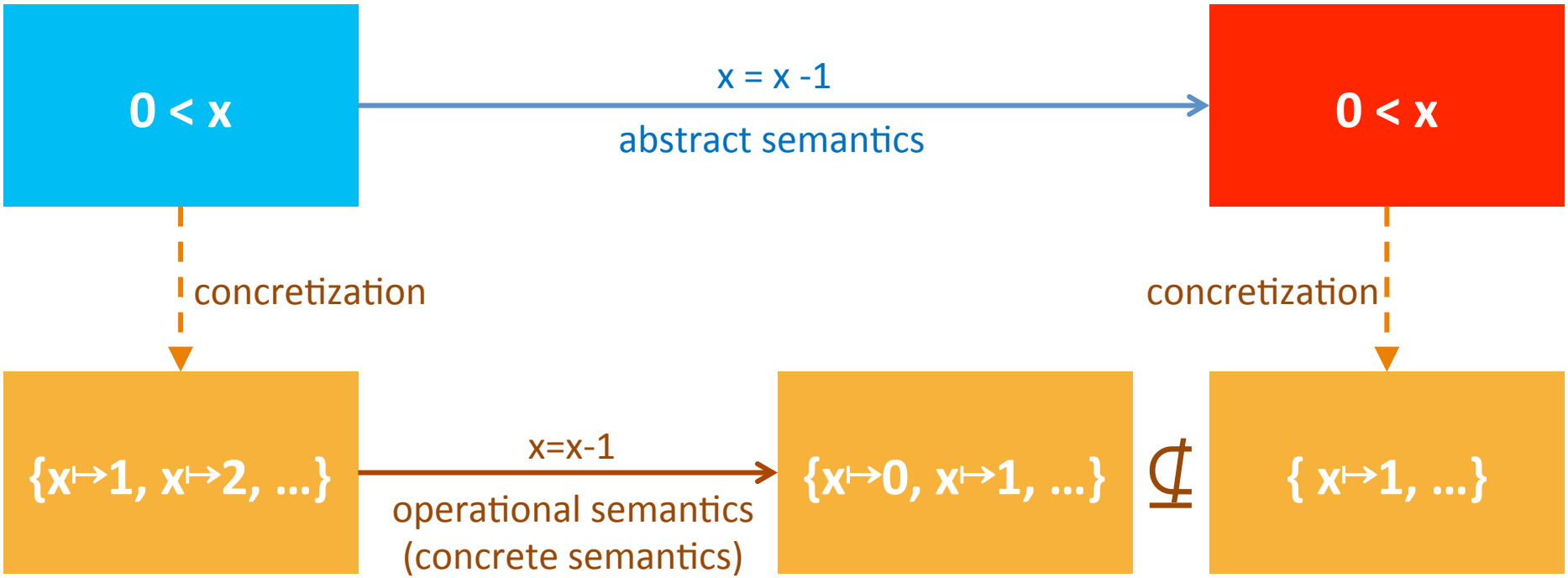
Abstract (conservative) interpretation



Abstract (conservative) interpretation



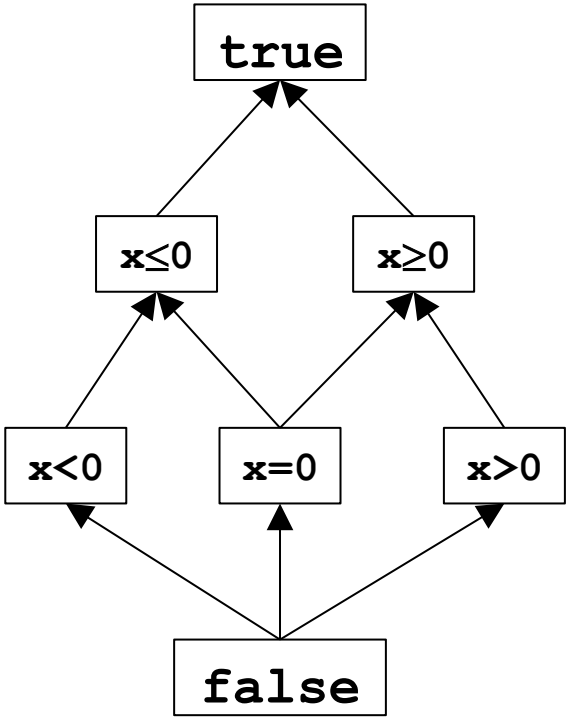
Abstract (non-conservative) interpretation

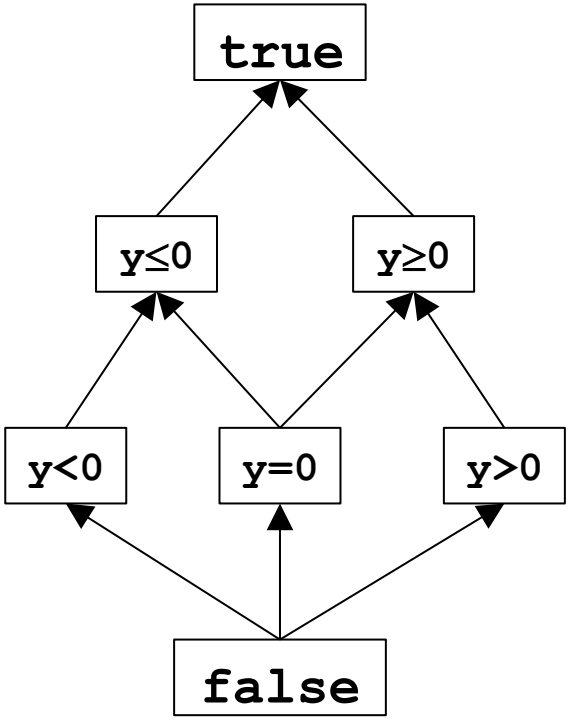


But …

- what if we have x & y?
  - Define lattice (semantics) for each variable
  - Compose lattices
    - Goal: compositional definition
- What if we have more than 1 statement?
  - Define semantics for entire program via CFG
  - Different “abstract states” at every CFG node

One lattice per variable



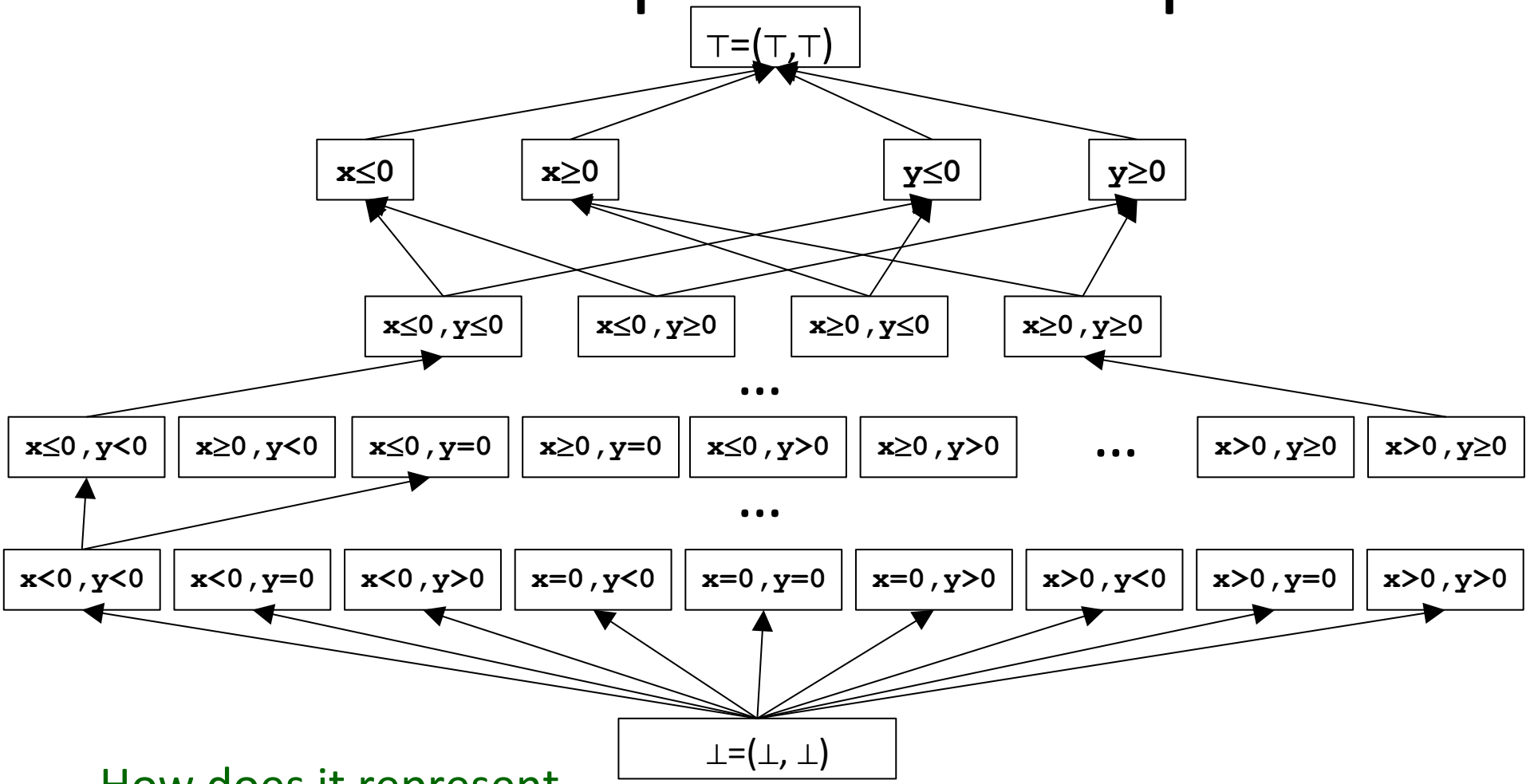


How can we compose them?

Cartesian product of complete lattices

- For two complete lattices
  $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- Define the poset
  $L_{cart} = (D_1 \times D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$
  as follows:
  - $(x_1, x_2) \sqsubseteq_{cart} (y_1, y_2)$ *iff* $x_1 \sqsubseteq_1 y_1\ x_2 \sqsubseteq_2 y_2$
  - $\sqcup_{cart} = ?$ $\sqcap_{cart} = ?$ $\bot_{cart} = ?$ $\top_{cart} = ?$
- **Lemma:** L is a complete lattice
- Define the Cartesian constructor $L_{cart} = \text{Cart}(L_1, L_2)$

Cartesian product example

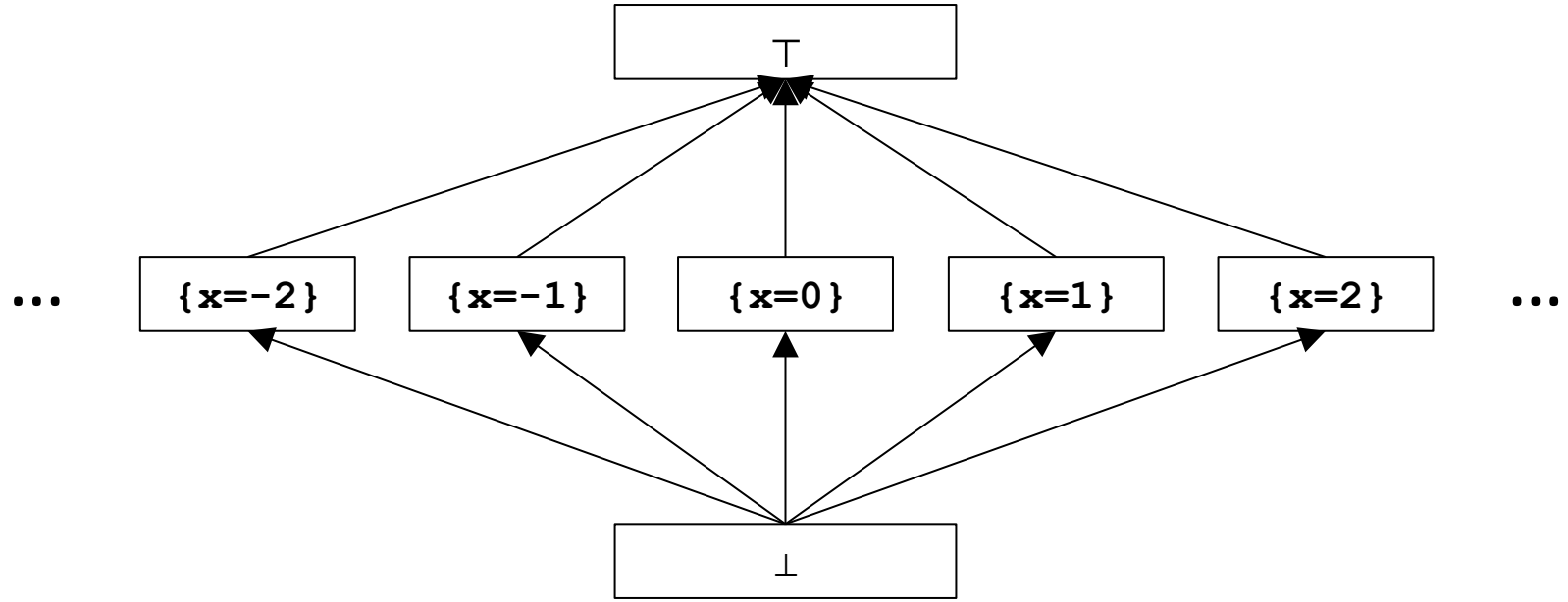


How does it represent
$(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

Disjunctive completion

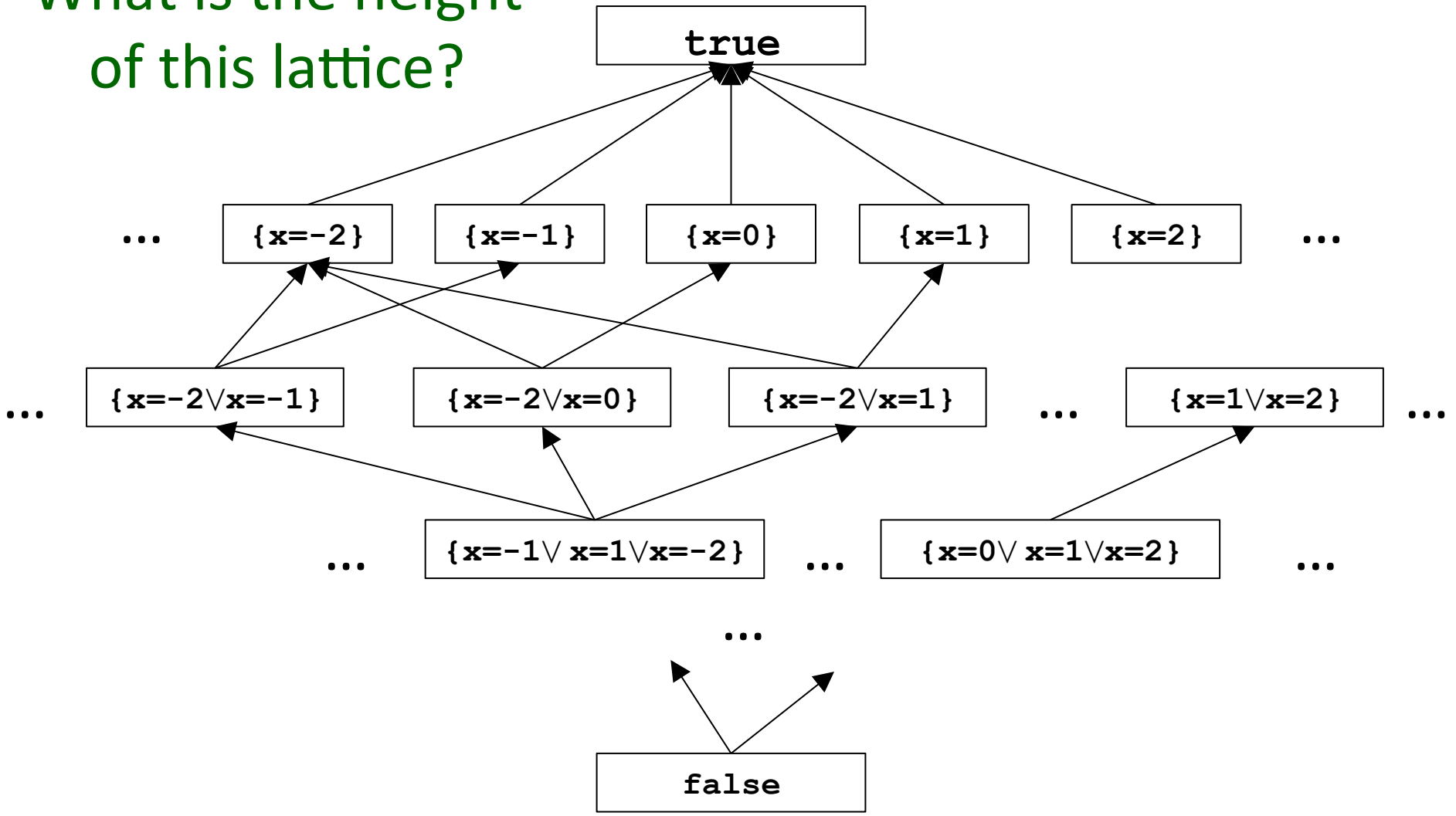
- For a complete lattice
  $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Define the powerset lattice
  $L_\vee = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
  $\sqsubseteq_\vee$ = ? $\sqcup_\vee$ = ? $\sqcap_\vee$ = ? $\bot_\vee$ = ? $\top_\vee$ = ?
- **Lemma:** $L_\vee$ is a complete lattice
- $L_\vee$ contains all subsets of D, which can be thought of as disjunctions of the corresponding predicates
- Define the disjunctive completion constructor
  $L_\vee$ = Disj(L)

The base lattice CP



The disjunctive completion of CP

What is the height of this lattice?



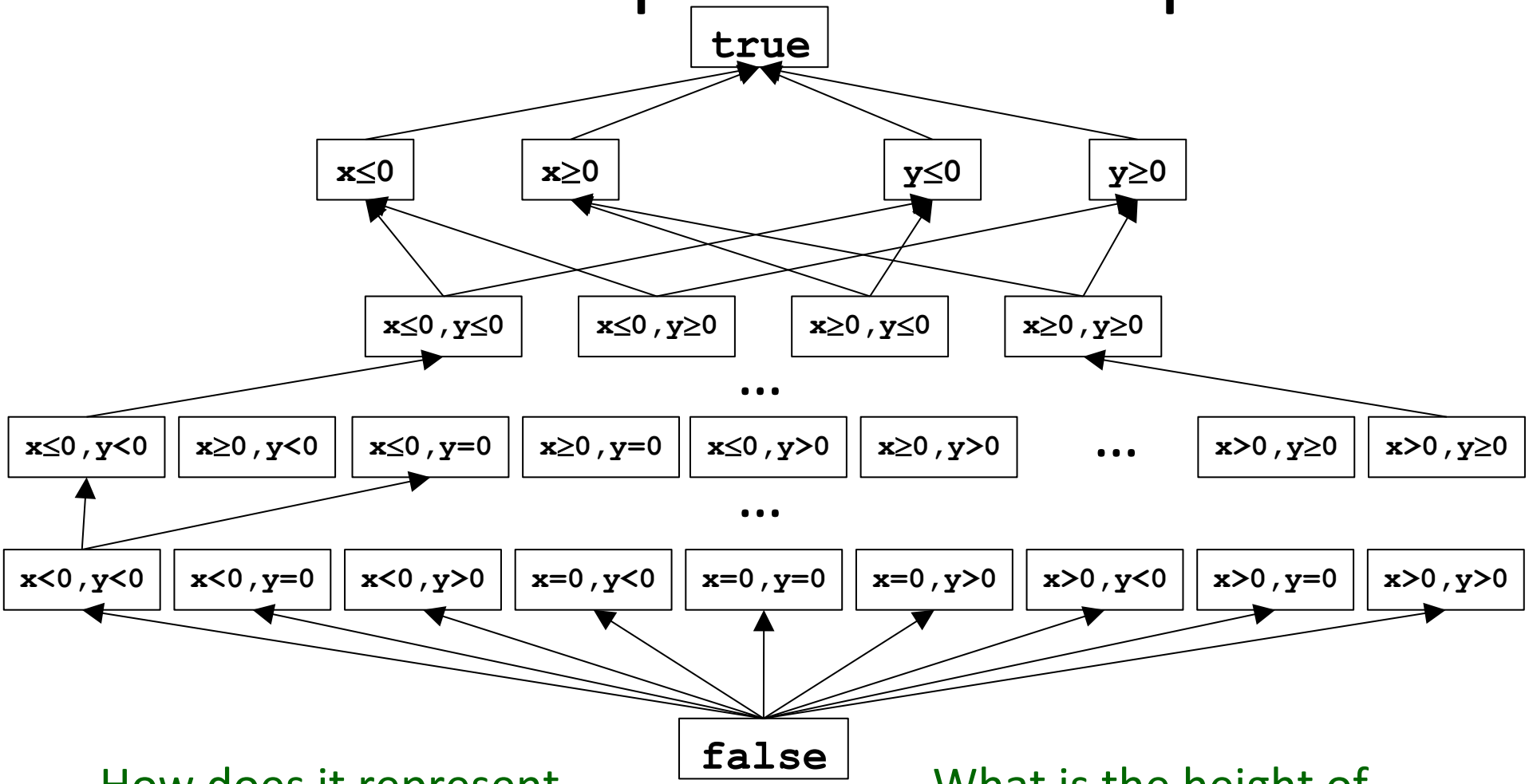
Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = ?

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = Disj(Cart(L_1, L_2))
- **Lemma:** *L* is a complete lattice
- What does it buy us?

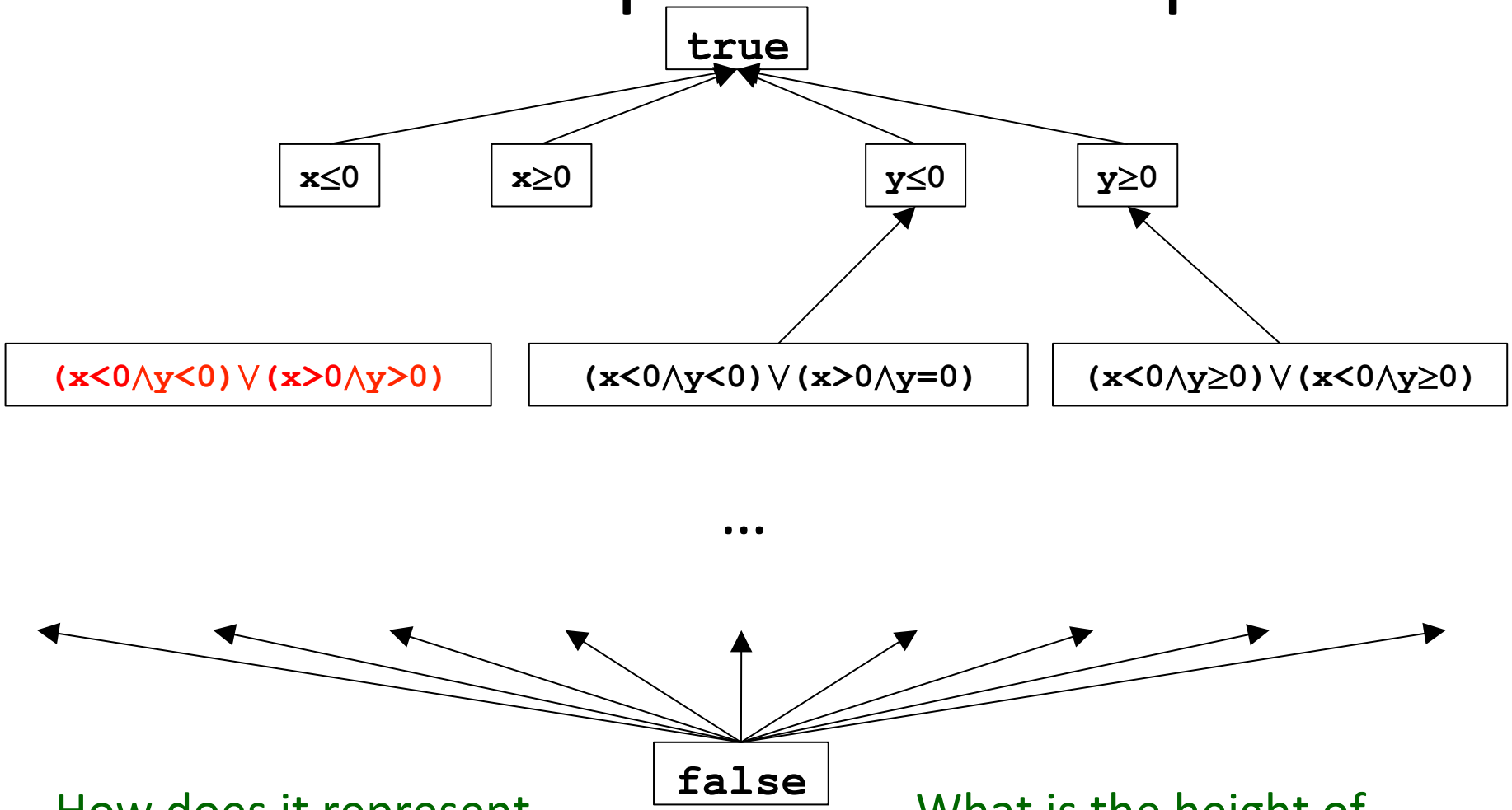
Cartesian product example



How does it represent $(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

What is the height of this lattice?

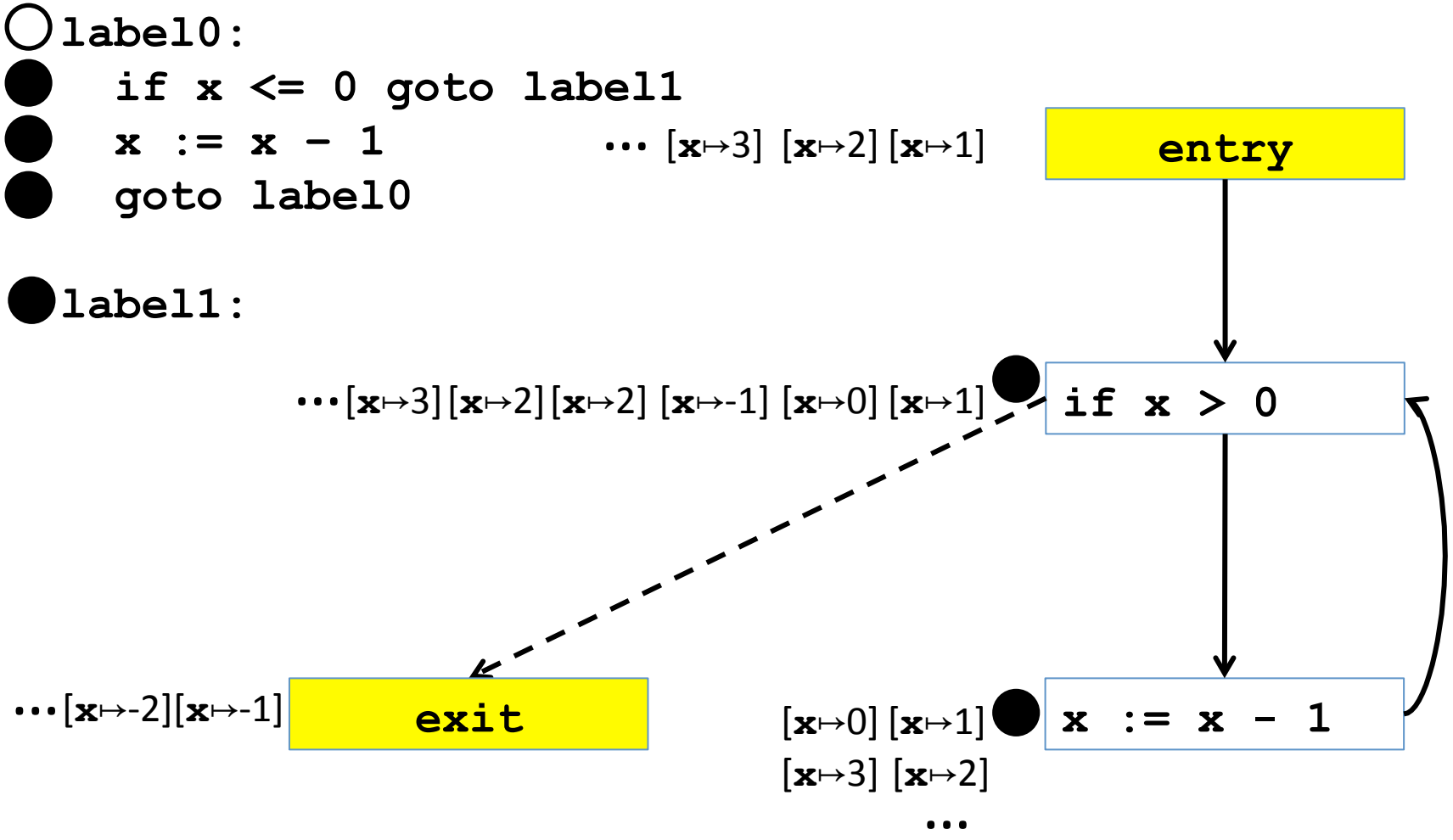
Relational product example



How does it represent $(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

What is the height of this lattice?

Collecting semantics



Defining the collecting semantics

- How should we represent the set of states at a given control-flow node by a lattice?
- How should we represent the sets of states at all control-flow nodes by a lattice?

Finite maps

- For a complete lattice
  $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
  and finite set V
- Define the poset
  $L_{V\to L} = (V \to D, \sqsubseteq_{V\to L}, \sqcup_{V\to L}, \sqcap_{V\to L}, \bot_{V\to L}, \top_{V\to L})$
  as follows:
  - $f_1 \sqsubseteq_{V\to L} f_2$ iff for all $v \in V$
    $f_1(v) \sqsubseteq f_2(v)$
  - $\sqcup_{V\to L}$ = ? $\sqcap_{V\to L}$ = ? $\bot_{V\to L}$ = ? $\top_{V\to L}$ = ?
- **Lemma:** L is a complete lattice
- Define the map constructor $L_{V\to L}$ = Map(V, L)

The collecting lattice

- Lattice for a given control-flow node *v*:

  ?

- Lattice for entire control-flow graph with nodes *V*:

  ?

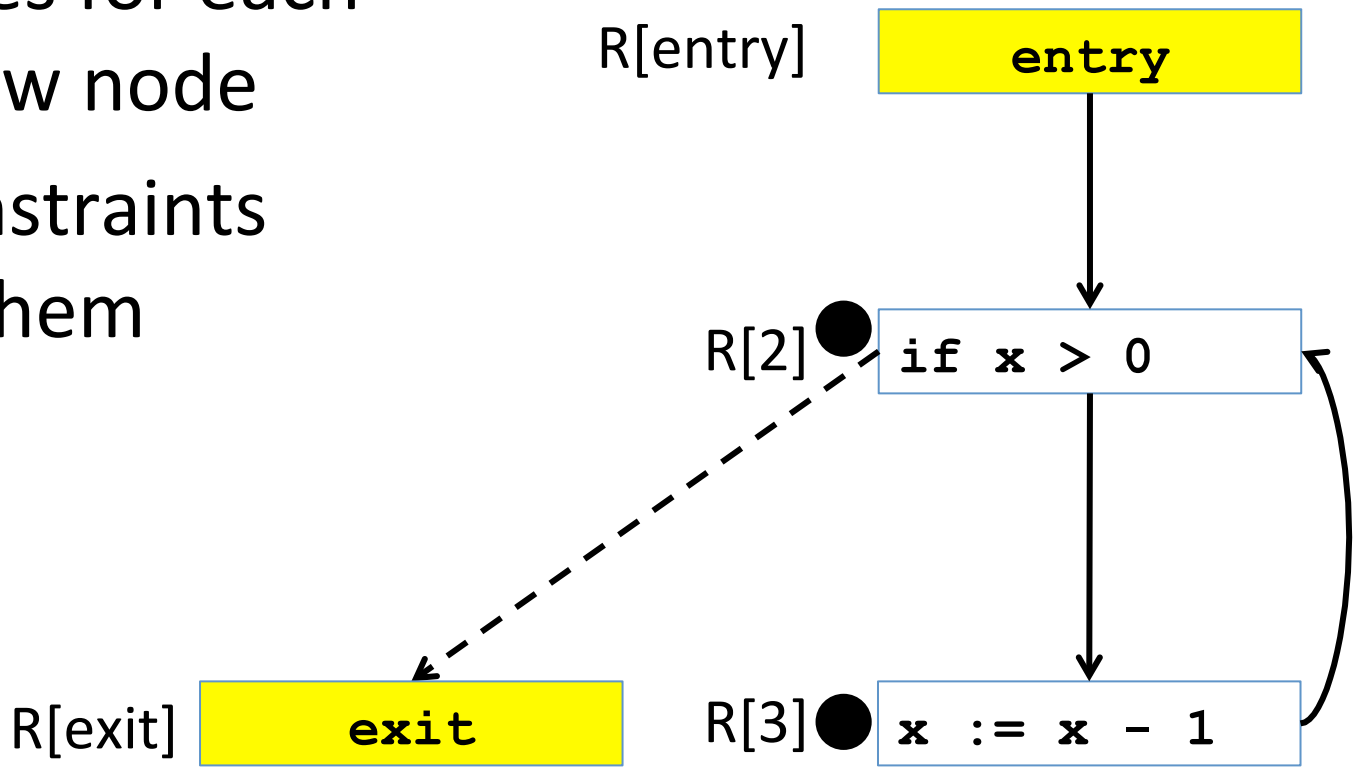
- We will use this lattice as a baseline for static analysis and define abstractions of its elements

The collecting lattice

- Lattice for a given control-flow node v:
  $L_v = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- Lattice for entire control-flow graph with nodes V:
  $$L_{CFG} = \text{Map}(V, L_v)$$
- We will use this lattice as a baseline for static analysis and define abstractions of its elements

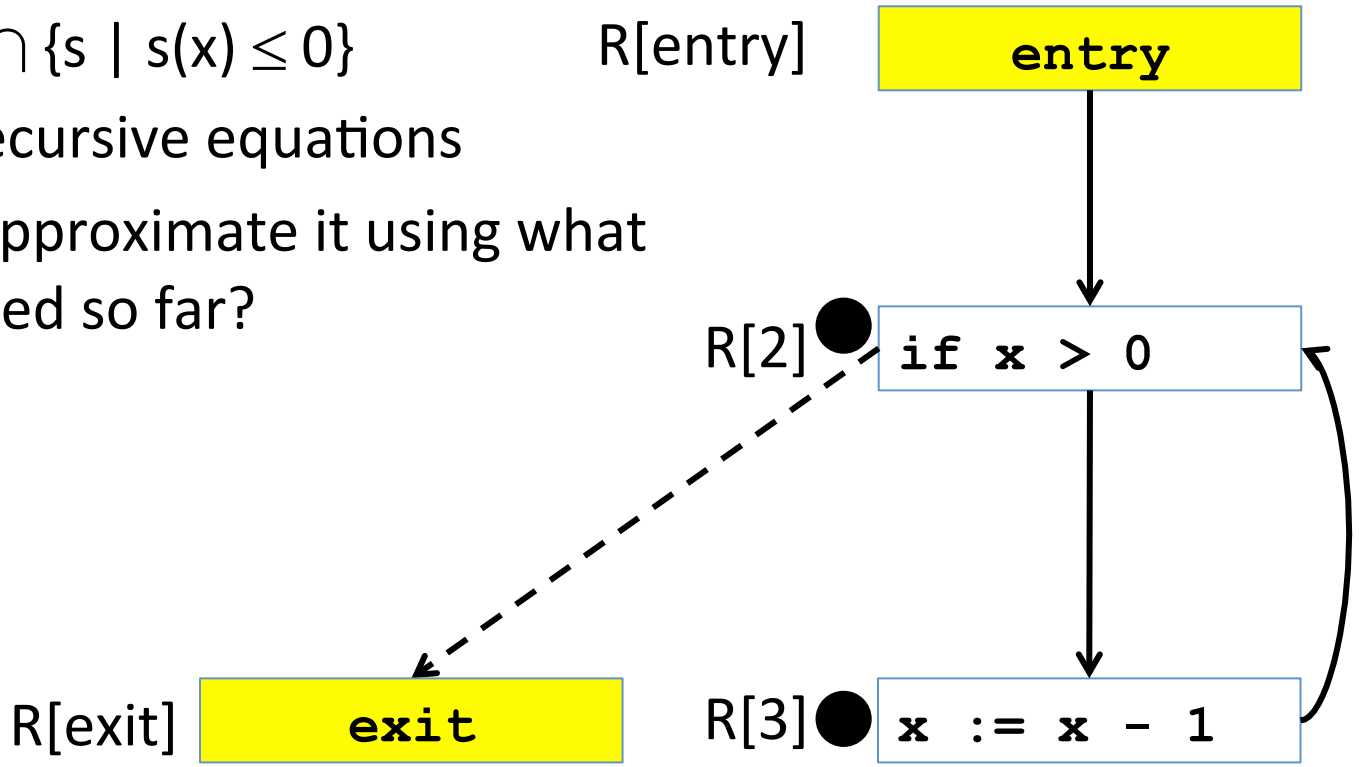
Equational definition of the semantics

- Define variables of type set of states for each control-flow node
- Define constraints between them



Equational definition of the semantics

- R[2] = R[entry] ∪ ⟦**x:=x-1**⟧ R[3]
- R[3] = R[2] ∩ {s | s(x) > 0}
- R[exit] = R[2] ∩ {s | s(x) ≤ 0}
- A system of recursive equations
- How can we approximate it using what we have learned so far?

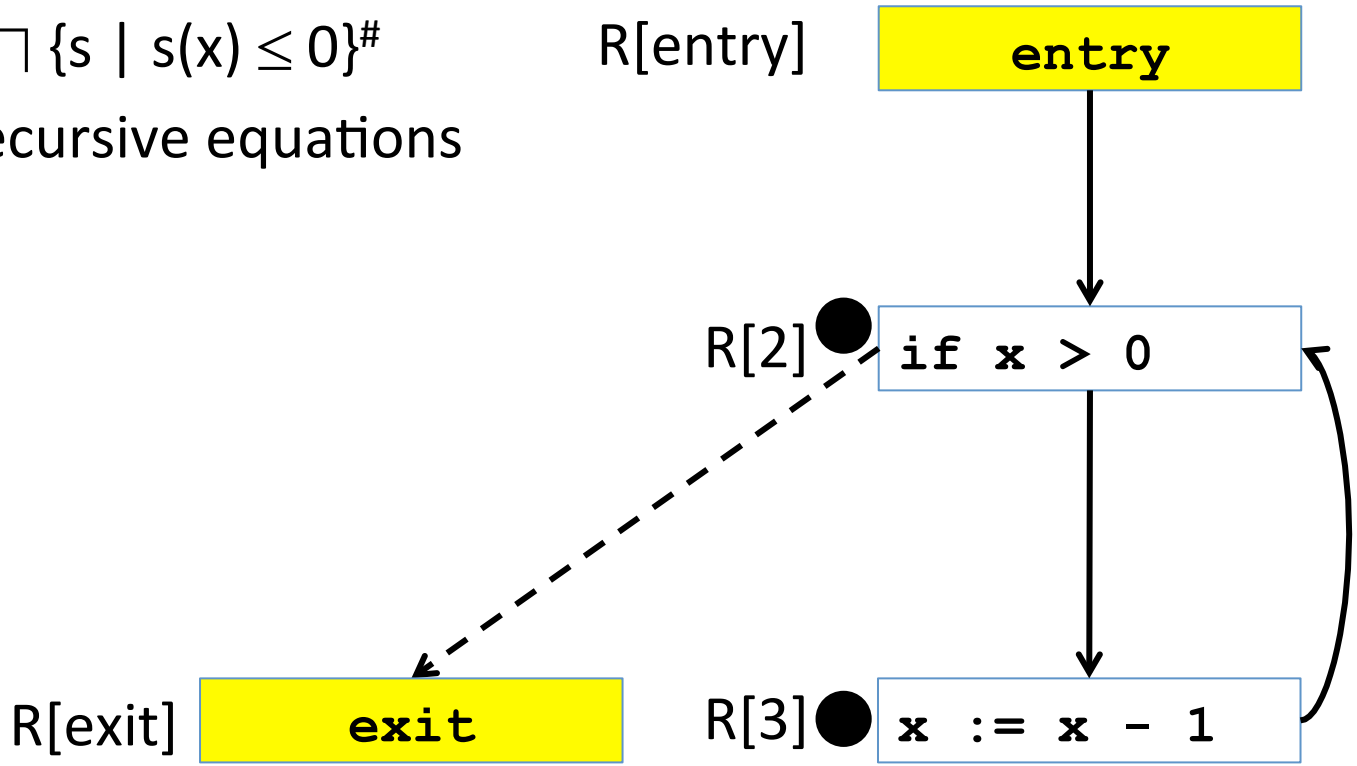


An abstract semantics

- R[2] = R[entry] ⊔ ⟦**x:=x-1**⟧$^{\#}$ R[3]
- R[3] = R[2] ⊓ {s | s(x) > 0}$^{\#}$
- R[exit] = R[2] ⊓ {s | s(x) ≤ 0}$^{\#}$
- A system of recursive equations

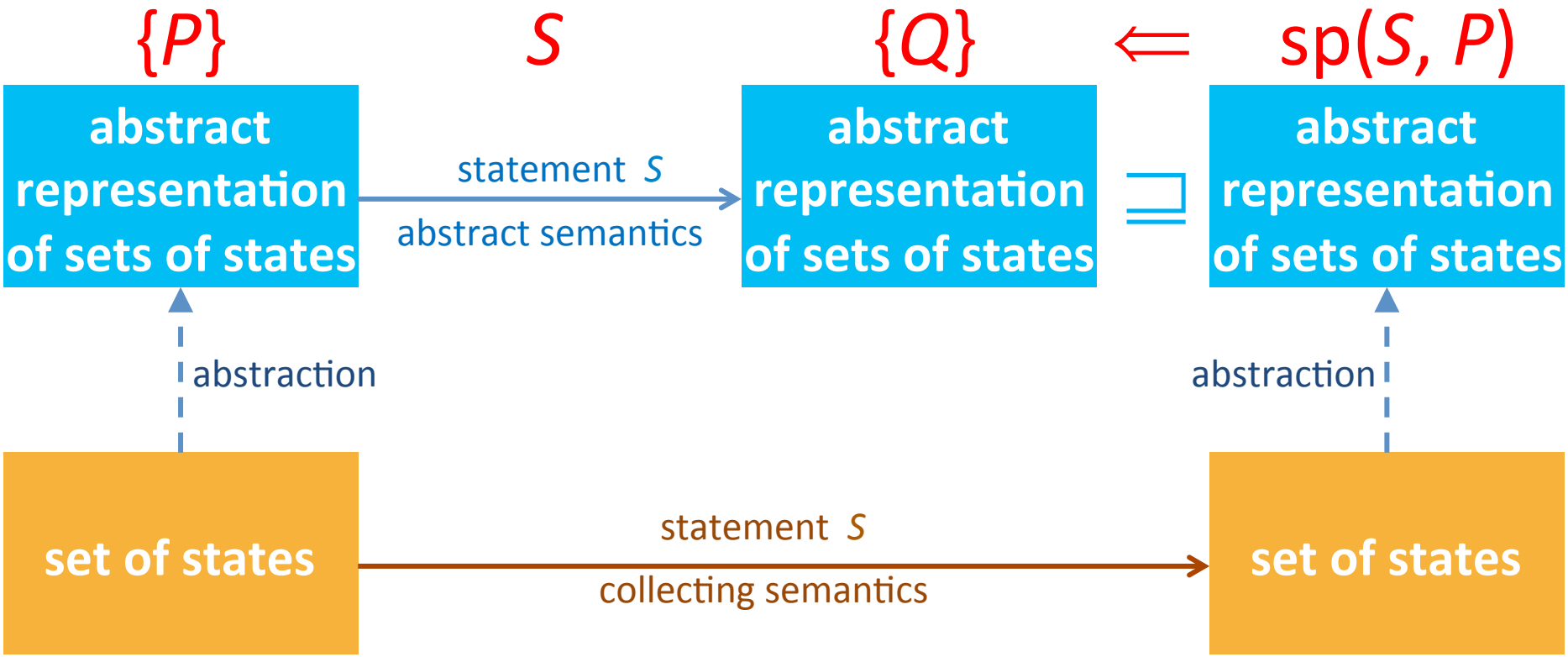
Abstract transformer for **x:=x-1**

Abstract representation of {s | s(x) < 0}

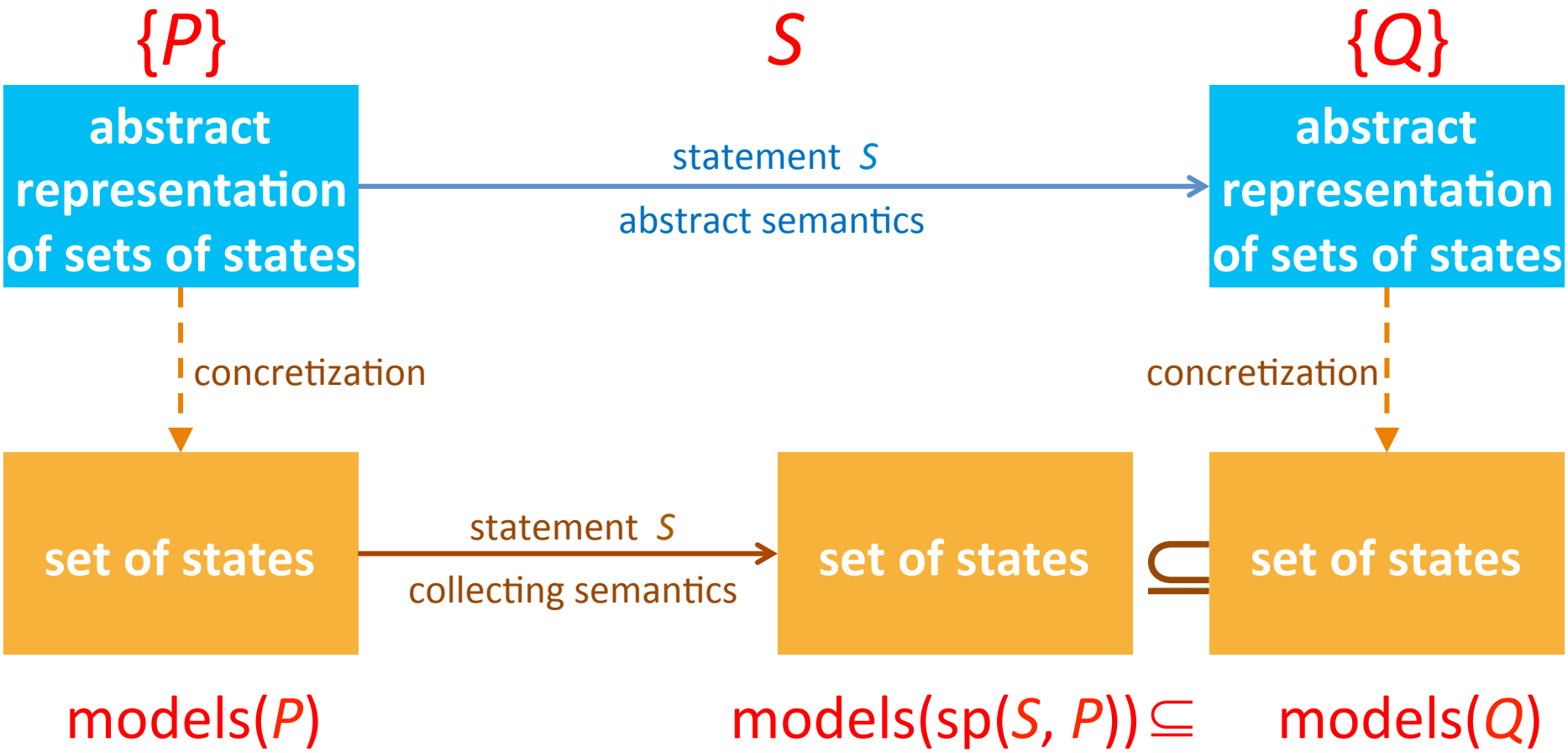


Abstract interpretation via abstraction

generalizes axiomatic verification



Abstract interpretation via concretization



Required knowledge

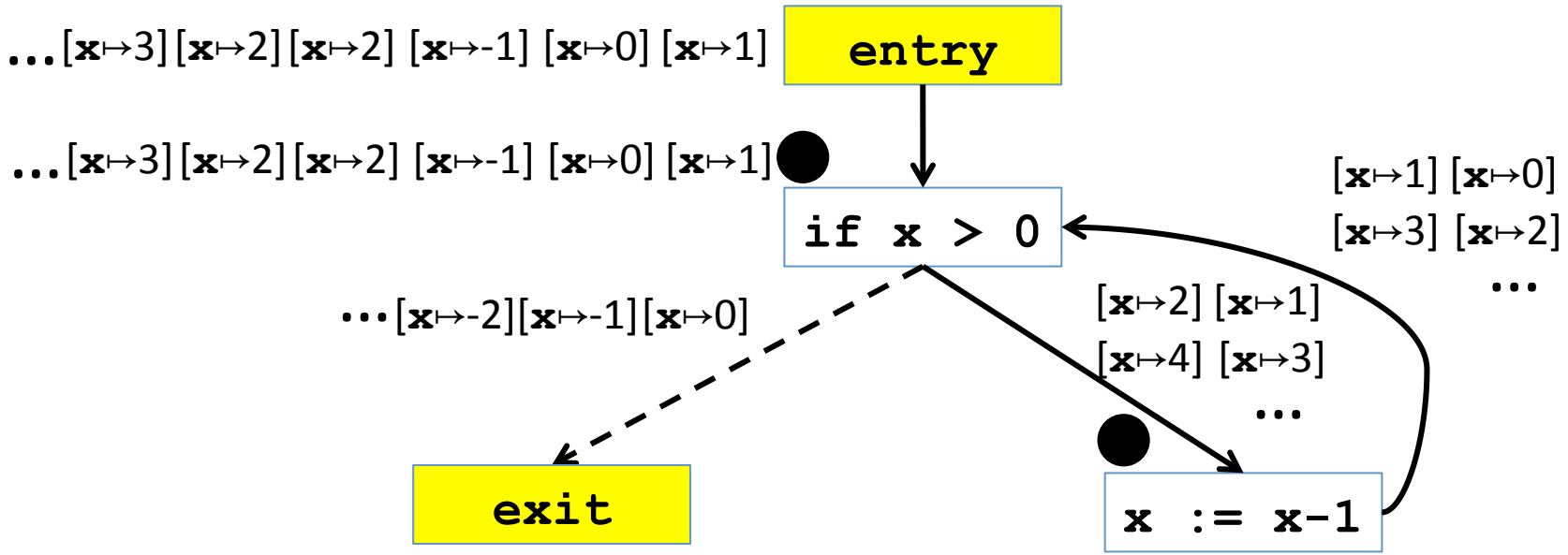
- ✓ Collecting semantics
- ✓ Abstract semantics
- Connection between collecting semantics and abstract semantics
- Algorithm to compute abstract semantics

The collecting lattice (sets of states)

- Lattice for a given control-flow node *v*:
  $L_v = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- Lattice for entire control-flow graph with nodes *V*:
  $$L_{CFG} = \text{Map}(V, L_v)$$
- We will use this lattice as a baseline for static analysis and define abstractions of its elements

Collecting semantics example

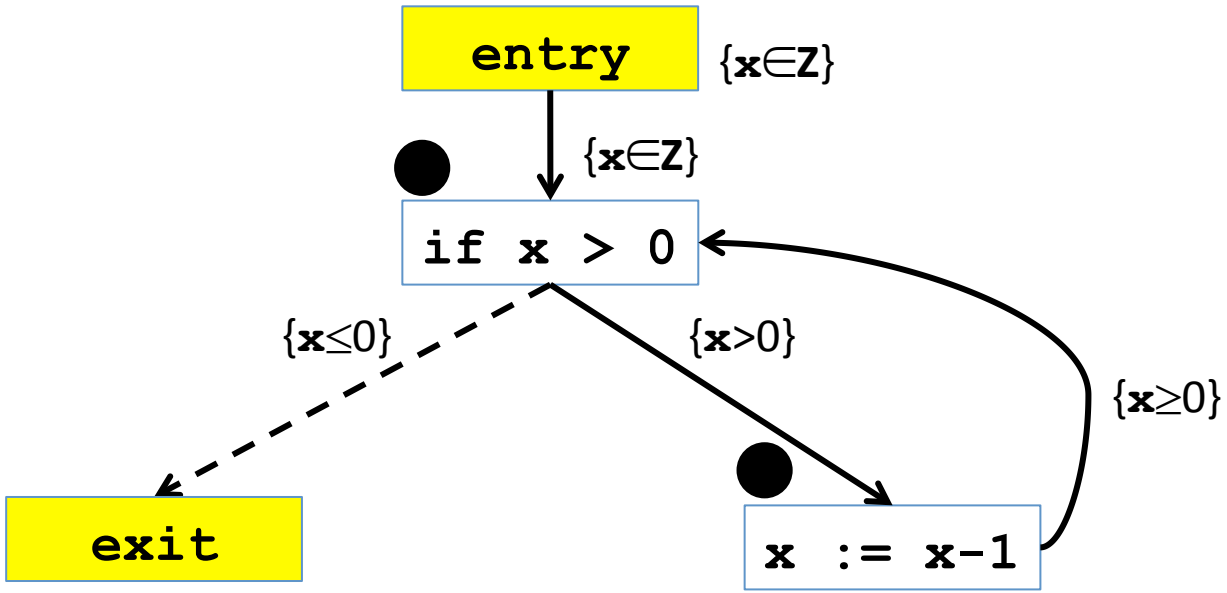
```
label0:
  if x <= 0 goto label1
  x := x - 1
  goto label0

label1:
```



Shorthand notation

○ `label0:`
● `  if x <= 0 goto label1`
● `  x := x - 1`
● `  goto label0`

● `label1:`

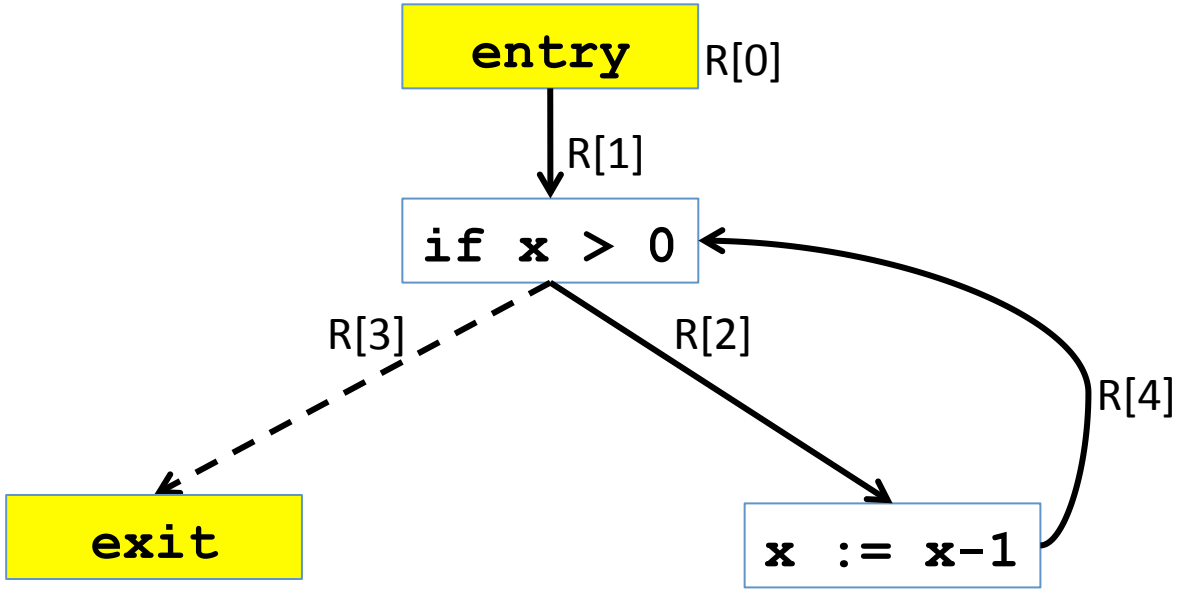


Equational definition example

- A vector of variables R[0, 1, 2, 3, 4]
- R[0] = {**x**∈**Z**} // established input
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]
- A (recursive) system of equations

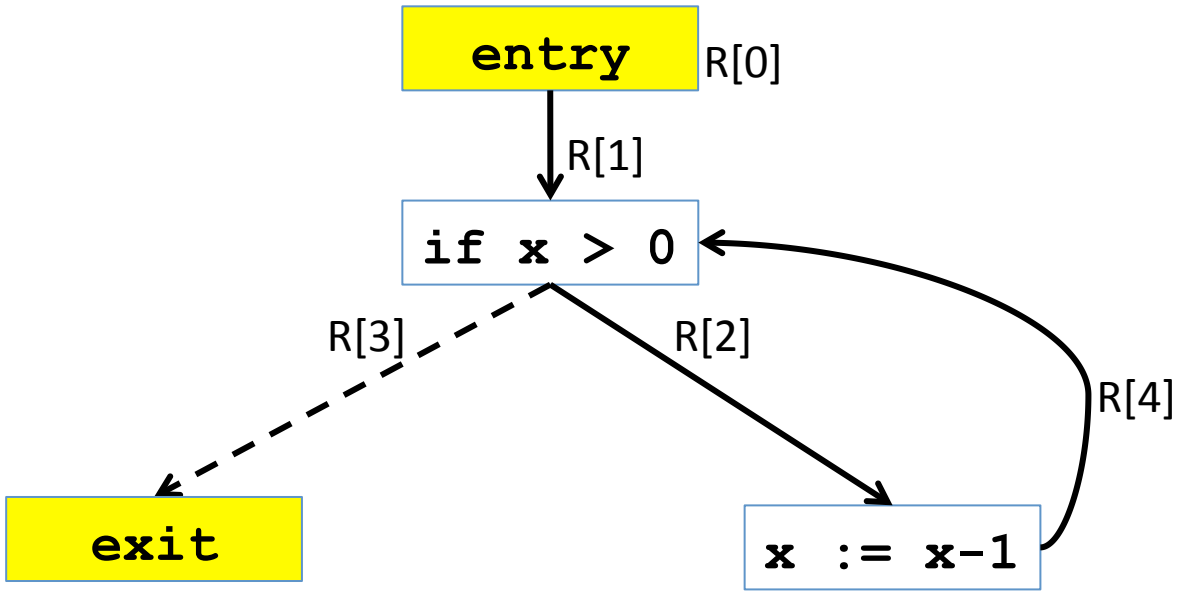
Semantic function for assume x>0

Semantic function for **x:=x-1** lifted to sets of states



General definition

- A vector of variables R[0, …, k] one per input/output of a node
  - R[0] is for entry
- For node n with multiple predecessors add equation
  R[n] = ∪{R[k] | k is a predecessor of n}
- For an atomic operation node R[m] *S* R[n] add equation
  R[n] = ⟦*S*⟧ R[m]
- Treat true/false branches of conditions as atomic statements
  `assume` *bexpr* and `assume` ¬*bexpr*



Equation systems in general

- Let L be a complete lattice $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Let R be a vector of variables $R[0, \ldots, n] \in D \times \ldots \times D$
- Let F be a vector of functions of the type
  $F[i] : R[0, \ldots, n] \rightarrow R[0, \ldots, n]$
- A system of equations
  $R[0] = f[0](R[0], \ldots, R[n])$
  ...
  $R[n] = f[n](R[0], \ldots, R[n])$
- In vector notation R = F(R)

Equation systems in general

- Let L be a complete lattice $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Let R be a vector of variables $R[0, \ldots, n] \in D \times \ldots \times D$
- Let F be a vector of functions of the type
  $F[i] : R[0, \ldots, n] \rightarrow R[0, \ldots, n]$
- A system of equations
  $R[0] = f[0](R[0], \ldots, R[n])$
  ...
  $R[n] = f[n](R[0], \ldots, R[n])$
- In vector notation R = F(R)
- Questions:
  1. Does a solution always exist?
  2. If so, is it unique?
  3. If so, is it computable?

Monotone functions

- Let $L_1=(D_1, \sqsubseteq)$ and $L_2=(D_2, \sqsubseteq)$ be two posets
- A function $f : D_1 \rightarrow D_2$ is monotone if for every pair $x, y \in D_1$

  $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$
- A special case: $L_1=L_2=(D, \sqsubseteq)$

  $f : D \rightarrow D$

Important cases of monotonicity

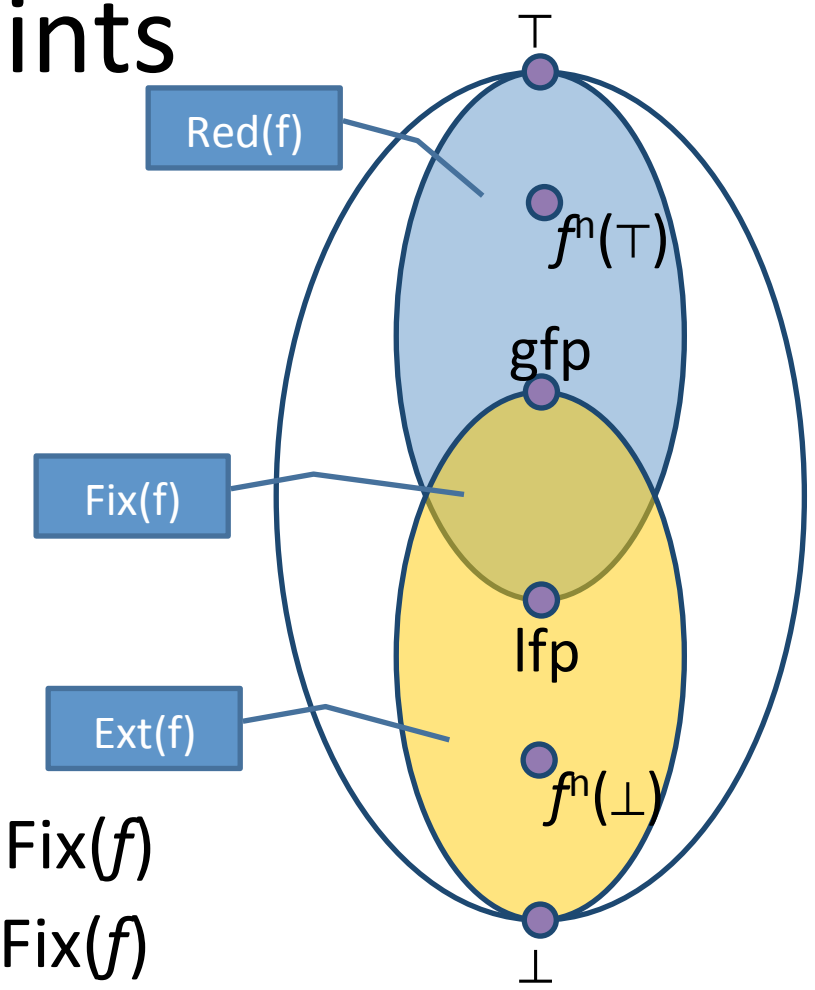
- Join: $f(X, Y) = X \sqcup Y$
  Prove it!
- For a set X and any function g
  $F(X) = \{ g(x) \mid x \in X \}$
  Prove it!
- Notice that the collecting semantics function is defined in terms of
  - Join (set union)
  - Semantic function for atomic statements lifted to sets of states

Extensive/reductive functions

- Let $L=(D, \sqsubseteq)$ be a poset
- A function $f : D \rightarrow D$ is extensive if for every pair $x \in D$, we have that $x \sqsubseteq f(x)$
- A function $f : D \rightarrow D$ is reductive if for every pair $x \in D$, we have that $x \sqsubseteq f(x)$

Fixed-points

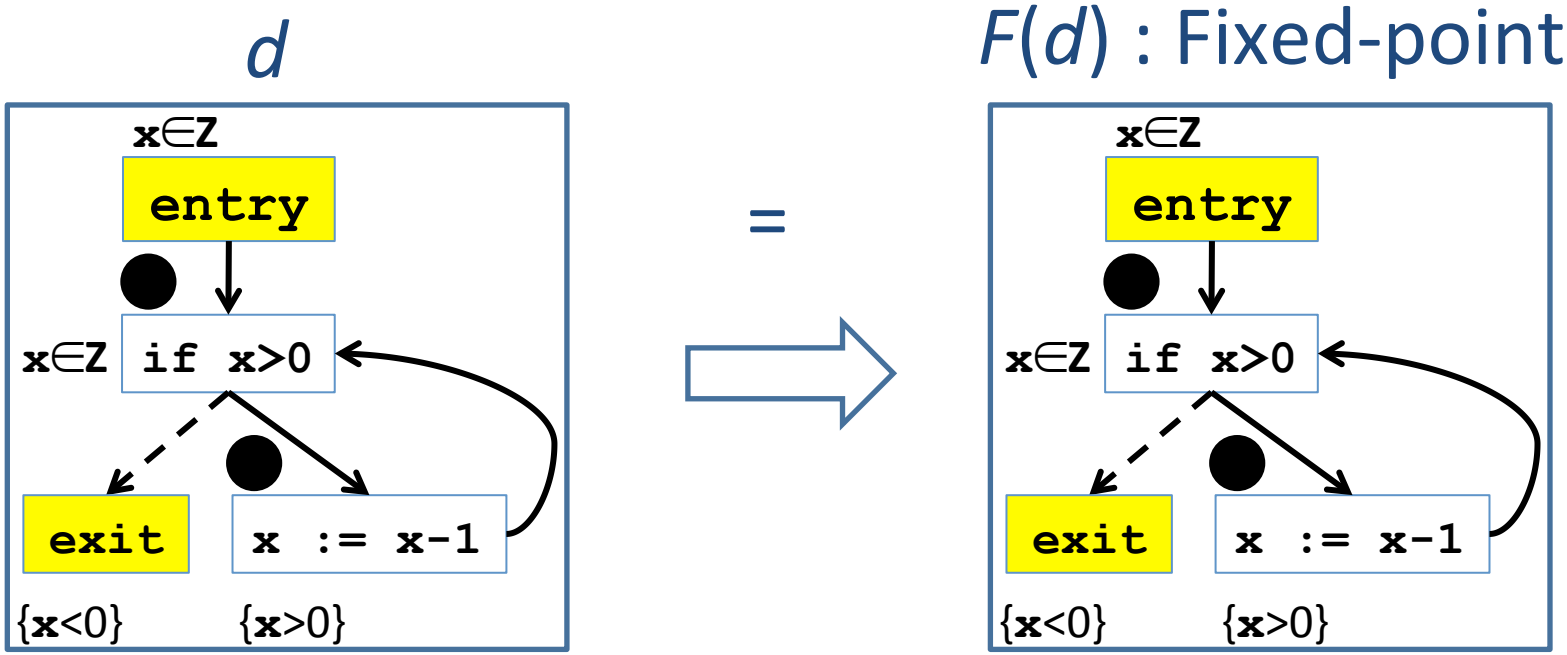
- $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- $f : D \rightarrow D$ **monotone**
- $\text{Fix}(f) = \{ d \mid f(d) = d \}$
- $\text{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\text{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\text{lfp}(f) = \sqcap\text{Fix}(f) = \sqcap\text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \sqcup\text{Fix}(f) = \sqcup\text{Ext}(f) \in \text{Fix}(f)$



1. Does a solution always exist? Yes
2. If so, is it unique? No, but it has least/greatest solutions
3. If so, is it computable? Under some conditions…

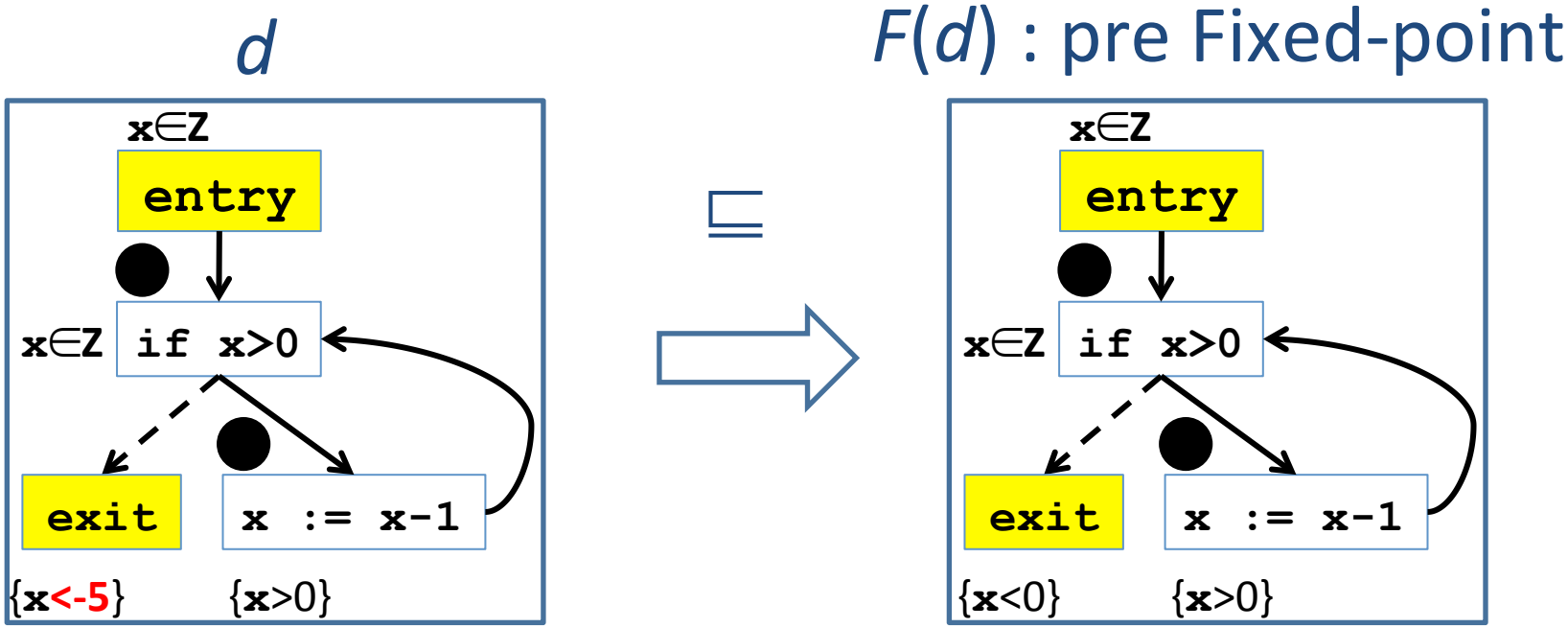
Fixed point example for program

- R[0] = {**x**∈**Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]



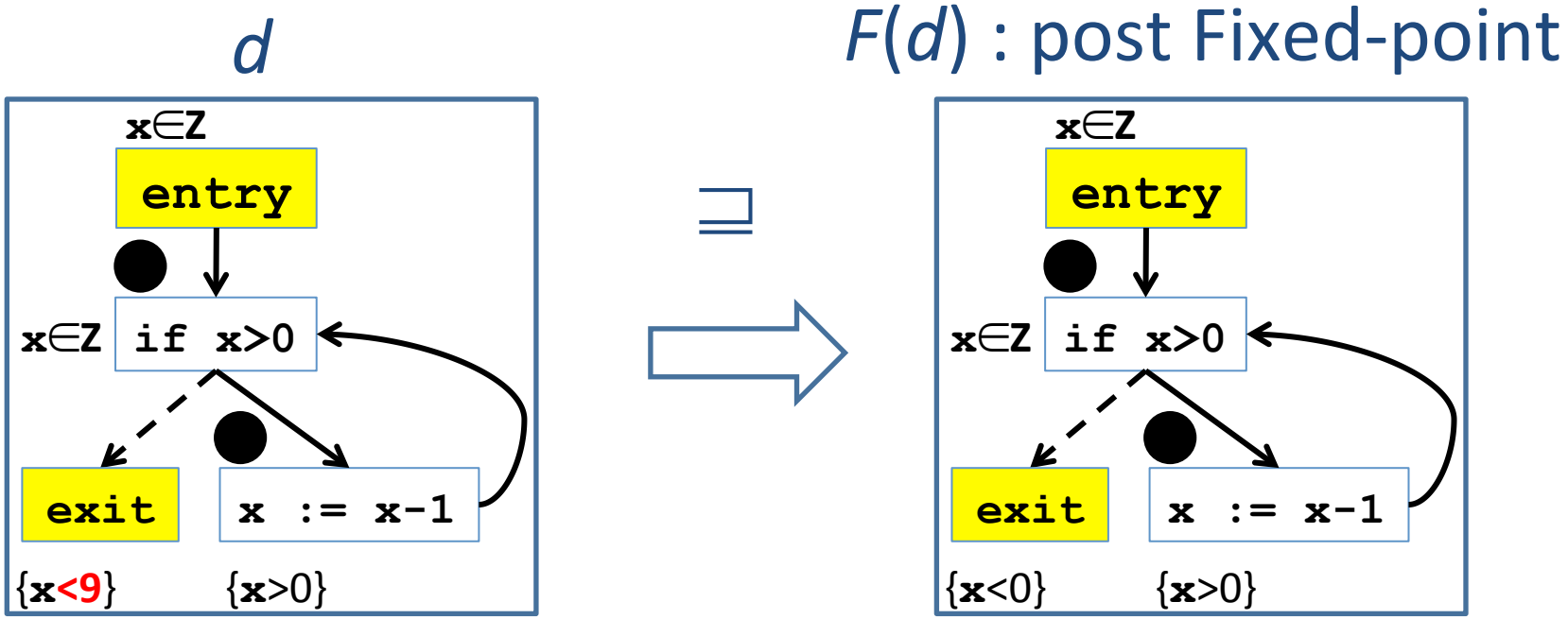
Fixed point example for program

- R[0] = {**x**∈**Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]



Fixed point example for program

- R[0] = {**x**∈**Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]



Continuity and ACC condition

- Let $L = (D, \sqsubseteq, \sqcup, \bot)$ be a complete partial order
  - Every ascending chain has an upper bound
- A function f is continuous if for every increasing chain $Y \subseteq D^*$,

$$f(\sqcup Y) = \sqcup \{ f(y) \mid y \in Y \}$$

- L satisfies the ascending chain condition (ACC) if every ascending chain eventually stabilizes:

$$d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n = d_{n+1} = \ldots$$

Fixed-point theorem [Kleene]

- Let $L = (D, \sqsubseteq, \sqcup, \bot)$ be a complete partial order and a **continuous** function $f: D \to D$ then

$$\mathrm{lfp}(f) = \bigsqcup_{n \in \mathbb{N}} f^n(\bot)$$

- **Lemma:** Monotone functions on posets satisfying ACC are continuous
  Proof:
  1. Every ascending chain Y eventually stabilizes
     $d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n = d_{n+1} = \ldots$
     hence d_n is the least upper bound of $\{d_0, d_1, \ldots, d_n\}$,
     thus $f(\bigsqcup Y) = f(d_n)$
  2. From monotonicity of f
     $f(d_0) \sqsubseteq f(d_1) \sqsubseteq \ldots \sqsubseteq f(d_n) = f(d_{n+1}) = \ldots$
     Hence $f(d_n)$ is the least upper bound of $\{f(d_0), f(d_1), \ldots, f(d_n)\}$,
     thus $\bigsqcup\{ f(y) \mid y \in Y \} = f(d_n)$

Resulting algorithm

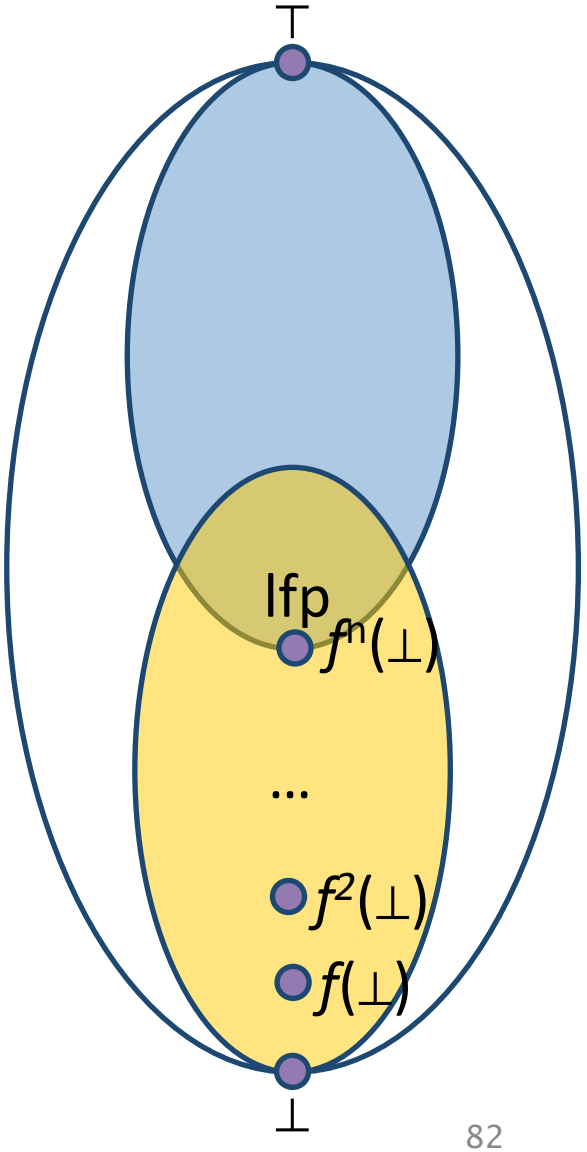
- Kleene's fixed point theorem gives a constructive method for computing the lfp

Mathematical definition

$$\text{lfp}(f) = \bigsqcup_{n \in N} f^n(\bot)$$

Algorithm

$d := \bot$
while $f(d) \neq d$ **do**
  $d := d \sqcup f(d)$
return d



Chaotic iteration

- Input:
  - A cpo $L = (D, \sqsubseteq, \sqcup, \bot)$ satisfying ACC
  - $L^n = L \times L \times \ldots \times L$
  - A monotone function $f : D^n \rightarrow D^n$
  - A system of equations $\{ X[i] \mid f(X) \mid 1 \leq i \leq n \}$
- Output: lfp(f)
- A worklist-based algorithm

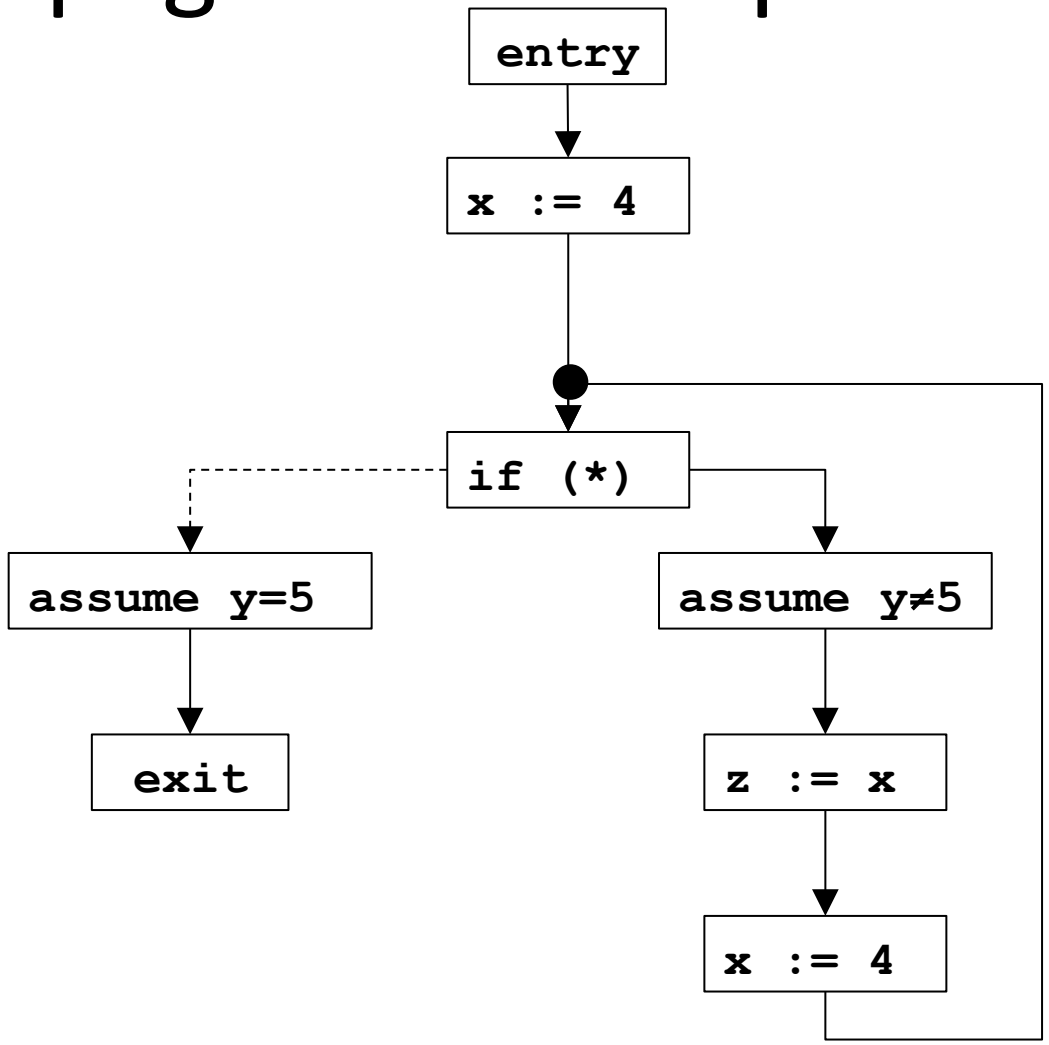
```
for i:=1 to n do
   X[i] := ⊥
WL = {1,…,n}
while WL ≠ ∅ do
   j := pop WL // choose index non-deterministically
   N := F[i](X)
   if N ≠ X[i] then
      X[i] := N
      add all the indexes that directly depend on i to WL
      (X[j] depends on X[i] if F[j] contains X[i])
return X
```

Chaotic iteration for static analysis

- Specialize chaotic iteration for programs
- Create a CFG for program
- Choose a cpo of properties for the static analysis to infer: $L = (D, \sqsubseteq, \sqcup, \bot)$
- Define variables R[0,…,*n*] for input/output of each CFG node such that $R[i] \in D$
- For each node v let v_{out} be the variable at the output of that node:
  $v_{out} = F[v](\sqcup u \mid (u,v) \text{ is a CFG edge})$
  - Make sure each F[v] is monotone
- Variable dependence determined by outgoing edges in CFG

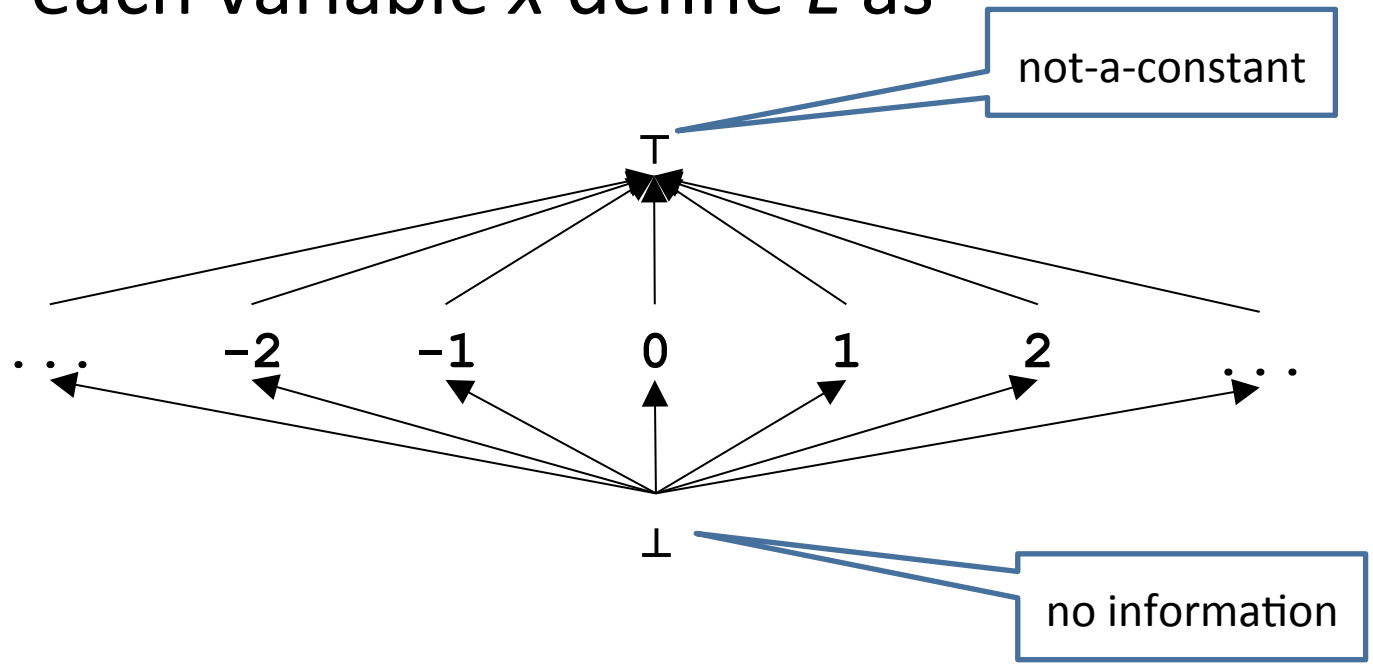
Constant propagation example

```
x := 4;
while (y≠5) do
     z := x;
     x := 4
```



Constant propagation lattice

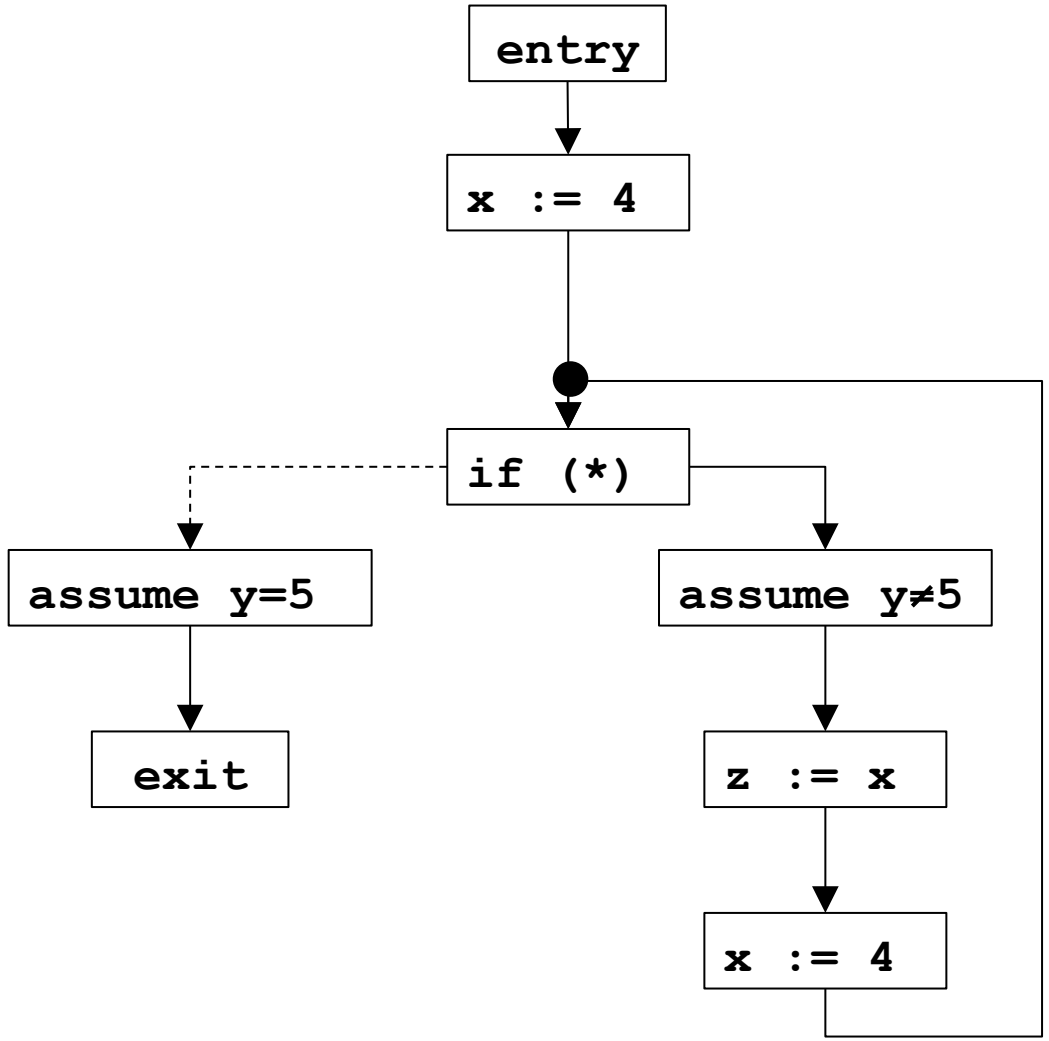
- For each variable *x* define *L* as



- For a set of program variables Var=$x_1,\ldots,x_n$
  $L^n = L \times L \times \ldots \times L$

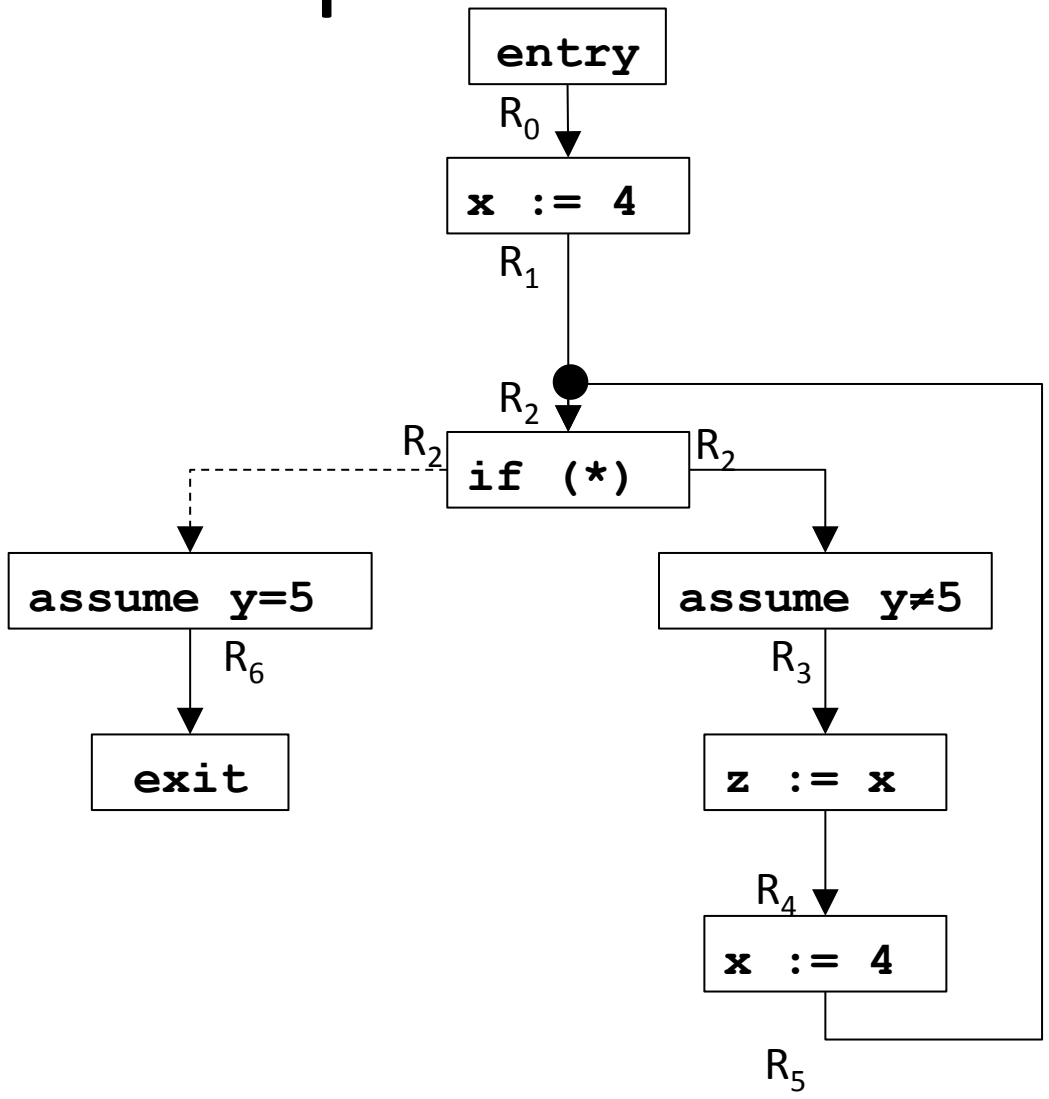
Write down variables

```
x := 4;
while (y≠5) do
     z := x;
     x := 4
```



Write down equations

```
x := 4;
while (y≠5) do
     z := x;
     x := 4
```



Collecting semantics equations

R_0 = **State**

$R_1 = [\![\texttt{x:=4}]\!]\, R_0$

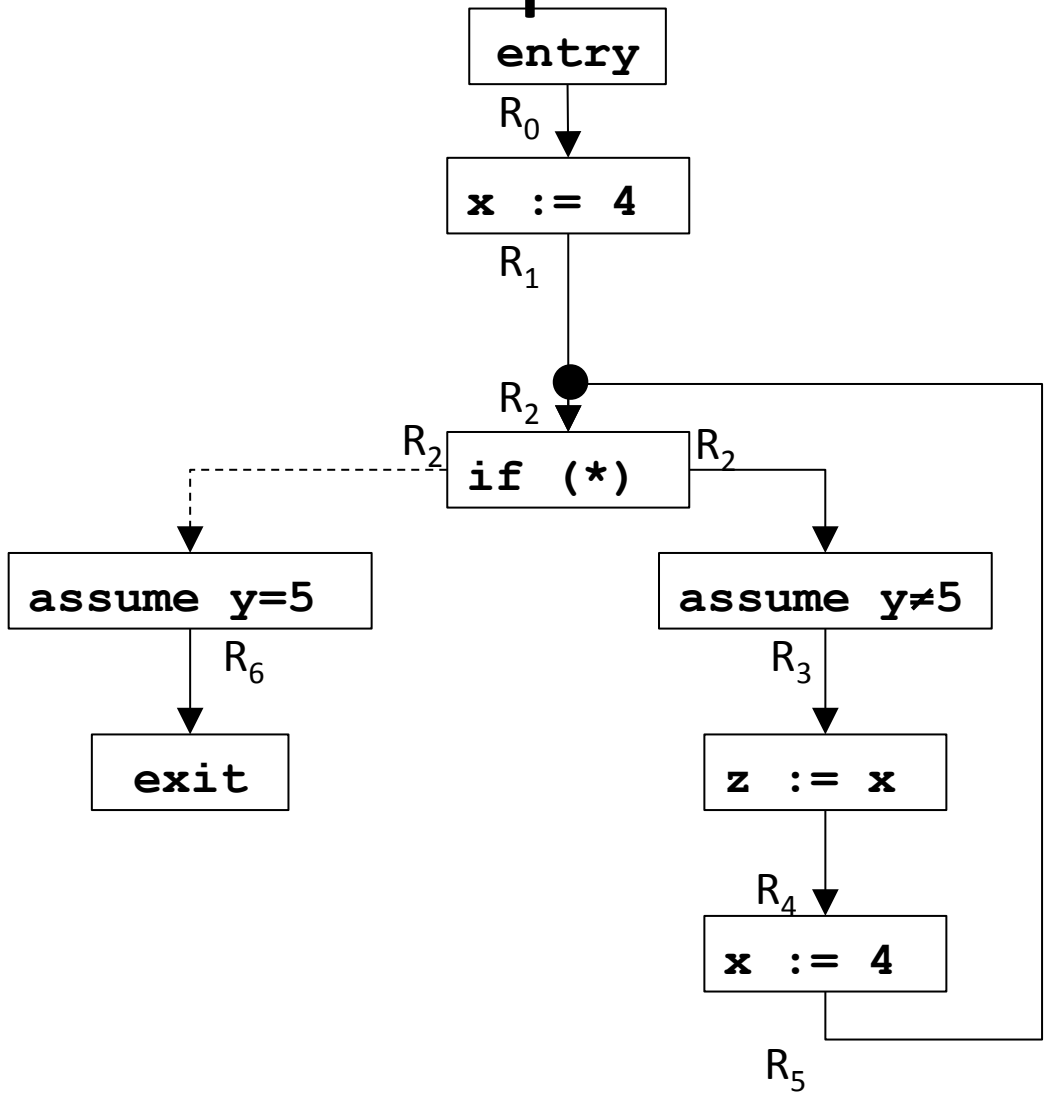
$R_2 = R_1 \cup R_5$

$R_3 = [\![\texttt{assume y≠5}]\!]\, R_2$

$R_4 = [\![\texttt{z:=x}]\!]\, R_3$

$R_5 = [\![\texttt{x:=4}]\!]\, R_4$

$R_6 = [\![\texttt{assume y=5}]\!]\, R_2$



Constant propagation equations

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^{\#} R_0$

$R_2 = R_1 \sqcup R_5$

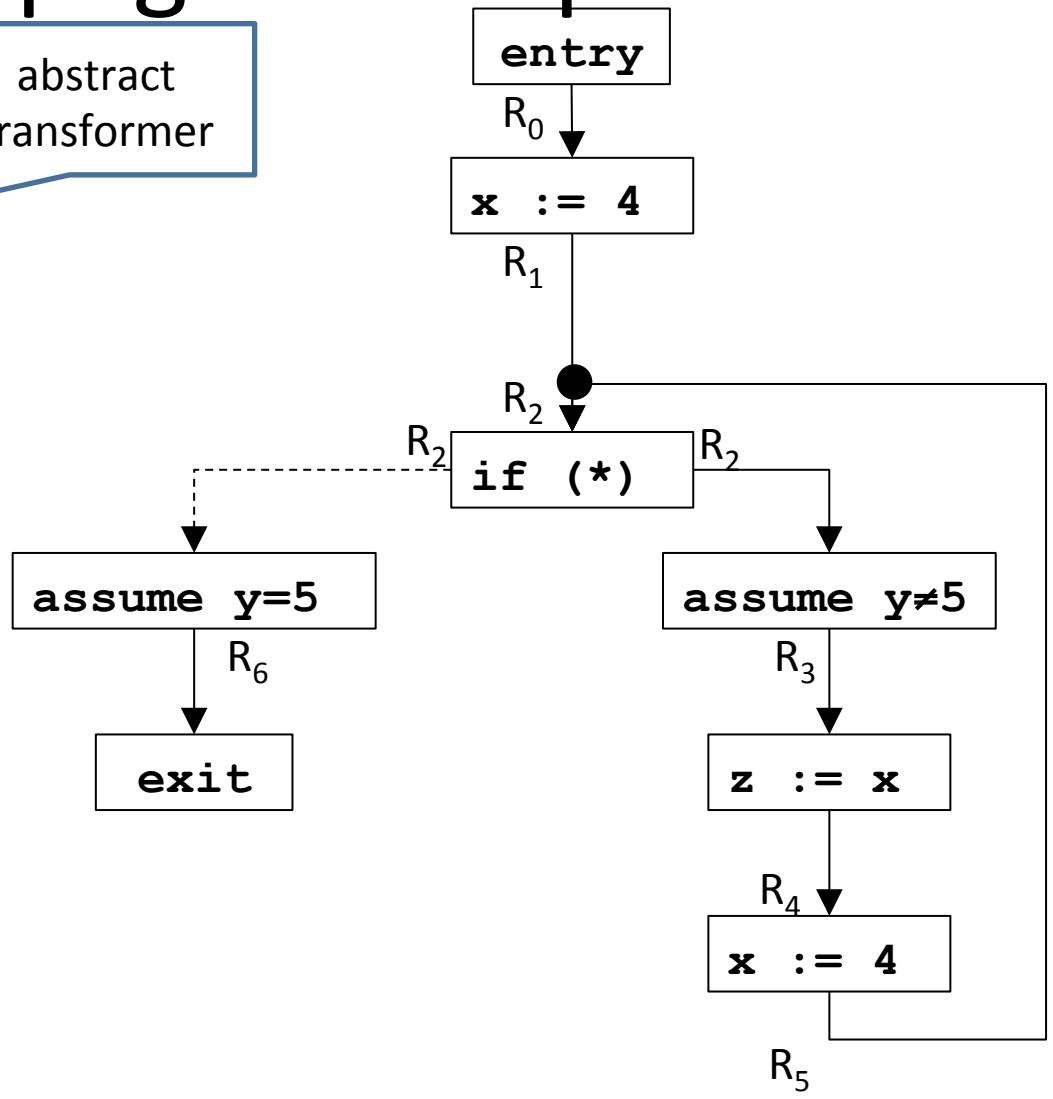
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

$R_4 = [\![\texttt{z:=x}]\!]^{\#} R_3$

$R_5 = [\![\texttt{x:=4}]\!]^{\#} R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^{\#} R_2$

abstract transformer



Abstract operations for CP

$R_0 = \top$

$R_1 = [\![\mathbf{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

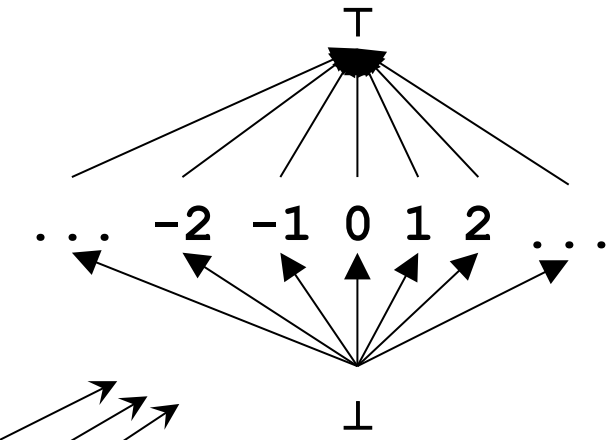
$R_3 = [\![\mathbf{assume\ y\neq 5}]\!]^\# R_2$

$R_4 = [\![\mathbf{z:=x}]\!]^\# R_3$

$R_5 = [\![\mathbf{x:=4}]\!]^\# R_4$

$R_6 = [\![\mathbf{assume\ y=5}]\!]^\# R_2$

CP lattice for a single variable



Lattice elements have the form: (v_x, v_y, v_z)

$[\![\mathbf{x:=4}]\!]^\# (v_x,v_y,v_z) = (4, v_y, v_z)$

$[\![\mathbf{z:=x}]\!]^\# (v_x,v_y,v_z) = (v_x, v_y, v_x)$

$[\![\mathbf{assume\ y\neq 5}]\!]^\# (v_x,v_y,v_z) = (v_x, v_y, v_x)$

$[\![\mathbf{assume\ y=5}]\!]^\# (v_x,v_y,v_z) =$ if $v_y = k \neq 5$ then $(\bot, \bot, \bot)$ else $(v_x, 5, v_z)$

$R_1 \sqcup R_5 = (a_1, b_1, c_1) \sqcup (a_5, b_5, c_5) = (a_1 \sqcup a_5, b_1 \sqcup b_5, c_1 \sqcup c_5)$

Chaotic iteration for CP: initialization

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^{\#} R_0$

$R_2 = R_1 \sqcup R_5$

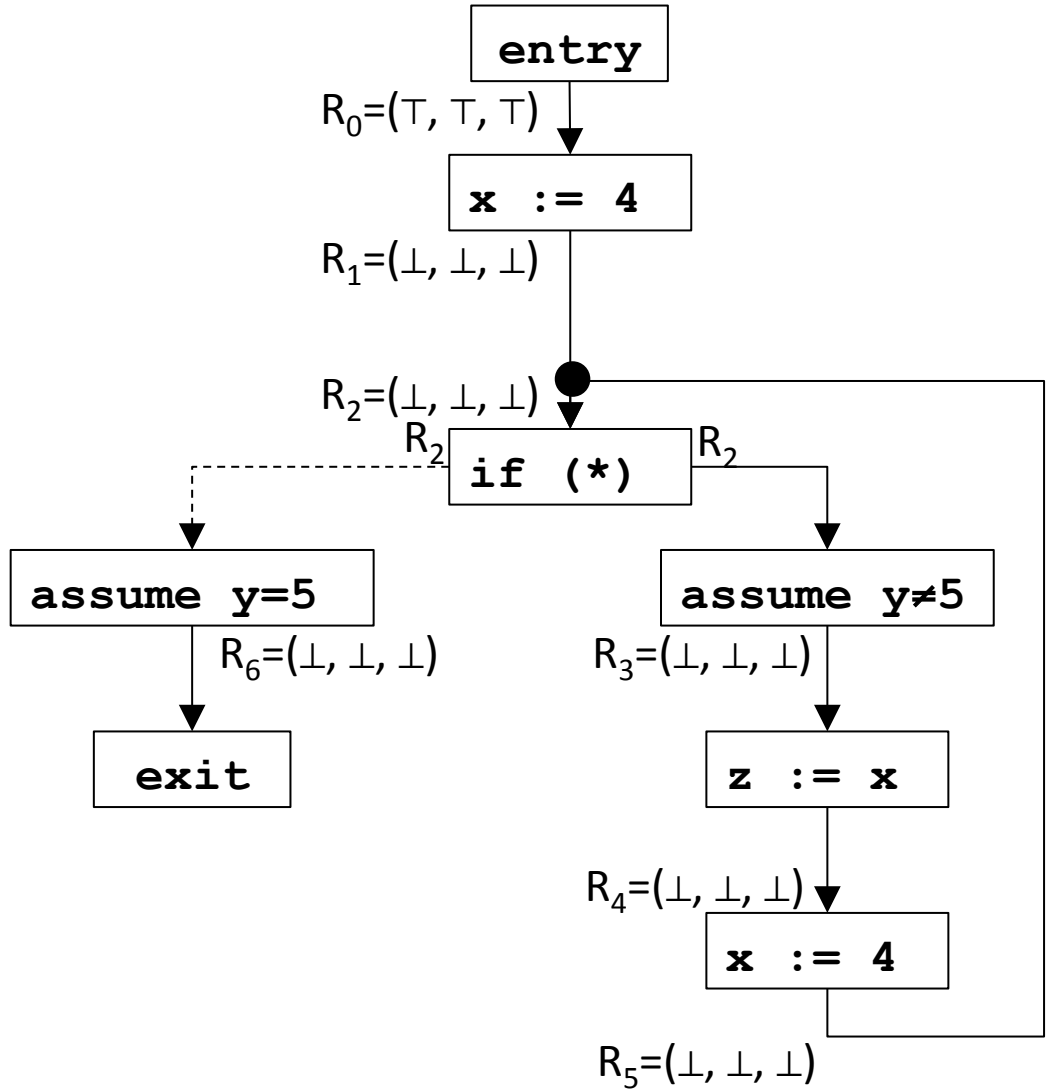
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

$R_4 = [\![\texttt{z:=x}]\!]^{\#} R_3$

$R_5 = [\![\texttt{x:=4}]\!]^{\#} R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^{\#} R_2$

WL = {R_0, R_1, R_2, R_3, R_4, R_5, R_6}



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^{\#} R_0$

$R_2 = R_1 \sqcup R_5$

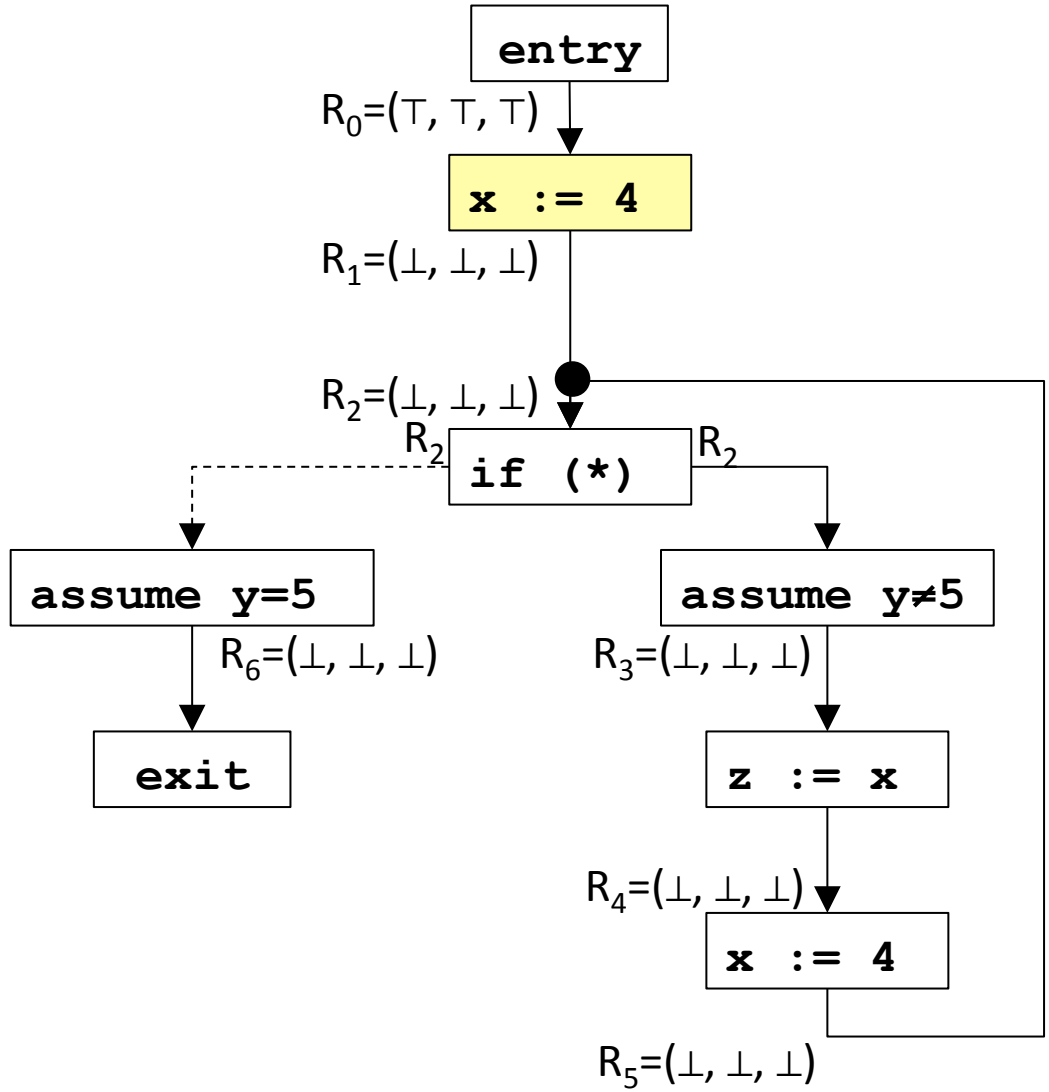
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

$R_4 = [\![\texttt{z:=x}]\!]^{\#} R_3$

$R_5 = [\![\texttt{x:=4}]\!]^{\#} R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^{\#} R_2$

WL = $\{R_1, R_2, R_3, R_4, R_5, R_6\}$



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

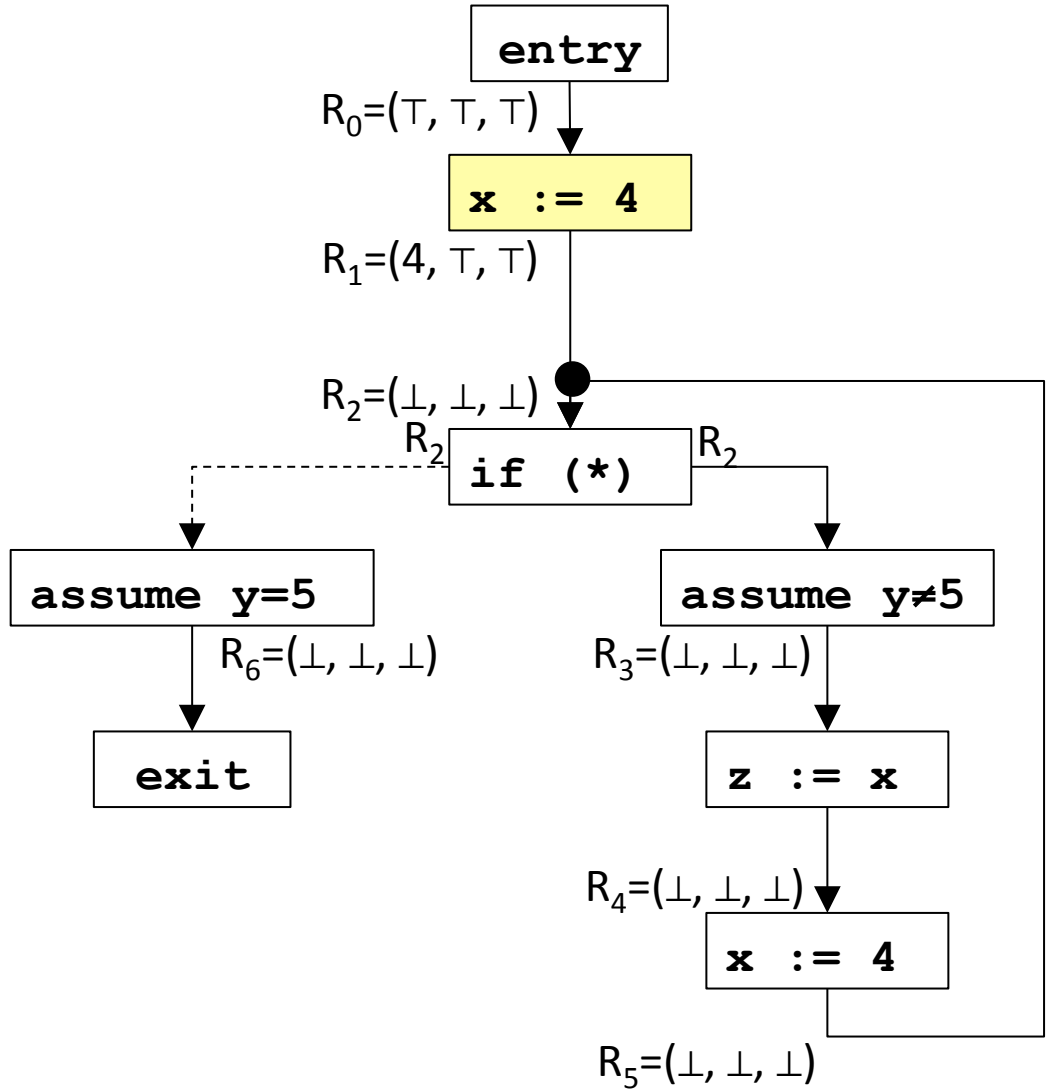
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = $\{R_2, R_3, R_4, R_5, R_6\}$



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^{\#} R_0$

$R_2 = R_1 \sqcup R_5$

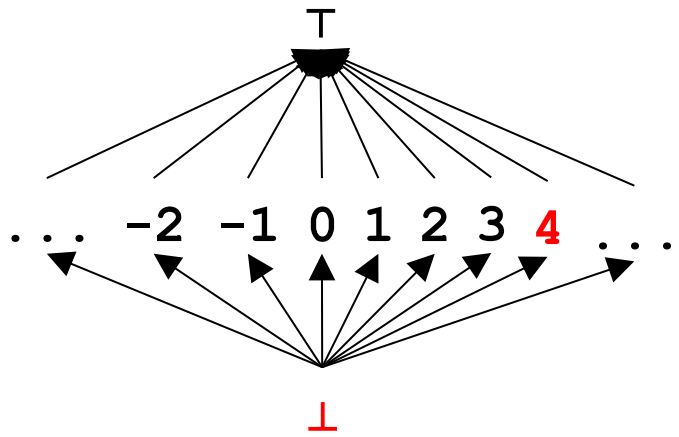
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

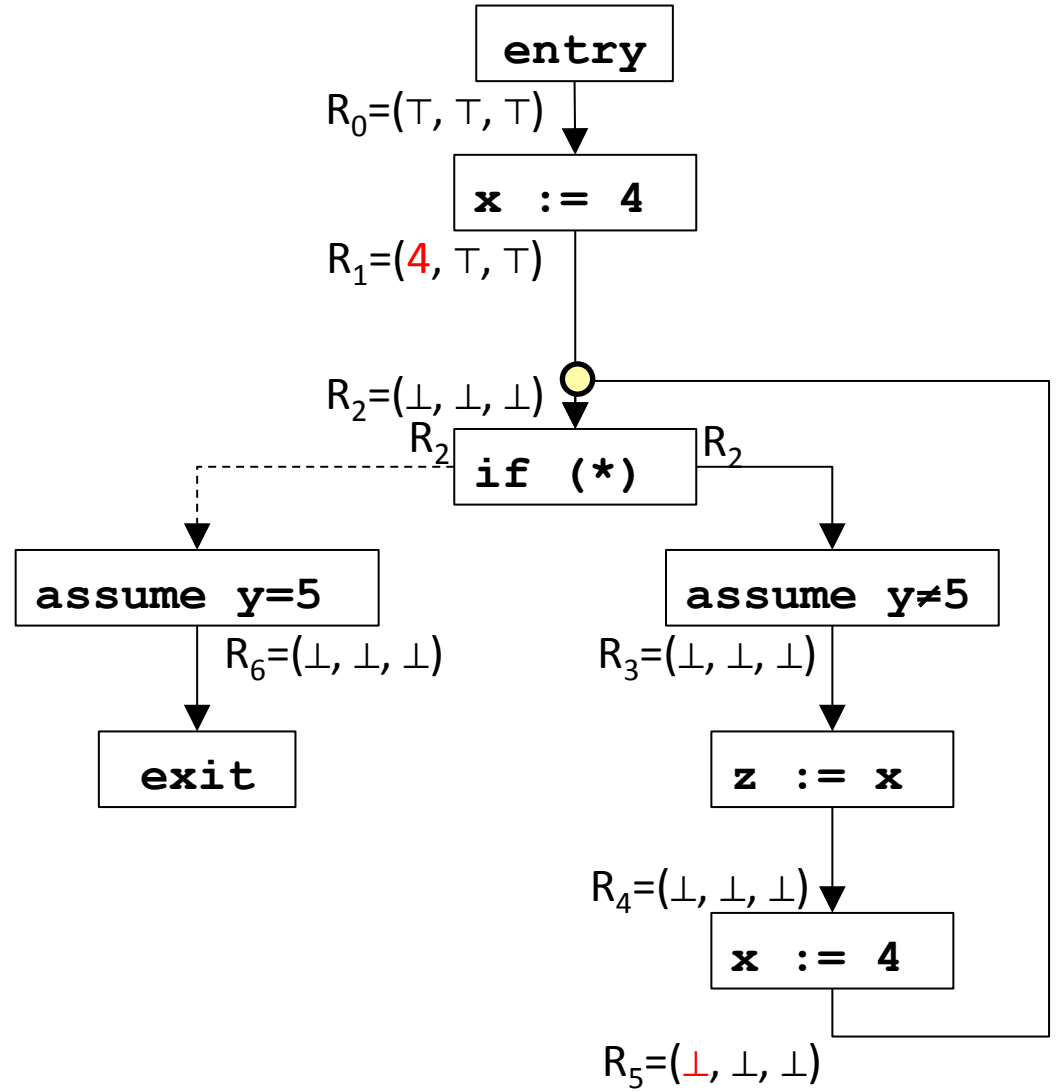
$R_4 = [\![\texttt{z:=x}]\!]^{\#} R_3$

$R_5 = [\![\texttt{x:=4}]\!]^{\#} R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^{\#} R_2$

WL = {R_2, R_3, R_4, R_5, R_6}





Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^{\#} R_0$

$R_2 = R_1 \sqcup R_5$

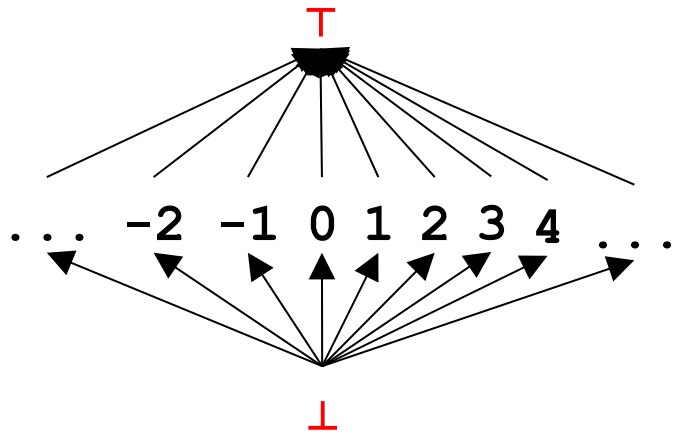
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

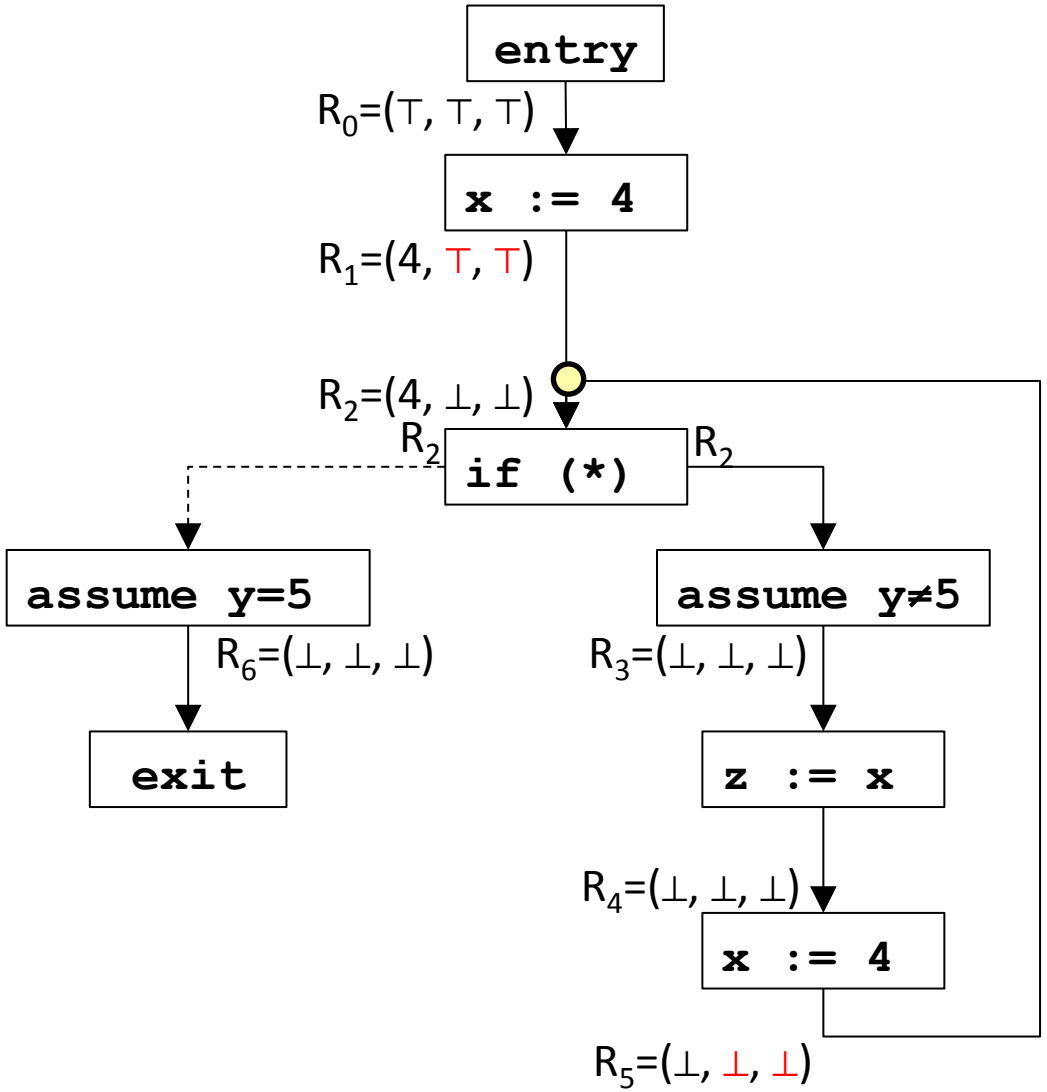
$R_4 = [\![\texttt{z:=x}]\!]^{\#} R_3$

$R_5 = [\![\texttt{x:=4}]\!]^{\#} R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^{\#} R_2$

WL = $\{R_2, R_3, R_4, R_5, R_6\}$





Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

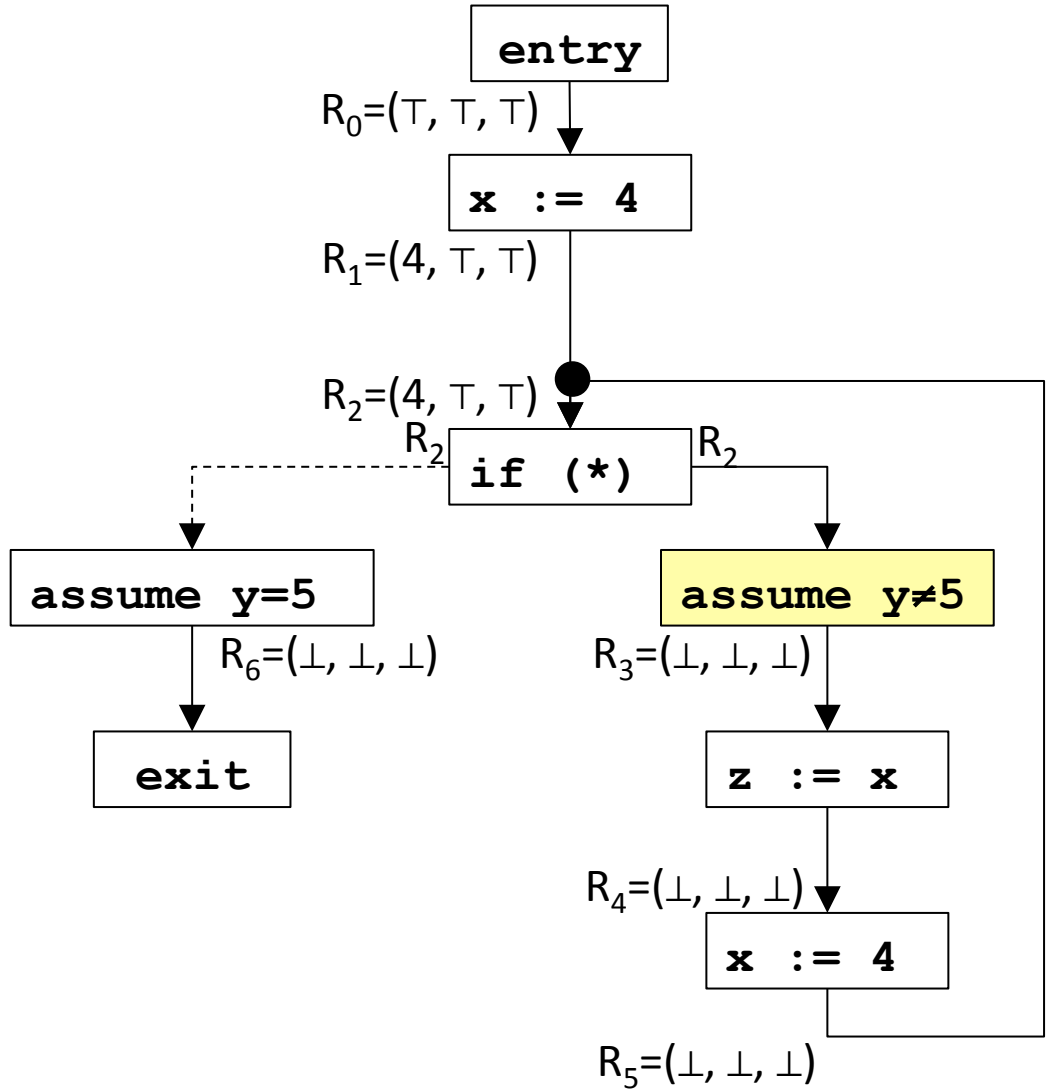
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = {R_3, R_4, R_5, R_6}



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

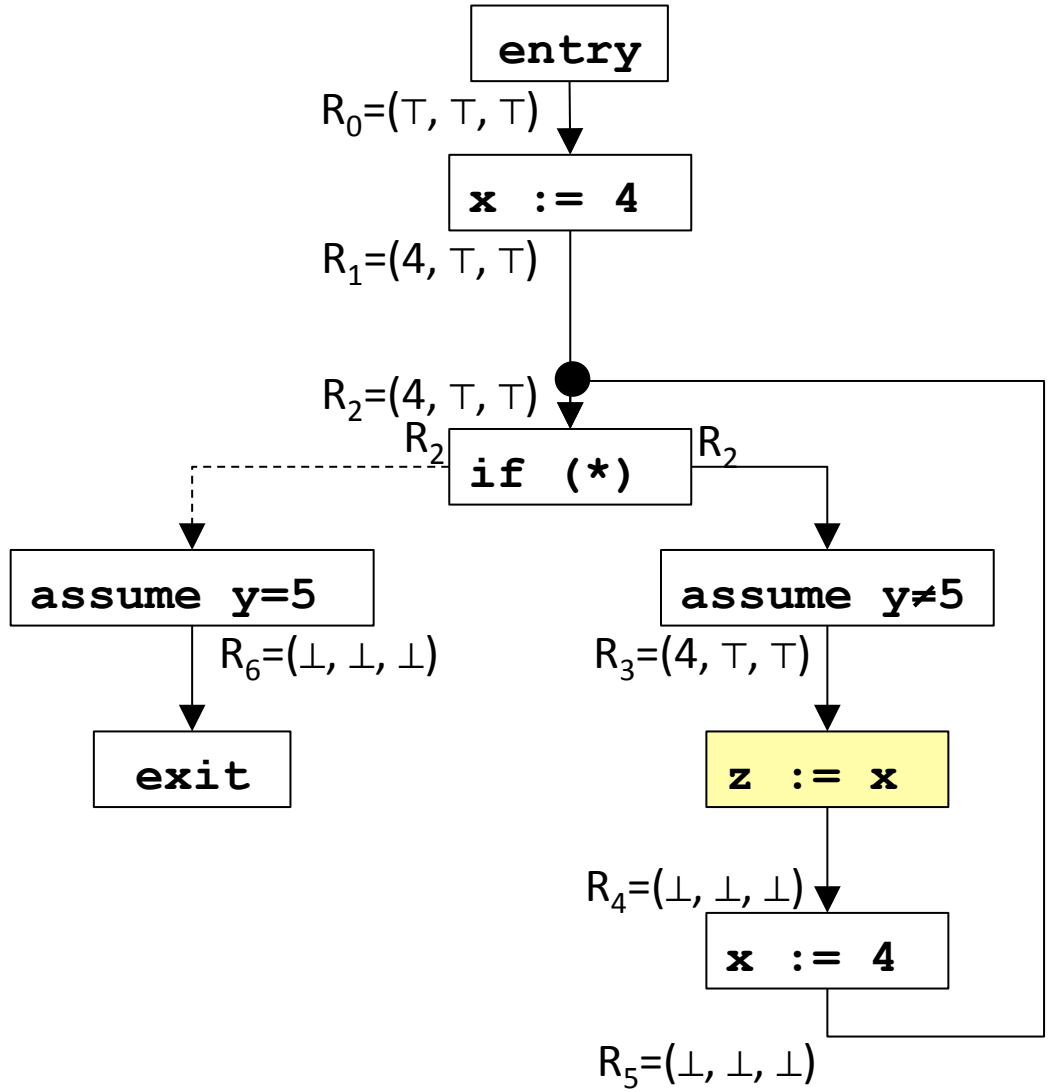
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = $\{R_4, R_5, R_6\}$



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

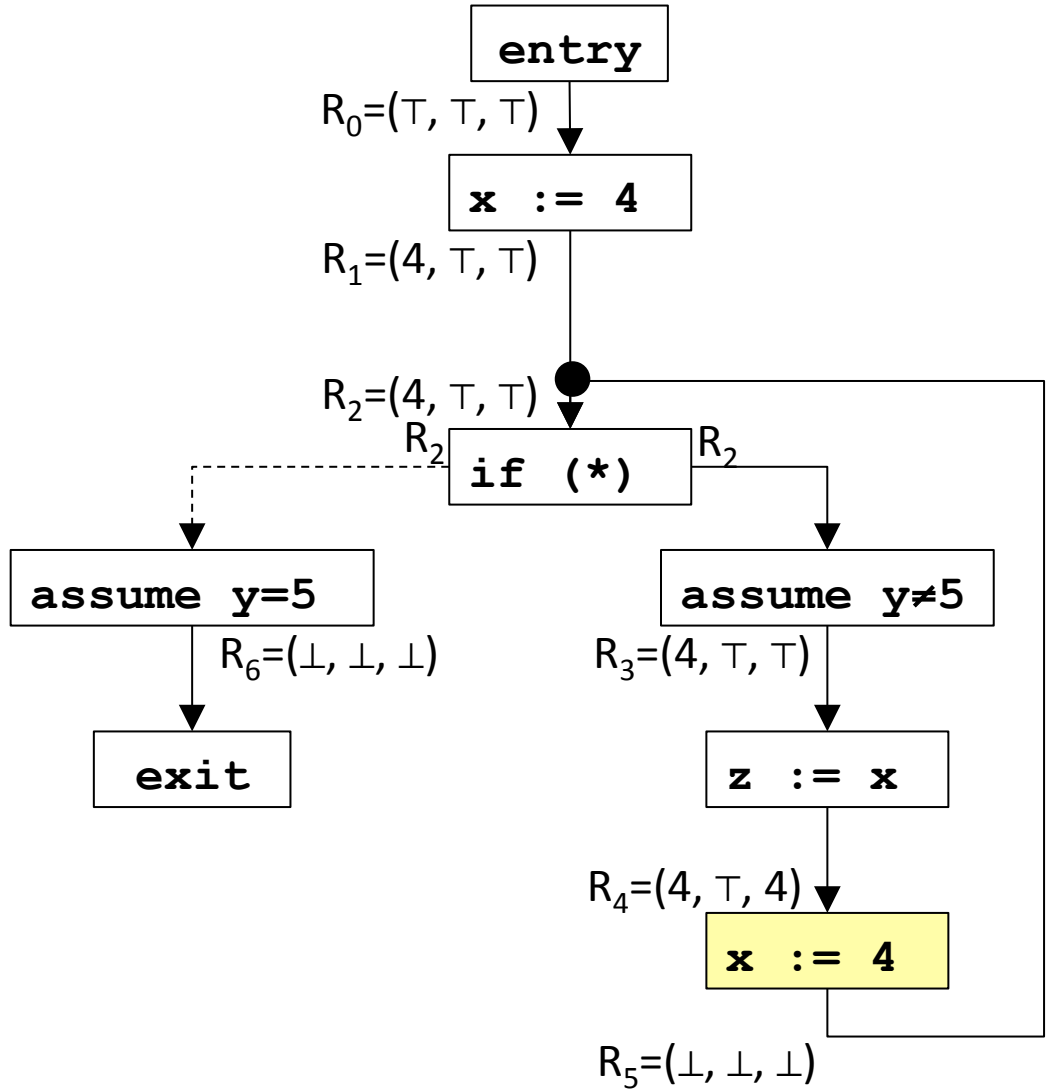
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = $\{R_5, R_6\}$



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

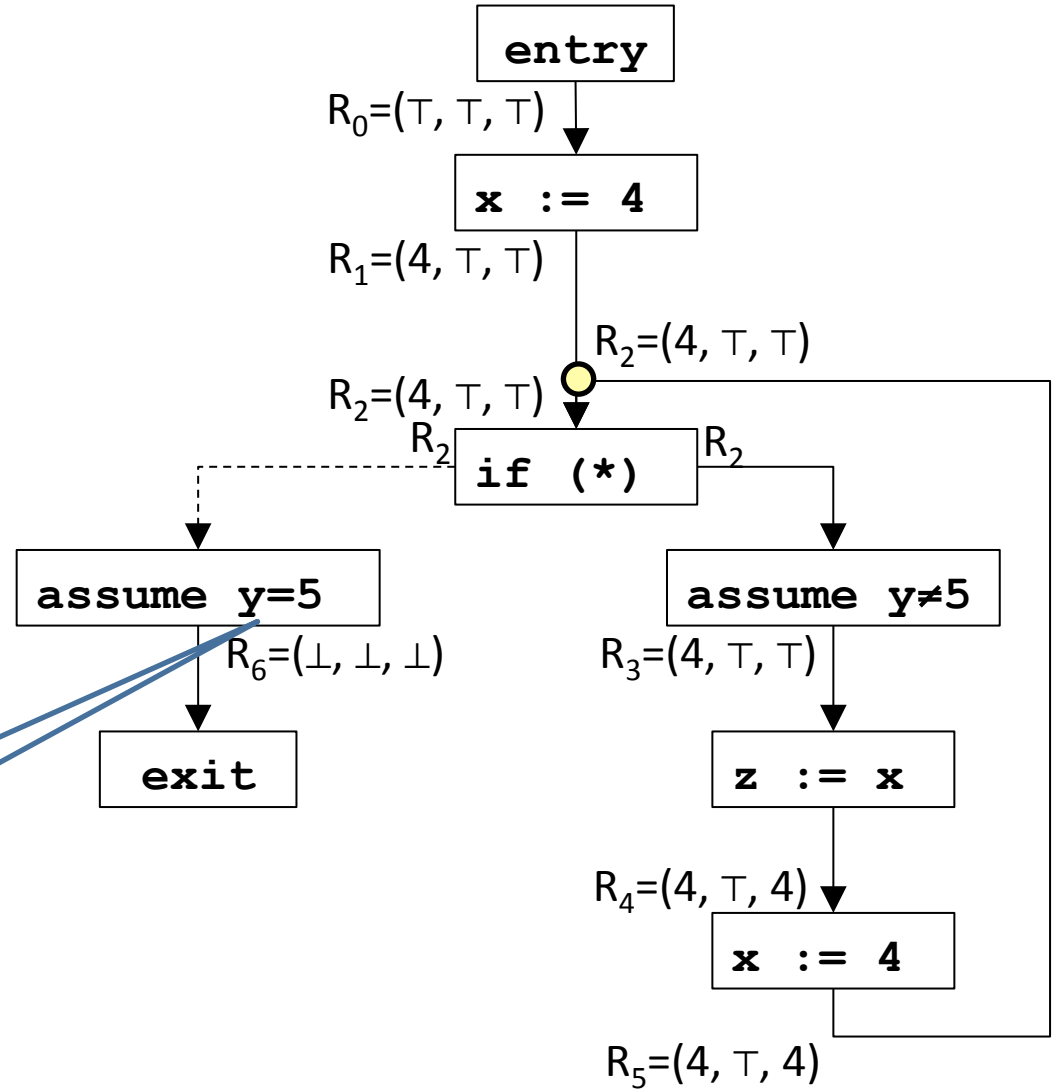
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = {R_2, R_6}



added R_2 back to worklist since it depends on R_5

Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

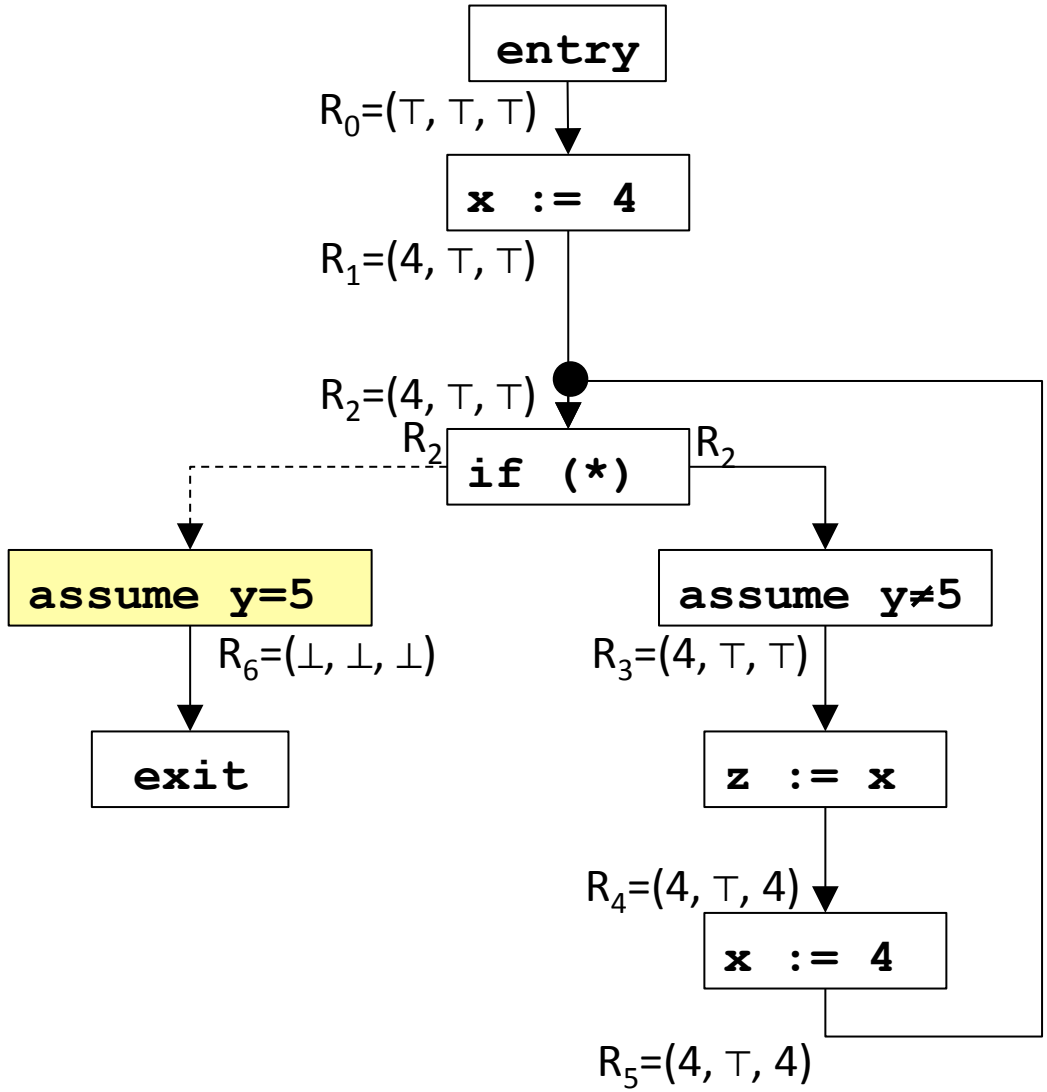
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = $\{R_6\}$



Chaotic iteration for CP

$R_0 = \top$

$R_1 = [\![\texttt{x:=4}]\!]^\# R_0$

$R_2 = R_1 \sqcup R_5$

$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

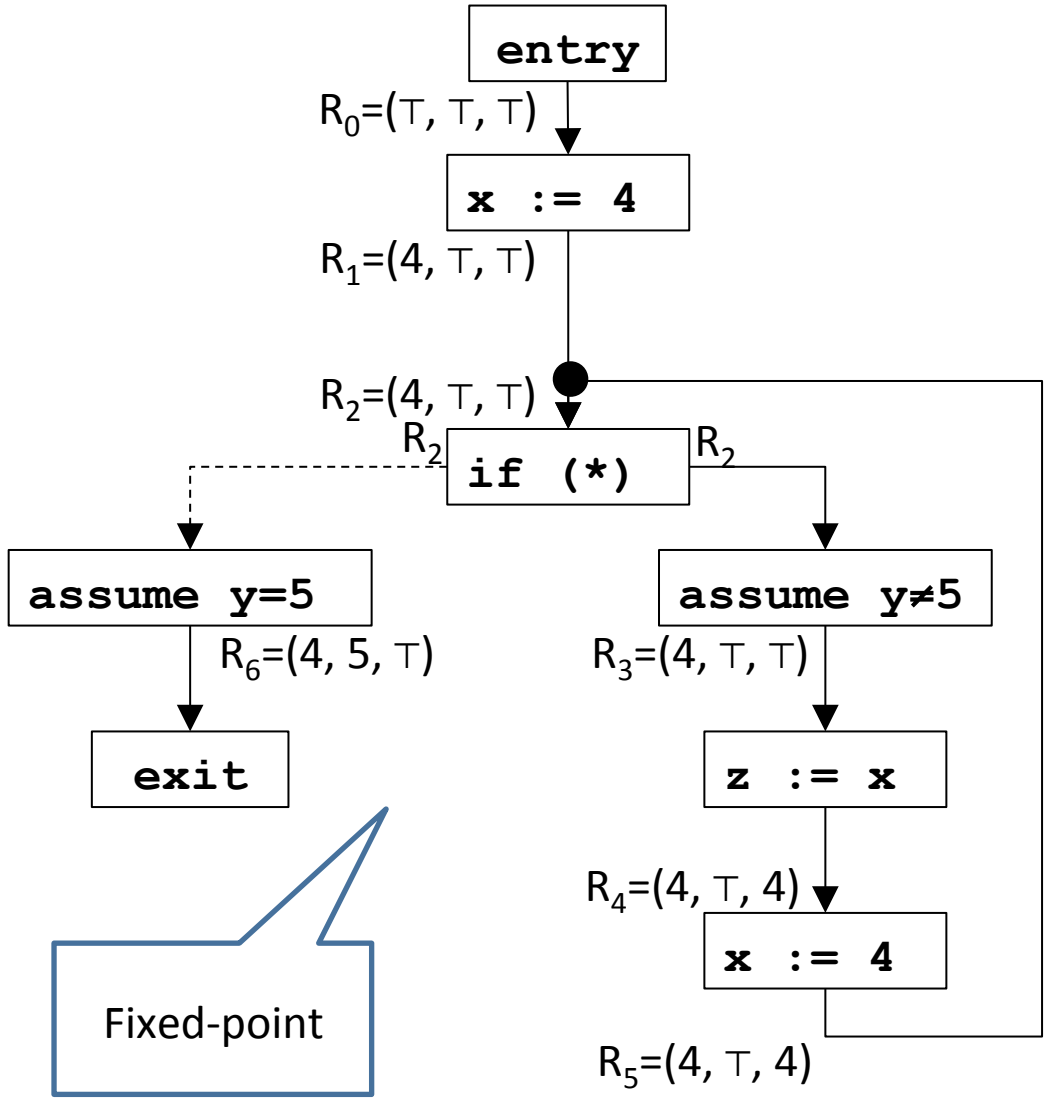
$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

$R_5 = [\![\texttt{x:=4}]\!]^\# R_4$

$R_6 = [\![\texttt{assume y=5}]\!]^\# R_2$

WL = {}

In practice maintain a worklist of nodes



Fixed-point

Chaotic iteration for static analysis

- Specialize chaotic iteration for programs
- Create a CFG for program
- Choose a cpo of properties for the static analysis to infer: $L = (D, \sqsubseteq, \sqcup, \bot)$
- Define variables R[0,…,*n*] for input/output of each CFG node such that R[i]$\in D$
- For each node v let v_{out} be the variable at the output of that node:
  $v_{out} = F[v](\sqcup u \mid (u,v)$ is a CFG edge)
  - Make sure each F[v] is monotone
- Variable dependence determined by outgoing edges in CFG

Complexity of chaotic iteration

- Parameters:
  - n the number of CFG nodes
  - k is the maximum in-degree of edges
  - Height h of lattice L
  - c is the maximum cost of
    - Applying F_v
    - $\sqcup$
    - Checking fixed-point condition for lattice L
- Complexity: $O(n \cdot h \cdot c \cdot k)$
- Incremental (worklist) algorithm reduces the n factor in factor
  - Implement worklist by priority queue and order nodes by reversed topological order

Required knowledge

- ✓ Collecting semantics
- ✓ Abstract semantics (over lattices)
- ✓ Algorithm to compute abstract semantics (chaotic iteration)
- • Connection between collecting semantics and abstract semantics
- • Abstract transformers

Recap

- We defined a reference semantics – the collecting semantics
- We defined an abstract semantics for a given lattice and abstract transformers
- We defined an algorithm to compute abstract least fixed-point when transformers are monotone and lattice obeys ACC

Recap

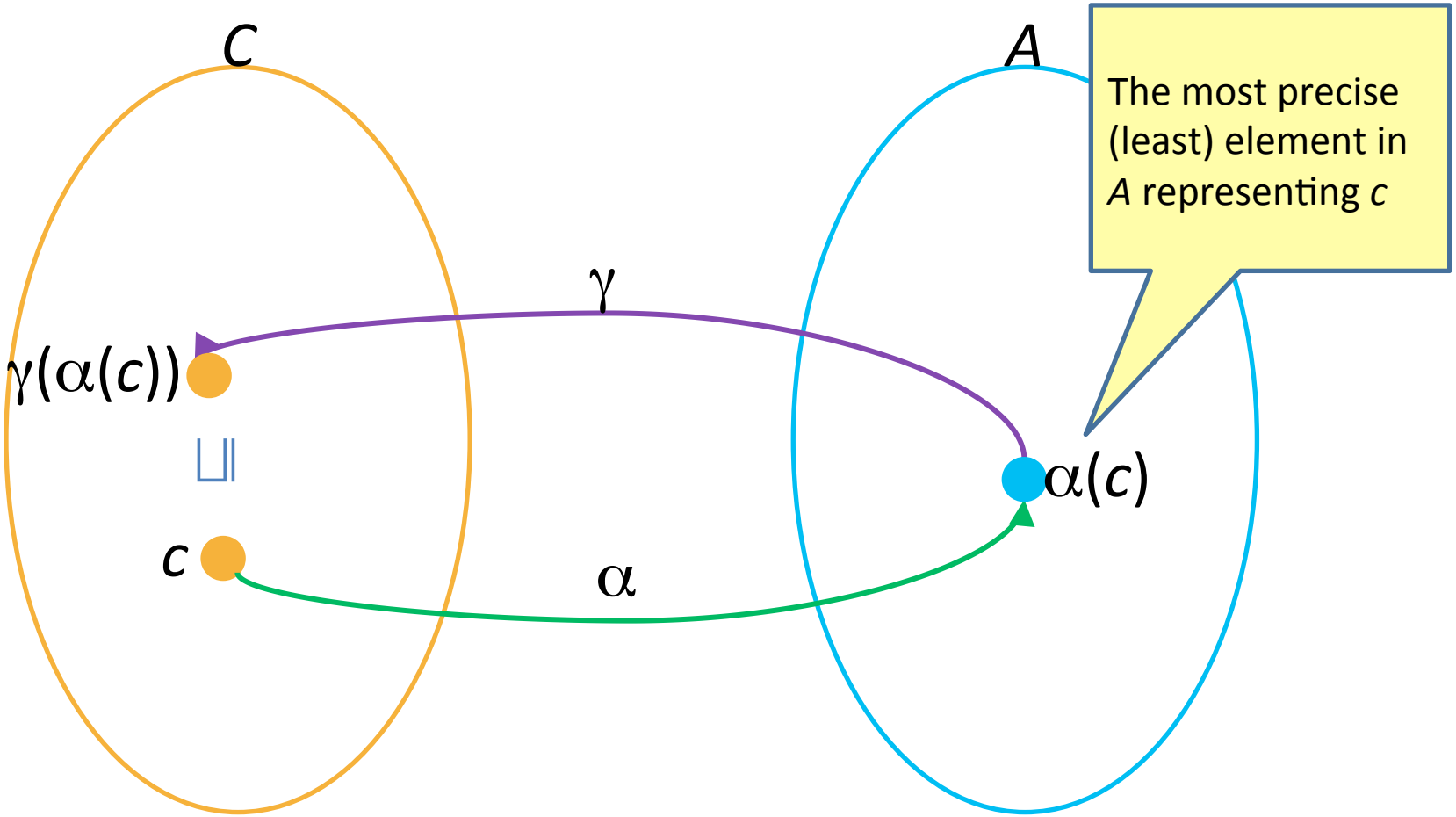
- We defined a reference semantics – the collecting semantics
- We defined an abstract semantics for a given lattice and abstract transformers
- We defined an algorithm to compute abstract least fixed-point when transformers are monotone and lattice obeys ACC
- Questions:

1. What is the connection between the two least fixed-points?
2. Transformer monotonicity is required for termination – what should we require for correctness?

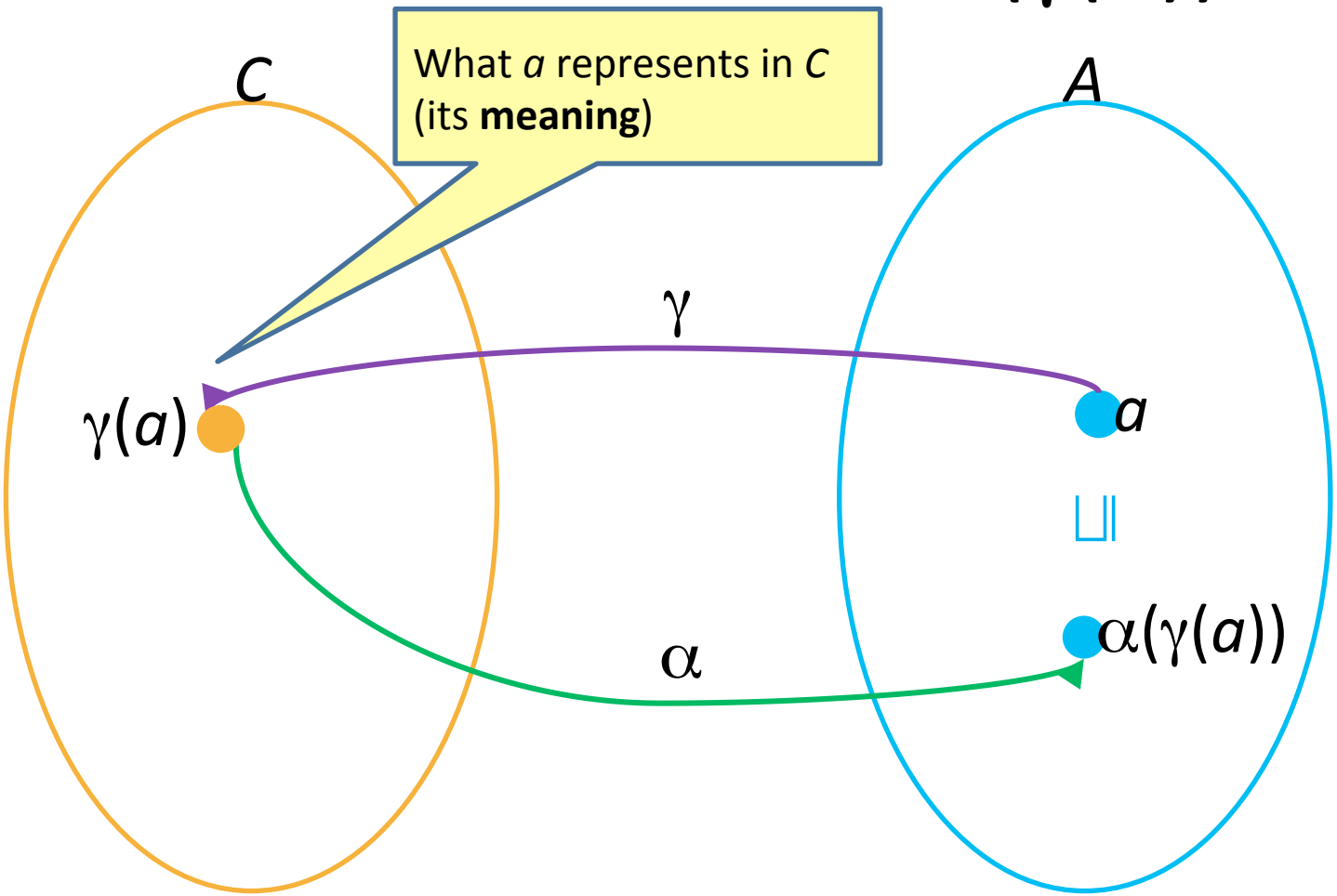
Galois Connection

- Given two complete lattices
  $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$ – concrete domain
  $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$ – abstract domain
- A Galois Connection (GC) is quadruple (C, α, γ, A) that relates C and A via the monotone functions
  - The abstraction function $\alpha : D^C \rightarrow D^A$
  - The concretization function $\gamma : D^A \rightarrow D^C$
- for every concrete element $c \in D^C$ and abstract element $a \in D^A$
  $\alpha(\gamma(a)) \sqsubseteq a$ and $c \sqsubseteq \gamma(\alpha(c))$
- Alternatively $\alpha(c) \sqsubseteq a$ iff $c \sqsubseteq \gamma(a)$

Galois Connection: $c \sqsubseteq \gamma(\alpha(c))$



Galois Connection: $\alpha(\gamma(a)) \sqsubseteq a$



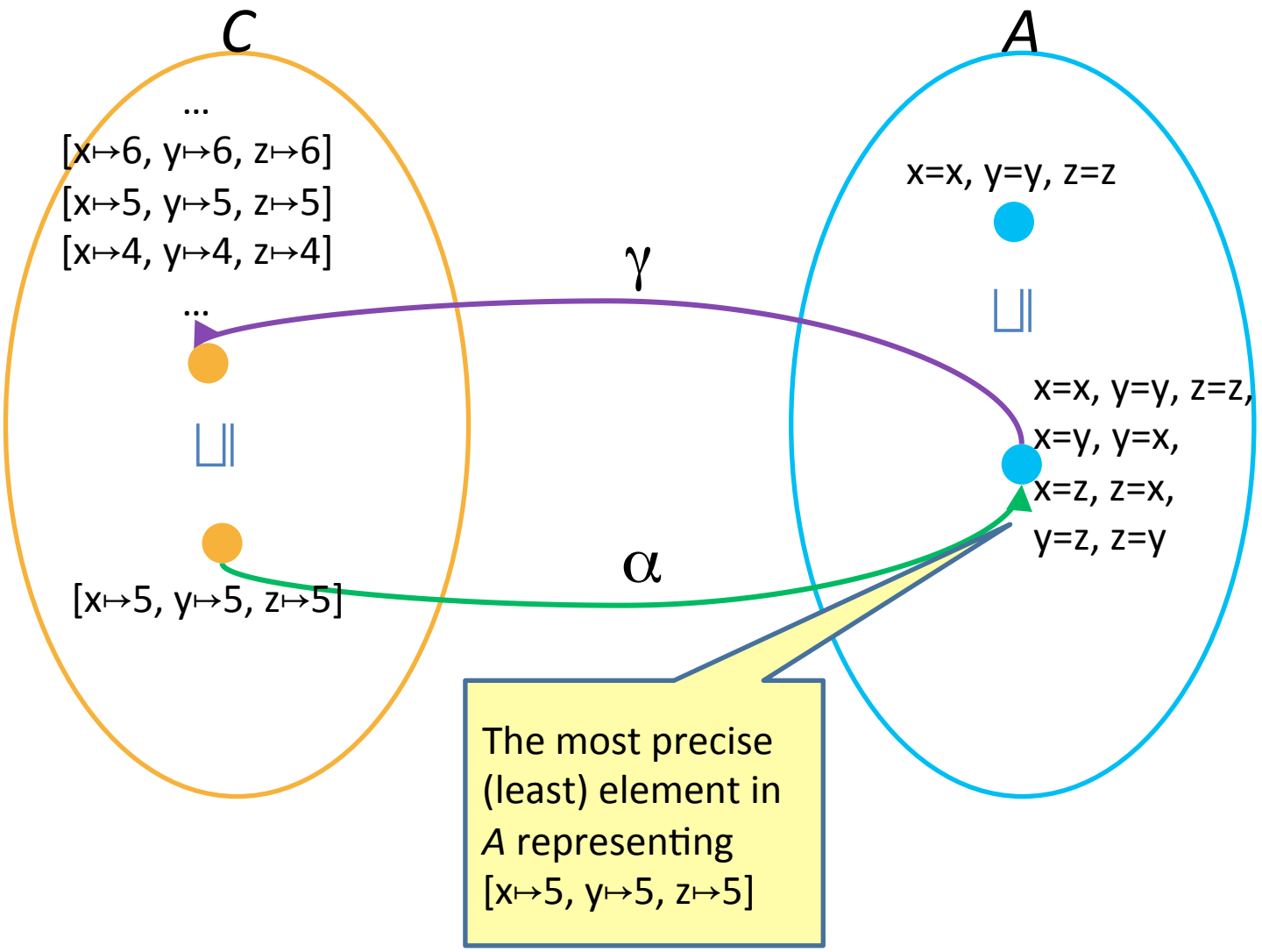
Example: lattice of equalities

- Concrete lattice:
  $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- Abstract lattice:
  $EQ = \{ x{=}y \mid x, y \in \text{Var}\}$
  $A = (2^{EQ}, \supseteq, \cap, \cup, EQ\ , \varnothing)$
  - Treat elements of *A* as both formulas and sets of constraints
- Useful for copy propagation – a compiler optimization
-
  $\alpha(X) = ?$
  $\gamma(Y) = ?$

Example: lattice of equalities

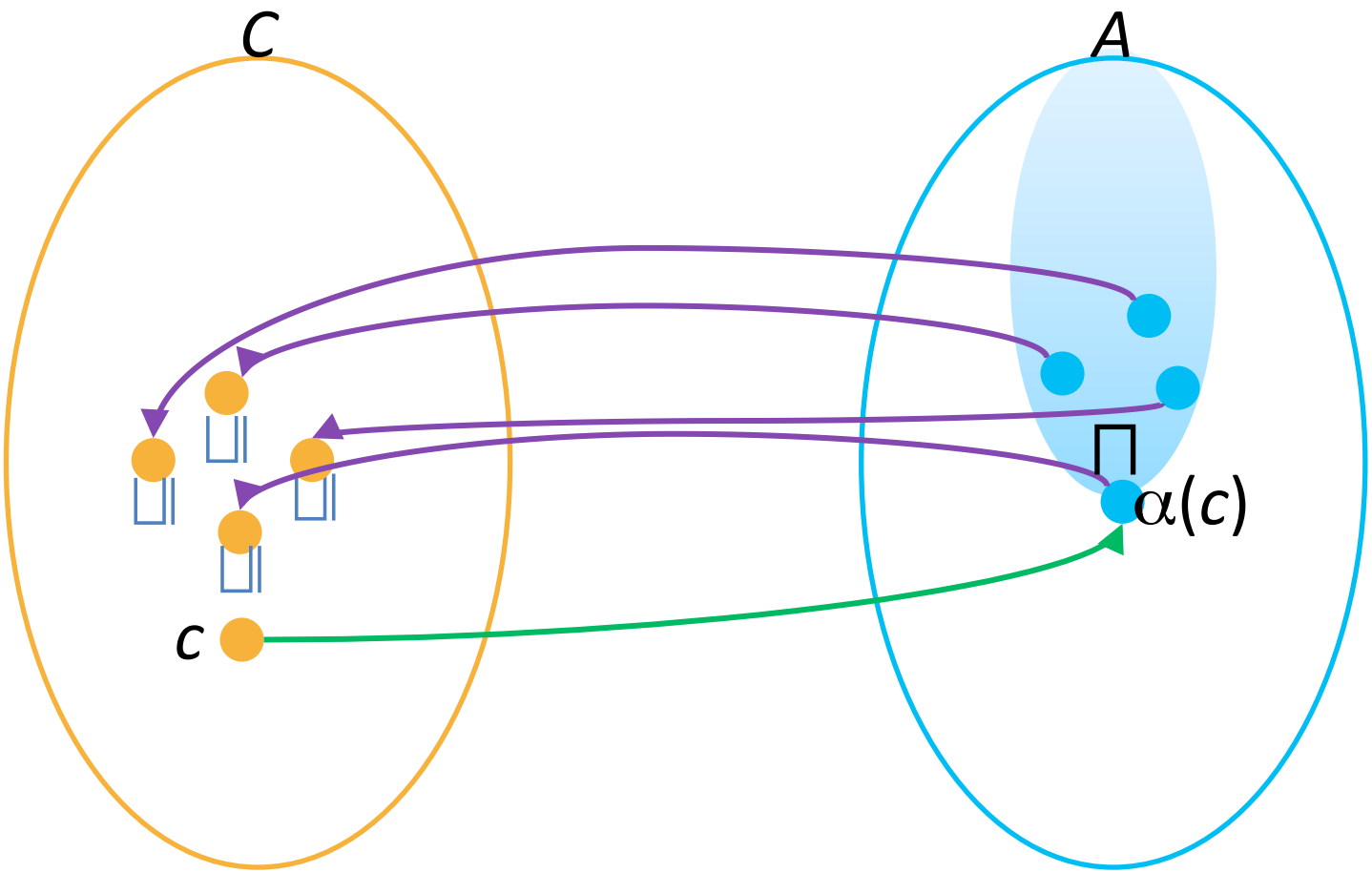
- Concrete lattice:
  $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- Abstract lattice:
  $EQ = \{ x{=}y \mid x, y \in \mathrm{Var}\}$
  $A = (2^{EQ}, \supseteq, \cap, \cup, EQ\ , \varnothing)$
  - Treat elements of A as both formulas and sets of constraints
- Useful for copy propagation – a compiler optimization
- $\beta(s) = \alpha(\{s\}) = \{ x{=}y \mid s\ x = s\ y\}$ that is $s \vDash x{=}y$
  $\alpha(X) = \cap\{\beta(s) \mid s{\in}X\} = \bigsqcup^{A} \{\beta(s) \mid s{\in}X\}$
  $\gamma(\mathrm{Y}) = \{ s \mid s \vDash \wedge\mathrm{Y} \} = \mathrm{models}(\wedge\mathrm{Y})$

Galois Connection: $c \sqsubseteq \gamma(\alpha(c))$



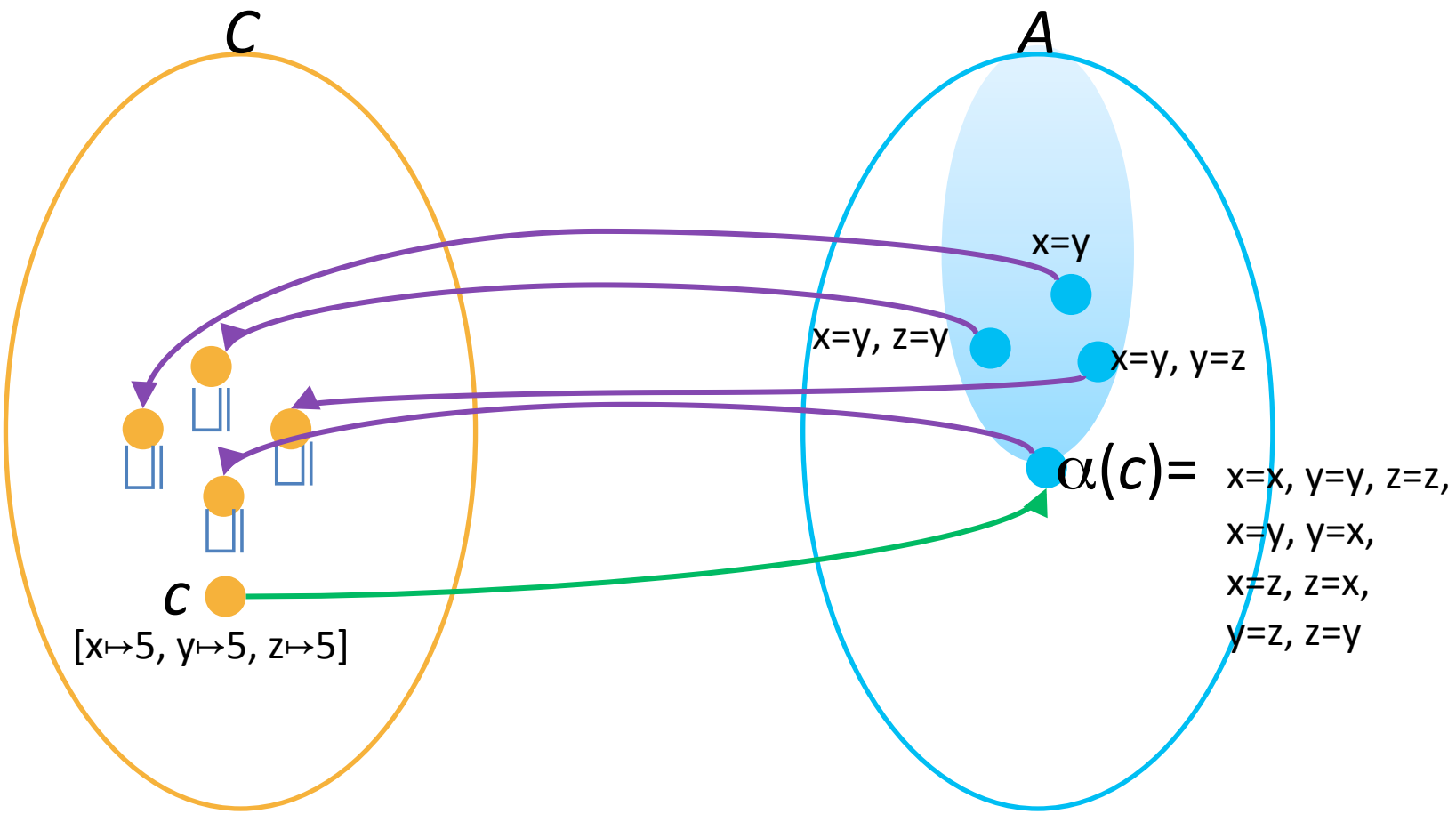
Most precise abstract representation

$$\alpha(c) = \sqcap\{c' \mid c \sqsubseteq \gamma(c')\}$$

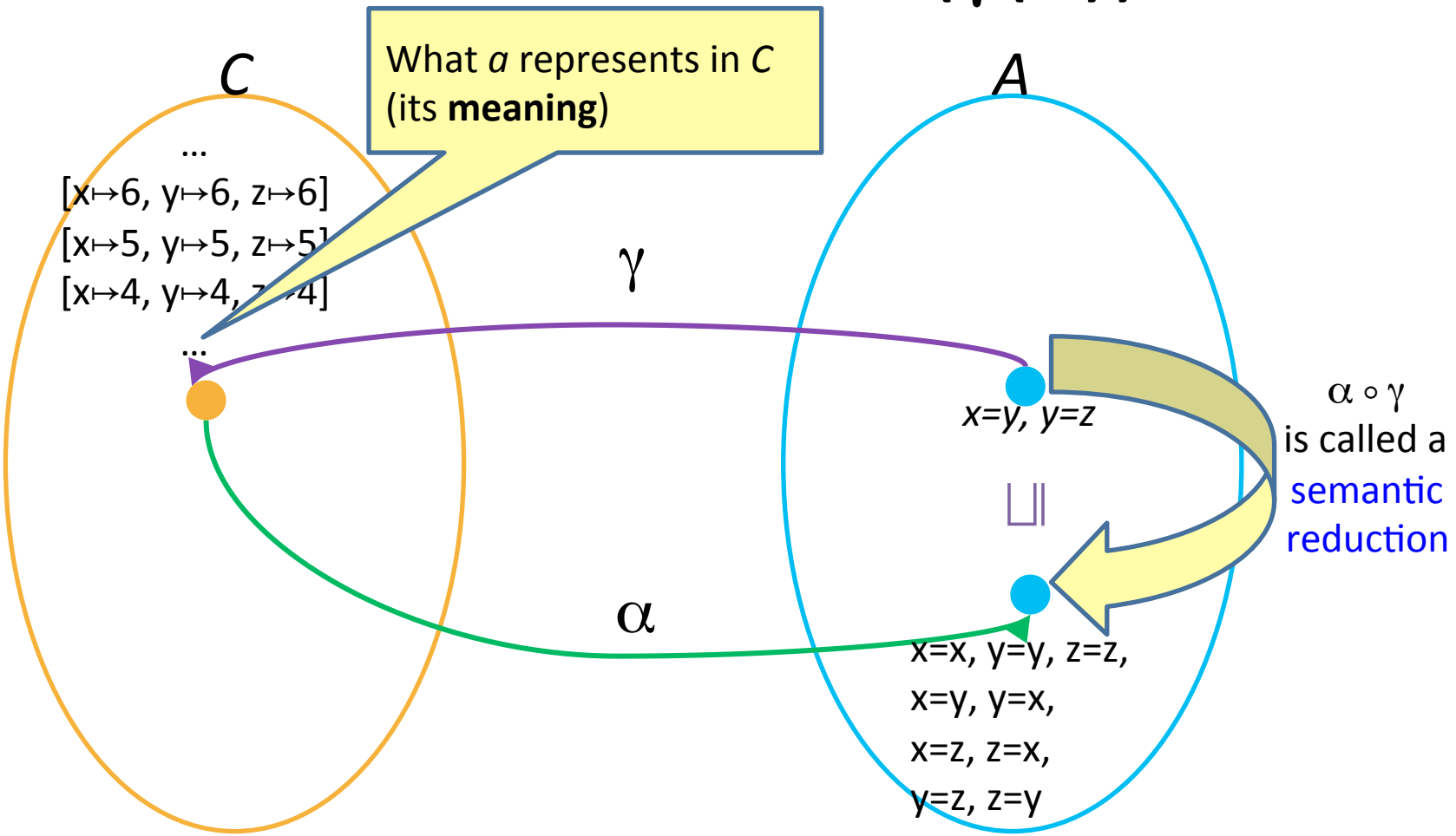


Most precise abstract representation

$$\alpha(c) = \sqcap\{c' \mid c \sqsubseteq \gamma(c')\}$$

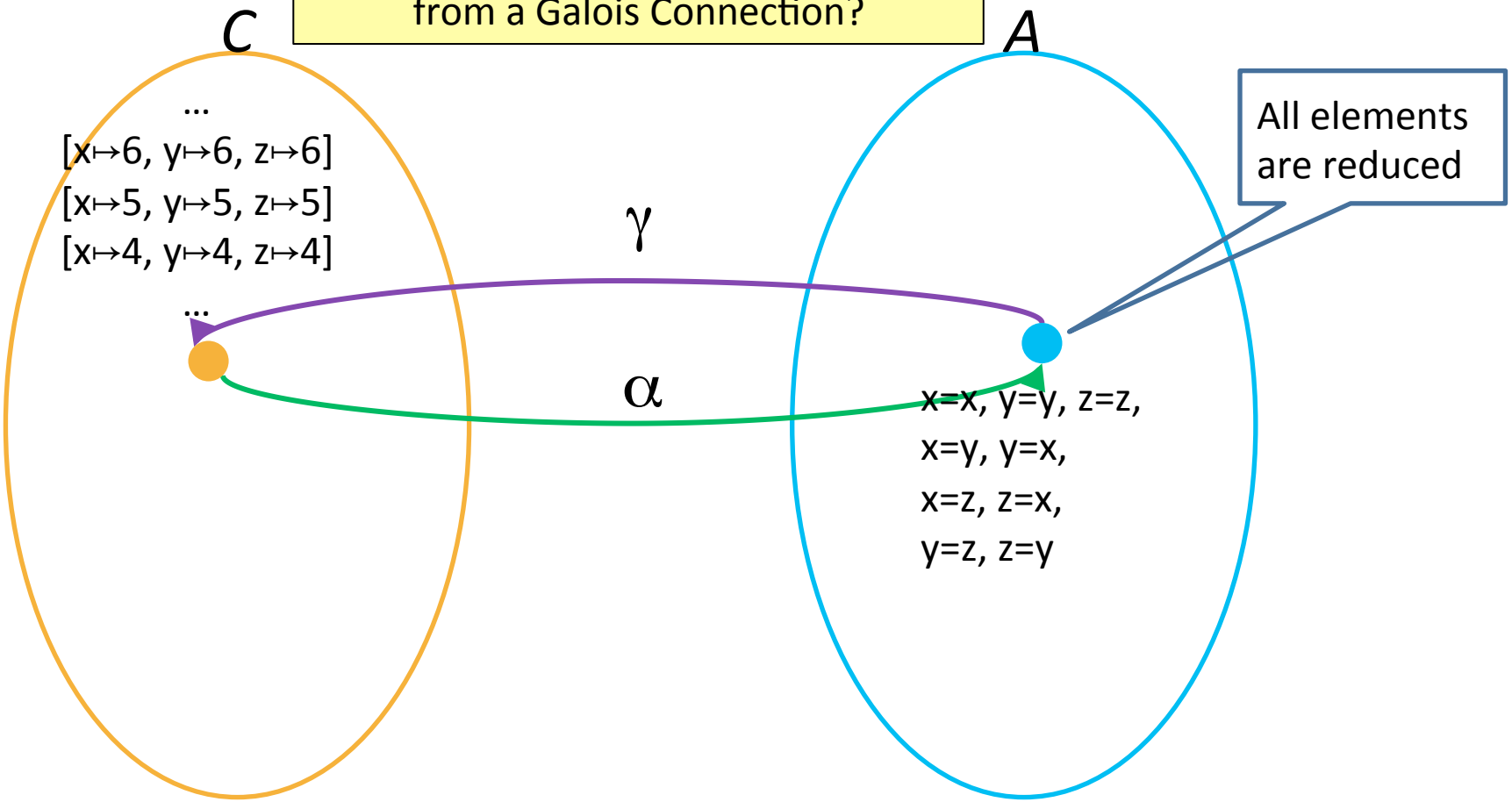


Galois Connection: $\alpha(\gamma(a)) \sqsubseteq a$



Galois Insertion $\forall a: \alpha(\gamma(a))=a$

How can we obtain a Galois Insertion from a Galois Connection?



Properties of a Galois Connection

- The abstraction and concretization functions uniquely determine each other:

$$\gamma(a) = \bigsqcup\{c \mid \alpha(c) \sqsubseteq a\}$$

$$\alpha(c) = \bigsqcap\{a \mid c \sqsubseteq \gamma(a)\}$$

Abstracting (disjunctive) sets

- It is usually convenient to first define the abstraction of single elements

$$\beta(s) = \alpha(\{s\})$$

- Then lift the abstraction to sets of elements

$$\alpha(X) = \bigsqcup^{A} \{\beta(s) \mid s \in X\}$$

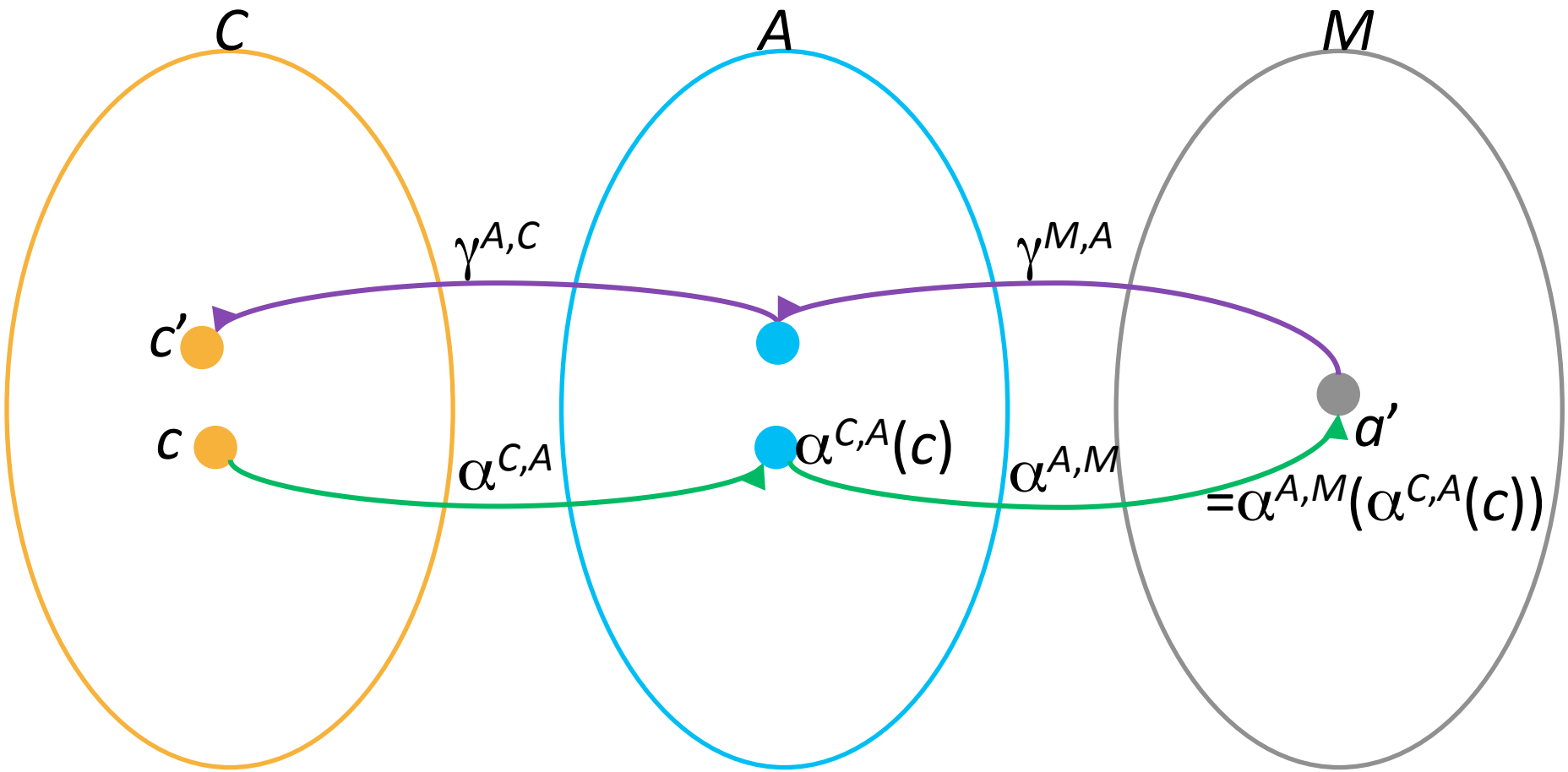
The case of symbolic domains

- An important class of abstract domains are symbolic domains – domains of formulas
- $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
  $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
- If D^A is a set of formulas then the abstraction of a state is defined as
  $$\beta(s) = \alpha(\{s\}) = \sqcap^A\{\varphi \mid s \vDash \varphi\}$$
  the least formula from D^A that s satisfies
- The abstraction of a set of states is
  $$\alpha(X) = \sqcup^A \{\beta(s) \mid s \in X\}$$
- The concretization is
  $$\gamma(\varphi) = \{ s \mid s \vDash \varphi \} = \text{models}(\varphi)$$

Inducing along the connections

- Assume the complete lattices
  $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
  $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
  $M = (D^M, \sqsubseteq^M, \sqcup^M, \sqcap^M, \bot^M, \top^M)$
  and
  Galois connections
  $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$ and $GC^{A,M}=(A, \alpha^{A,M}, \gamma^{M,A}, M)$
- **Lemma:** both connections induce the
  $GC^{C,M}= (C, \alpha^{C,M}, \gamma^{M,C}, M)$
  defined by $\alpha^{C,M} = \alpha^{C,A} \circ \alpha^{A,M}$ and $\gamma^{M,C} = \gamma^{M,A} \circ \gamma^{A,C}$

Inducing along the connections

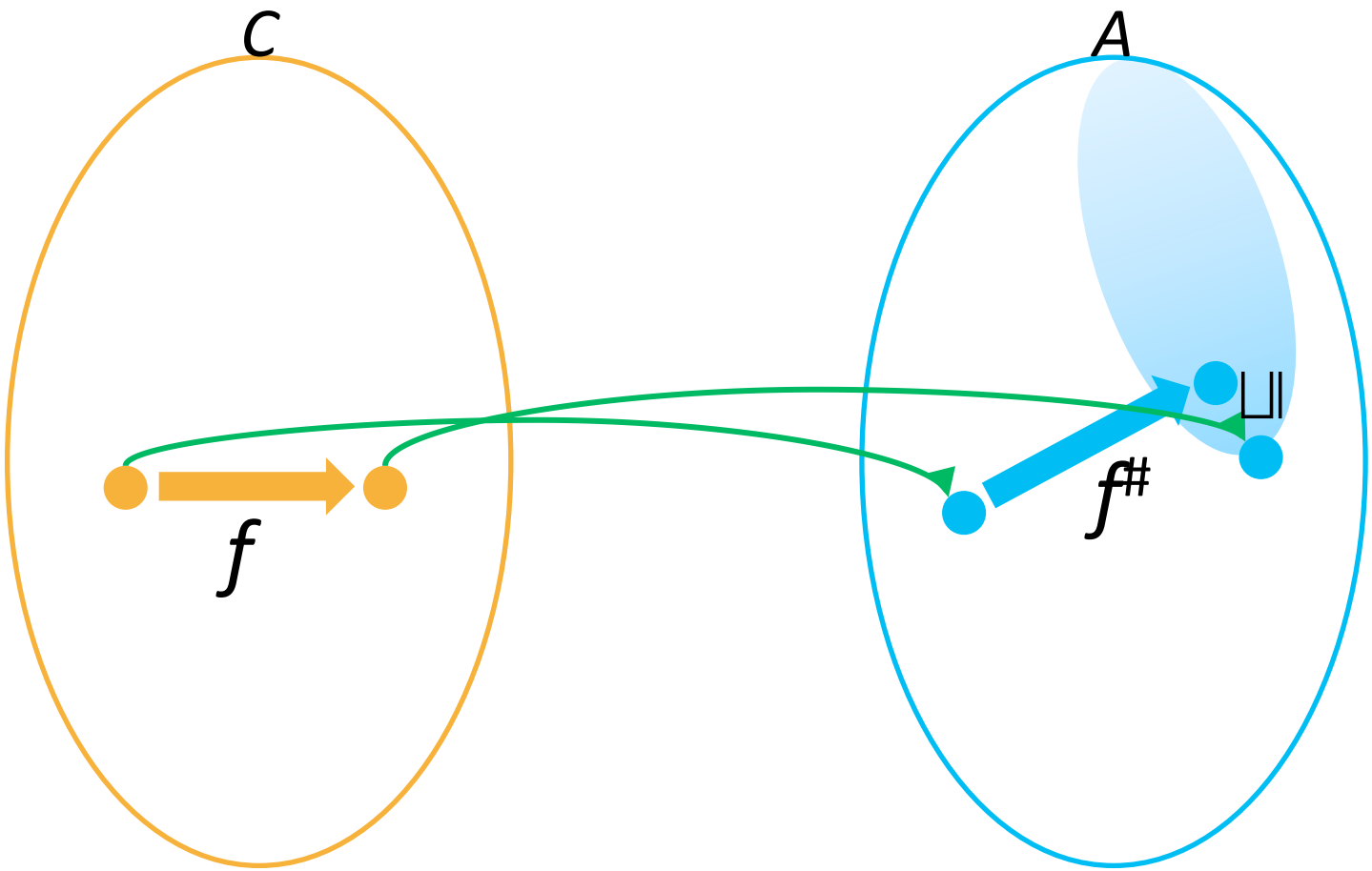


Sound abstract transformer

- Given two lattices
  $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
  $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
  and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
- A concrete transformer $f : D^C \rightarrow D^C$
  an abstract transformer $f^\# : D^A \rightarrow D^A$
- We say that $f^\#$ is a sound transformer (w.r.t. f) if
  - $\forall c: f(c)=c' \Rightarrow \alpha(f^\#(c)) \sqsupseteq \alpha(c')$
  - For every a and a' such that
    $\alpha(f(\gamma(a))) \sqsubseteq^A f^\#(a)$

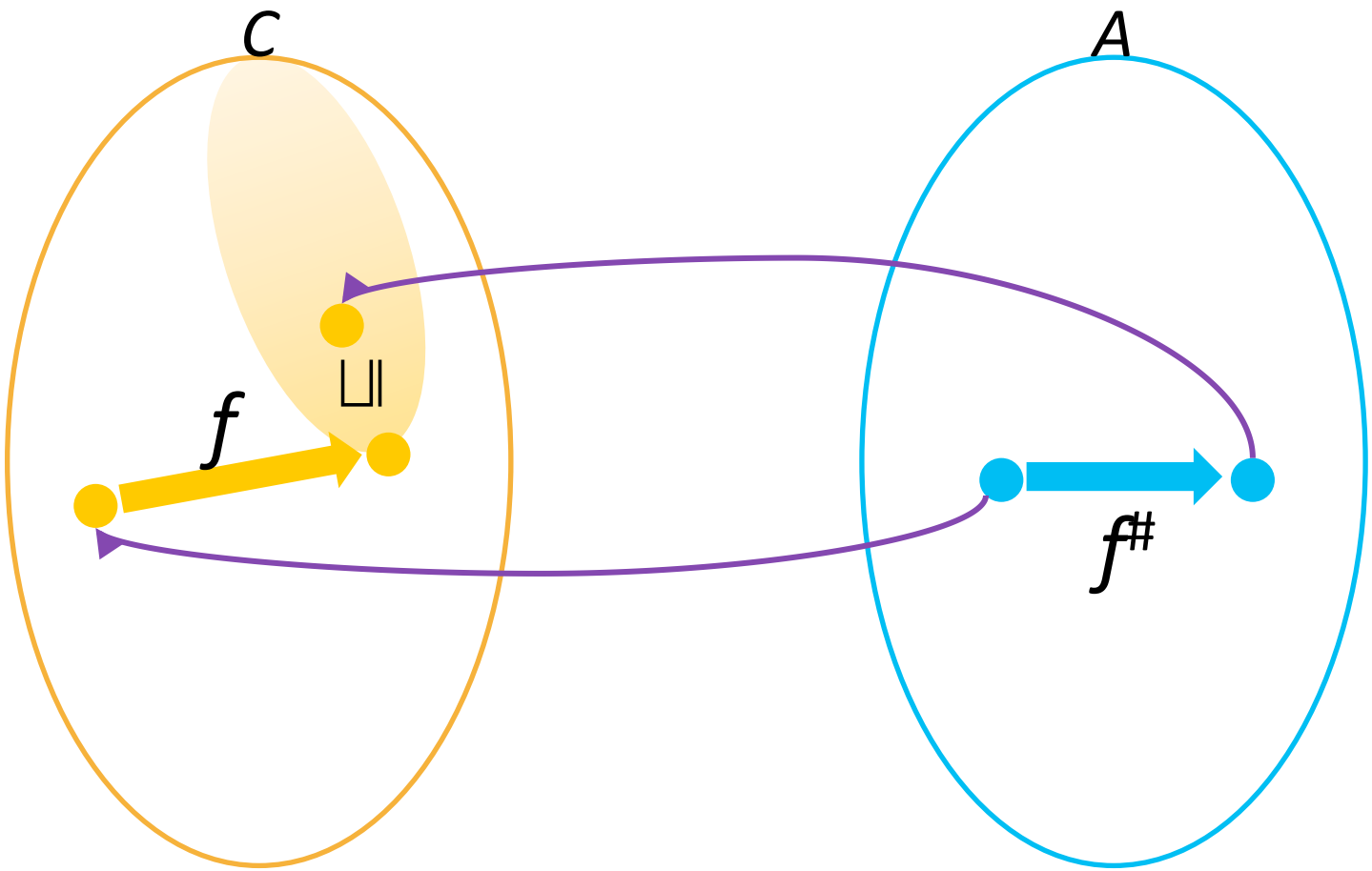
Transformer soundness condition 1

$$\forall c: f(c)=c' \Rightarrow \alpha(f^{\#}(c)) \sqsupseteq \alpha(c')$$



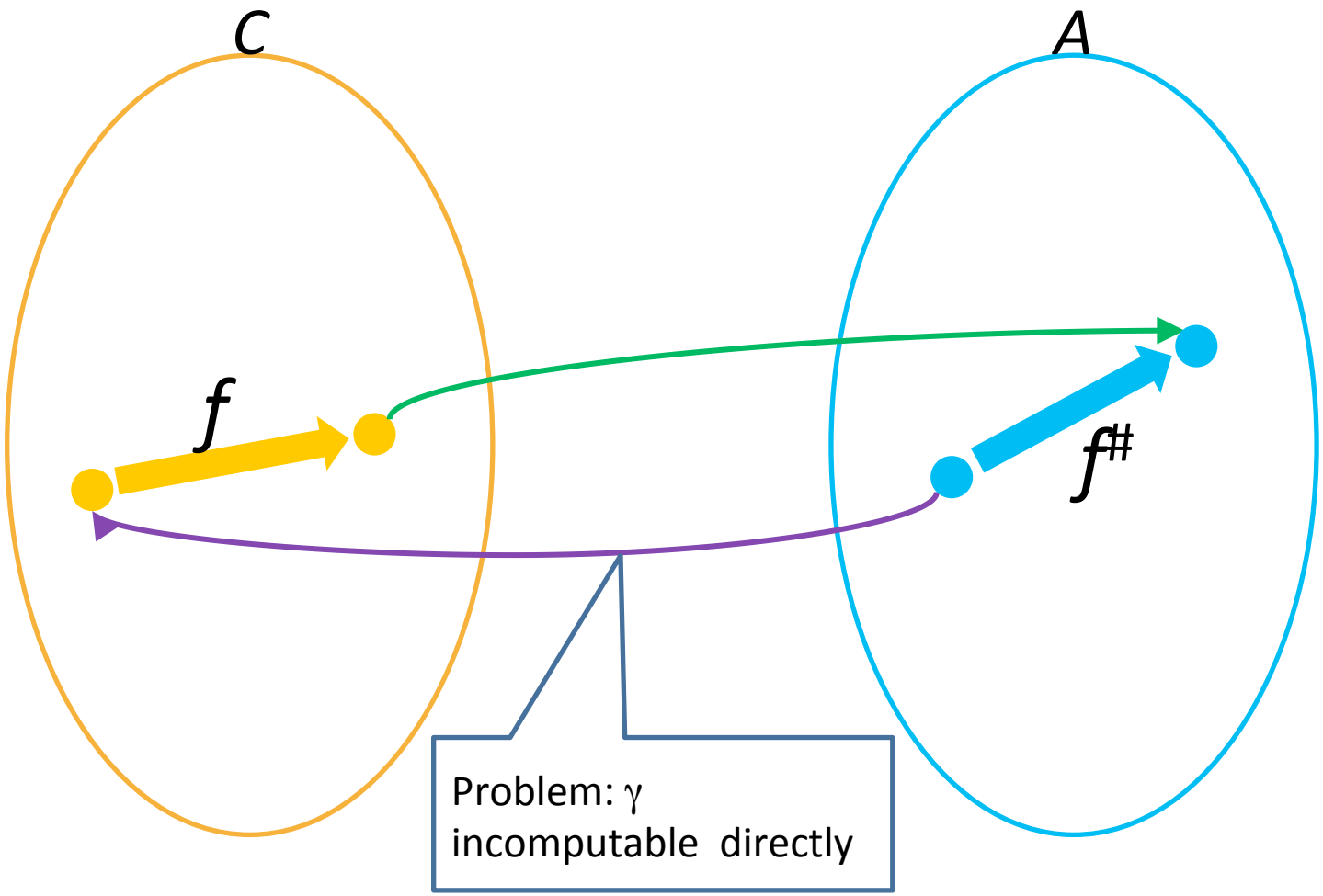
Transformer soundness condition 2

$$\forall a: f^{\#}(a)=a' \Rightarrow f(\gamma(a)) \sqsubseteq \gamma(a')$$



Best (induced) transformer

$$f^{\#}(a) = \alpha(f(\gamma(a)))$$



Best abstract transformer [CC'77]

- Best in terms of **precision**
  - Most precise abstract transformer
  - May be too expensive to compute
- Constructively defined as

$$f^{\#} = \alpha \circ f \circ \gamma$$

  - Induced by the GC
- Not directly computable because first step is concretization
- We often compromise for a "good enough" transformer
  - Useful tool: partial concretization

Transformer example

- $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- $EQ = \{ x{=}y \mid x, y \in \mathrm{Var}\}$
  $A = (2^{EQ}, \supseteq, \cap, \cup, EQ\ , \varnothing)$
- $\beta(s) = \alpha(\{s\}) = \{ x{=}y \mid s\ x = s\ y\}$ that is $s \vDash x{=}y$
  $\alpha(X) = \cap\{\beta(s) \mid s{\in}X\} = \sqcup^A \{\beta(s) \mid s{\in}X\}$
  $\gamma(\varphi) = \{ s \mid s \vDash \varphi \} = \mathrm{models}(\varphi)$
- Concrete: $[\![x{:=}y]\!]\ X = \{ s[x{\mapsto}s\ y] \mid s{\in}X\}$
- Abstract: $[\![x{:=}y]\!]^{\#}\ X = ?$

Developing a transformer for *EQ* - 1

- Input has the form $X = \bigwedge\{a=b\}$
- $\mathrm{sp}(x:=expr,\ \varphi) = \exists v.\ x=\mathrm{expr}[v/\mathrm{x}] \wedge \varphi[v/\mathrm{x}]$
- $\mathrm{sp}(x:=y,\ X) = \exists v.\ x=y[v/\mathrm{x}] \wedge \bigwedge\{a=b\}[v/\mathrm{x}] = \ldots$
- Let's define helper notations:
  - $\mathrm{EQ}(X, y) = \{y=a,\ b=y \in X\}$
    - Subset of equalities containing *y*
  - $\mathrm{EQc}(X, y) = X \setminus \mathrm{EQ}(X, y)$
    - Subset of equalities **not** containing *y*

Developing a transformer for *EQ* - 2

- sp(*x*:=*y*, *X*) = $\exists v.\ x{=}y[v/x] \wedge \bigwedge\{a{=}b\}[v/x]$ = …
- Two cases
  - *x* is *y*: sp(*x*:=*y*, *X*) = *X*
  - *x* is different from *y*:
    sp(*x*:=*y*, *X*) = $\exists v.\ x{=}y \wedge \mathrm{EQ}_{)}\mathrm{X}, \mathrm{x})[v/x] \wedge \mathrm{EQc}(X, x)[v/x]$
    $= x{=}y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}_{)}\mathrm{X}, \mathrm{x})[v/x]$
    $\Longrightarrow x{=}y \wedge \mathrm{EQc}(X, x)$
- Vanilla transformer: $[\![x{:=}y]\!]^{\#1}\ X = x{=}y \wedge \mathrm{EQc}(X, x)$
- Example: $[\![x{:=}y]\!]^{\#1} \bigwedge\{\mathrm{x{=}p, q{=}x, m{=}n}\} = \bigwedge\{\mathrm{x{=}y, m{=}n}\}$
  Is this the most precise result?

Developing a transformer for *EQ* - 3

- $[\![x{:=}y]\!]^{\#1} \bigwedge\{x{=}p, x{=}q, m{=}n\} = \bigwedge\{x{=}y, m{=}n\} \sqsupseteq \bigwedge\{x{=}y, m{=}n, p{=}q\}$
  - Where does the information p=q come from?
- sp($x{:=}y$, X) =

  $x{=}y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}_{)}(X, x)[v/x]$
- $\exists v.\ \mathrm{EQ}_{)}(X, x)[v/x]$ holds possible equalities between different *a*'s and *b*'s – how can we account for that?

Developing a transformer for *EQ* - 4

- Define a reduction operator:
  Explicate(X) = if exist $\{a=b, b=c\} \subseteq X$
  but not $\{a=c\} \subseteq X$ then
  Explicate($X \cup \{a=c\}$)
  else
  X
- Define $[\![x:=y]\!]^{\#2} = [\![x:=y]\!]^{\#1} \circ$ Explicate
- $[\![x:=y]\!]^{\#2}(\bigwedge\{x=p, x=q, m=n\}) = \bigwedge\{x=y, m=n, p=q\}$
  is this the best transformer?

Developing a transformer for *EQ* - 5

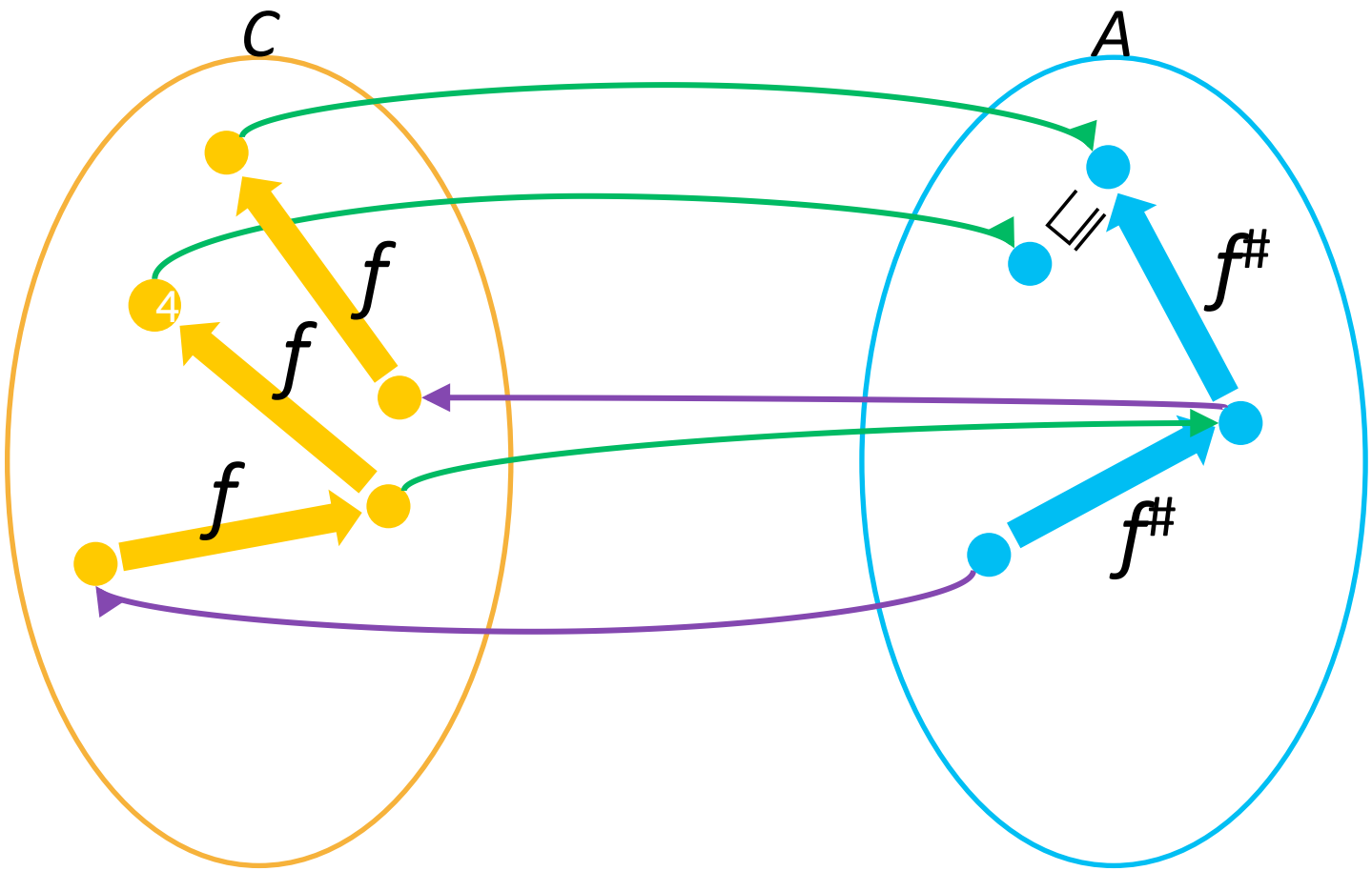
- $[\![x{:=}y]\!]^{\#2}\ )\wedge\{y{=}z\}) = \{x{=}y,\ y{=}z\} \sqsupseteq \{x{=}y,\ y{=}z,\ x{=}z\}$
- Idea: apply reduction operator again after the vanilla transformer
- $[\![x{:=}y]\!]^{\#3}$ = Explicate $\circ$ $[\![x{:=}y]\!]^{\#1}$ $\circ$ Explicate
- Observation : after the first time we apply Explicate, all subsequent values will be in the image of the abstraction so really we only need to apply it once to the input
- Finally: $[\![x{:=}y]\!]^{\#}(X)$ = Explicate $\circ$ $[\![x{:=}y]\!]^{\#1}$
  - Best transformer for reduced elements (elements in the image of the abstraction)

Negative property of best transformers

- Let $f^{\#} = \alpha \circ f \circ \gamma$
- Best transformer does not compose

$$\alpha(f(f(\gamma(a)))) \sqsubseteq f^{\#}(f^{\#}(a))$$

$$\alpha(f(f(\gamma(a)))) \sqsubseteq f^{\#}(f^{\#}(a))$$



Soundness theorem 1

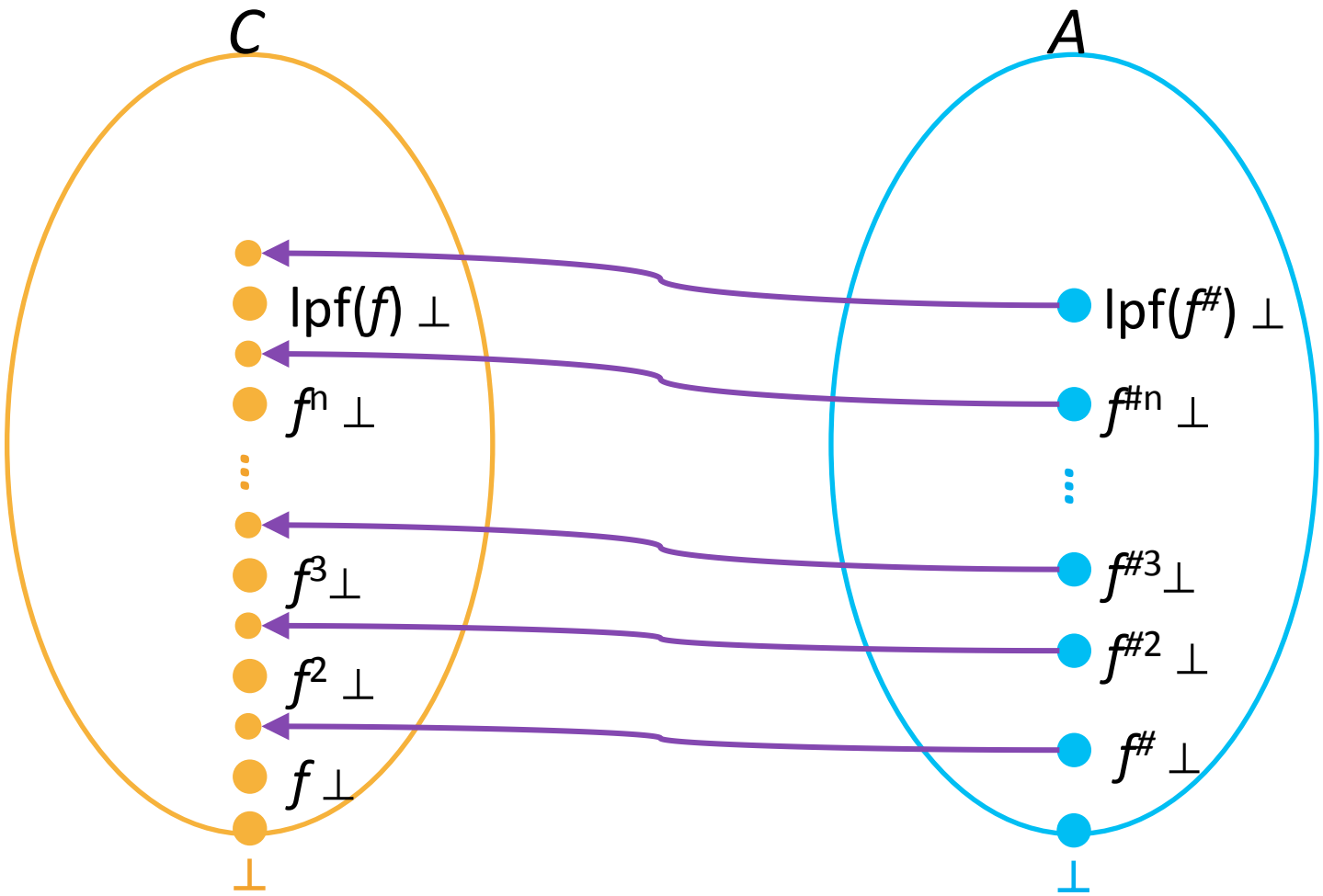
1. Given two complete lattices
   $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
   $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
   and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \longrightarrow D^C$
3. Monotone abstract transformer $f^\# : D^A \longrightarrow D^A$
4. $\forall a \in D^A : f(\gamma(a)) \sqsubseteq \gamma(f^\#(a))$

Then

$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^\#))$$
$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^\#)$$

Soundness theorem 1

$\forall a \in D^A : f(\gamma(a)) \sqsubseteq \gamma(f^\#(a)) \Rightarrow \forall a \in D^A : f^n(\gamma(a)) \sqsubseteq \gamma(f^{\#n}(a))$
$\Rightarrow \forall a \in D^A : \mathrm{lfp}(f^n)(\gamma(a)) \sqsubseteq \gamma(\mathrm{lfp}(f^{\#n})(a))$
$\Rightarrow \mathrm{lfp}(f)\,\bot \sqsubseteq \mathrm{lfp}(f^\#)\,\bot$



Soundness theorem 2

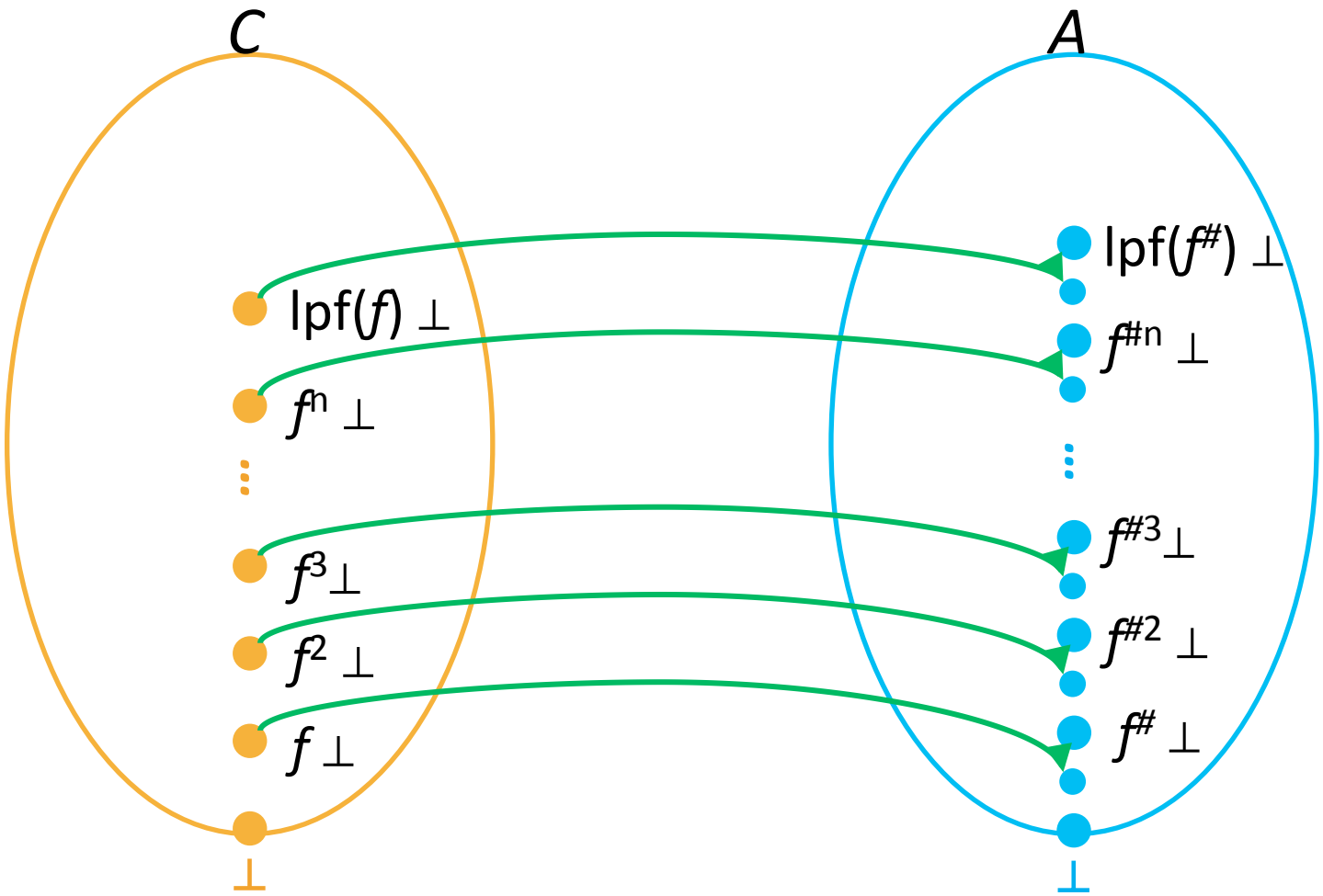
1. Given two complete lattices
   $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
   $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
   and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \longrightarrow D^C$
3. Monotone abstract transformer $f^\# : D^A \longrightarrow D^A$
4. $\forall c \in D^C : \alpha(f(c)) \sqsubseteq f^\#(\alpha(c))$

Then

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^\#)$$
$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^\#))$$

Soundness theorem 2

$\forall c \in D^C : \alpha(f(c)) \sqsubseteq f^\#(\alpha(c)) \Rightarrow \forall c \in D^C : \alpha(f^n(c)) \sqsubseteq f^{\#n}(\alpha(c))$
$\Rightarrow \forall c \in D^C : \alpha(\mathrm{lfp}(f)(c)) \sqsubseteq \mathrm{lfp}(f^\#)(\alpha(c))$
$\Rightarrow \mathrm{lfp}(f)\,\bot \sqsubseteq \mathrm{lfp}(f^\#)\,\bot$



A recipe for a sound static analysis

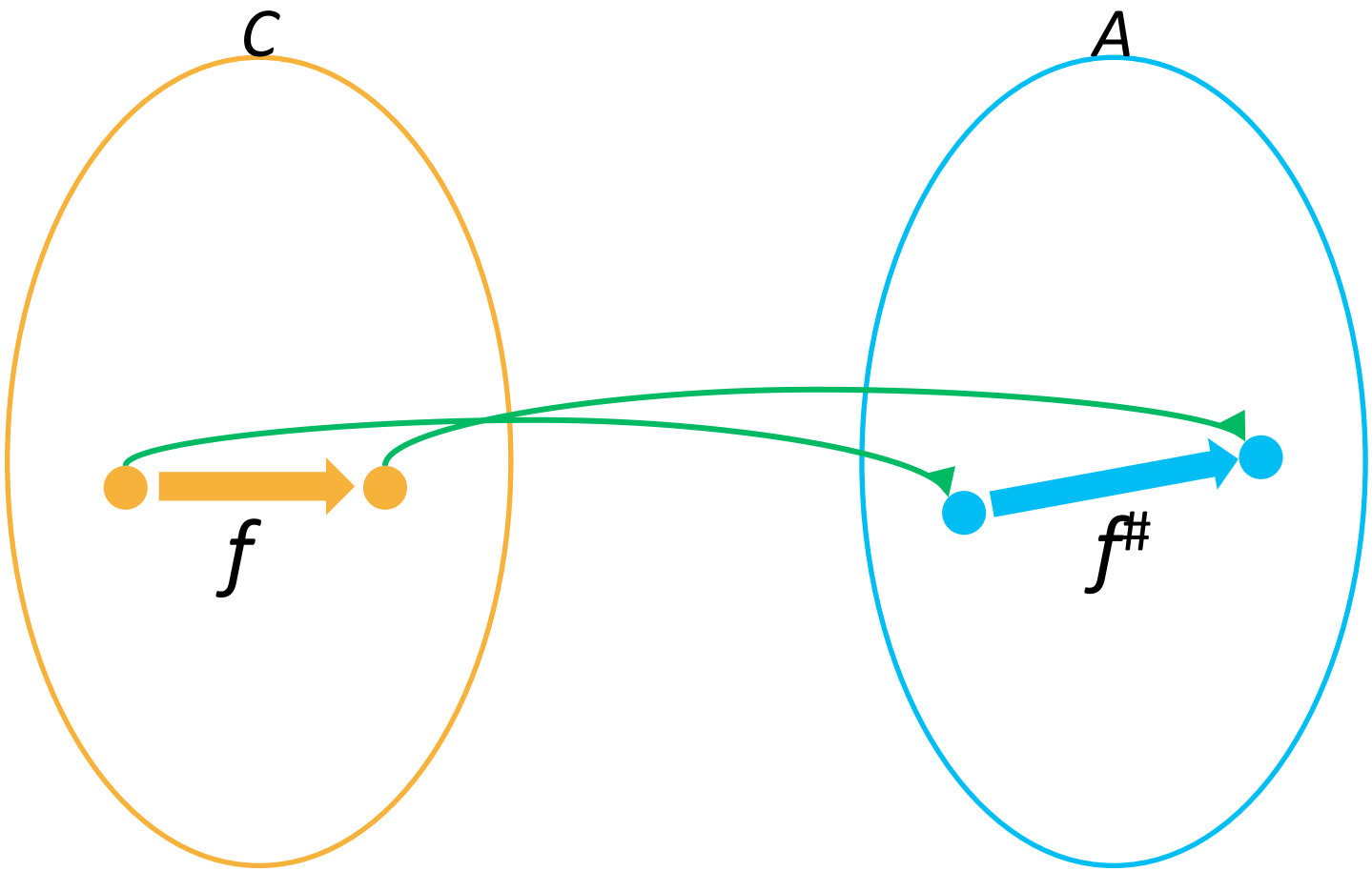
- Define an "appropriate" operational semantics
- Define "collecting" structural operational semantics
- Establish a Galois connection between collecting states and abstract states
- **Local correctness:** show that the abstract interpretation of every atomic statement is sound
  w.r.t. the collecting semantics
- **Global correctness:** conclude that the analysis is sound

Completeness

- Local property:
  - forward complete: $\forall c: \alpha(f^{\#}(c)) = \alpha(f(c))$
  - backward complete: $\forall a: f(\gamma(a)) = \gamma(f^{\#}(a))$
- A property of domain and the (best) transformer
- Global property:
  - $\alpha(\text{lfp}(f)) = \text{lfp}(f^{\#})$
  - $\text{lfp}(f) = \gamma(\text{lfp}(f^{\#}))$
- Very ideal but usually not possible unless we change the program model (apply strong abstraction) and/or aim for very simple properties

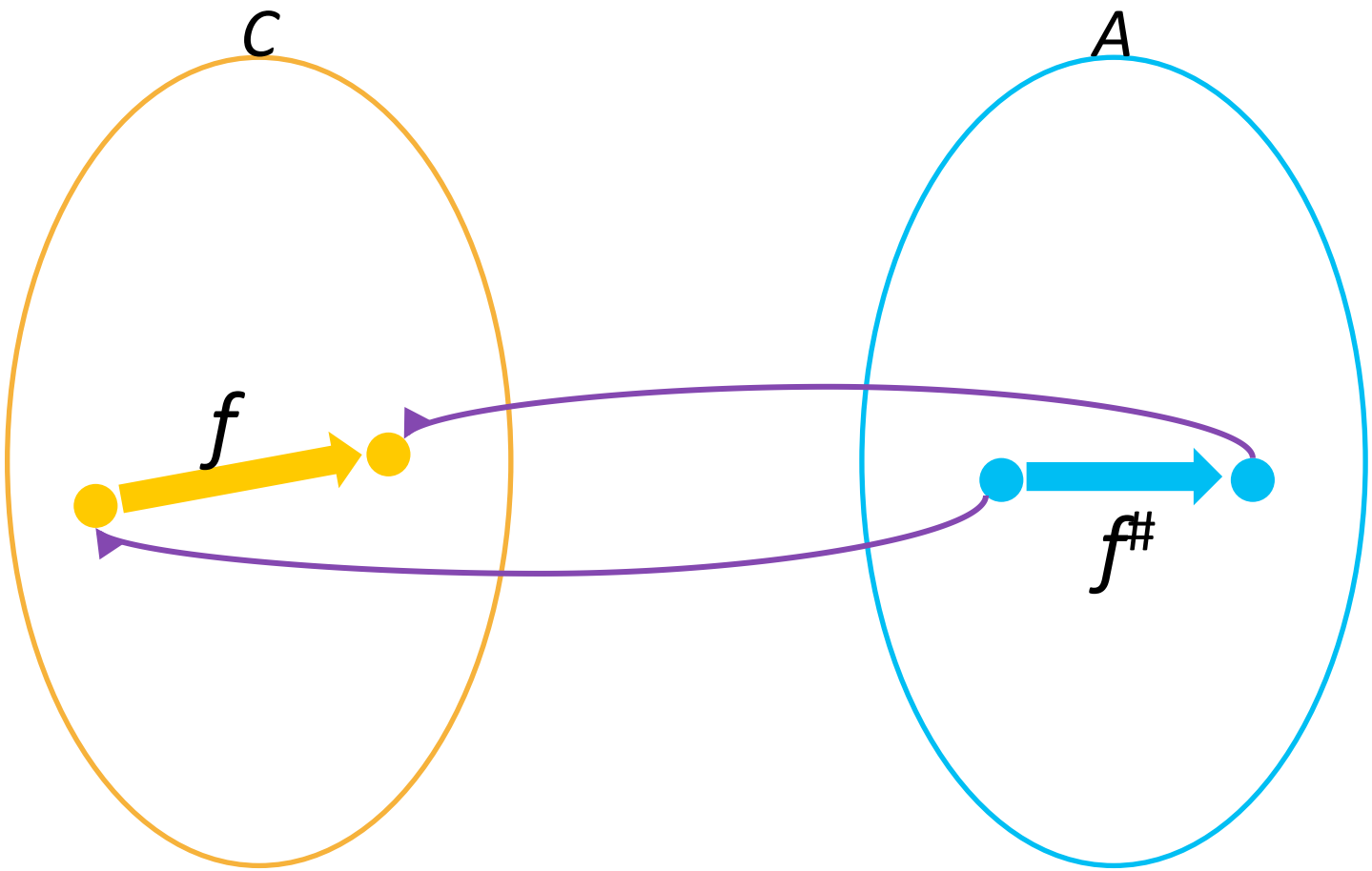
Forward complete transformer

$\forall c: \alpha(f^{\#}(c)) = \alpha(f(c))$



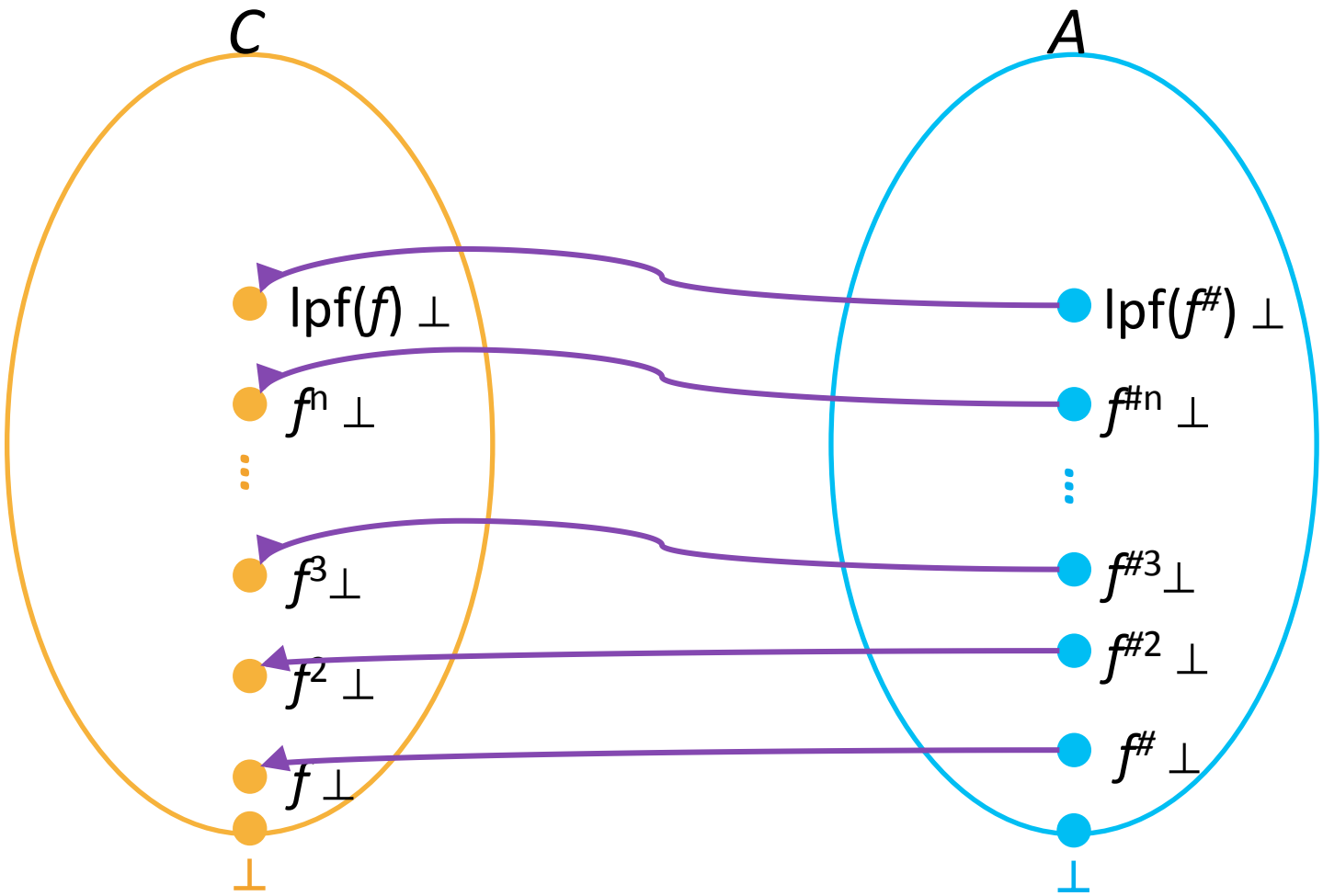
Backward complete transformer

$$\forall a: f(\gamma(a)) = \gamma(f^{\#}(a))$$



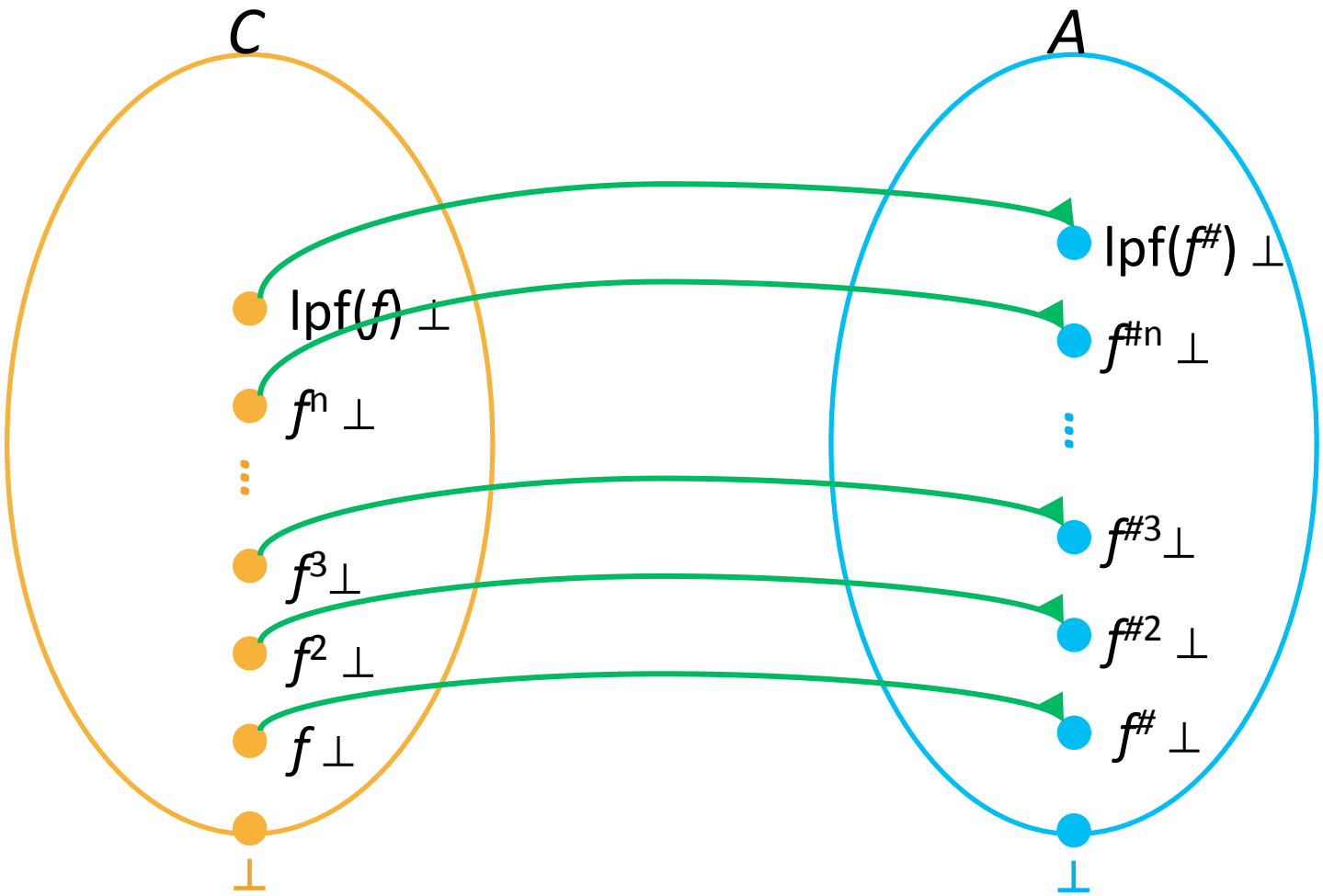
Global (backward) completeness

$\forall a: f(\gamma(a)) = \gamma(f^{\#}(a)) \Rightarrow \forall a: f^n(\gamma(a)) = \gamma(f^{\#n}(a))$
$\Rightarrow \forall a \in D^A : \mathrm{lfp}(f^n)(\gamma(a)) = \gamma(\mathrm{lfp}(f^{\#n})(a))$
$\Rightarrow \mathrm{lfp}(f)\,\bot = \mathrm{lfp}(f^{\#})\,\bot$



Global (forward) completeness

$\forall c \in D^C : \alpha(f(c)) = f^\#(\alpha(c)) \Rightarrow \forall c \in D^C : \alpha(f^n(c)) = f^{\#n}(\alpha(c))$
$\Rightarrow \forall c \in D^C : \alpha(\mathrm{lfp}(f)(c)) = \mathrm{lfp}(f^\#)(\alpha(c))$
$\Rightarrow \mathrm{lfp}(f)\,\bot = \mathrm{lfp}(f^\#)\,\bot$



Three example analyses

- Abstract states are conjunctions of constraints
- **V**ariable **E**qualities
  - *VE-factoids* = { $x=y \mid x, y \in \text{Var}\} \cup \text{false}$
    $VE = (2^{VE\text{-}factoids}, \supseteq, \cap, \cup, \mathit{false}, \varnothing)$
- **C**onstant **P**ropagation
  - *CP-factoids* = { $x=c \mid x \in \text{Var}, c \in \mathbf{Z}\} \cup \text{false}$
    $CP = (2^{CP\text{-}factoids}, \supseteq, \cap, \cup, \mathit{false}, \varnothing)$
- **A**vailable **E**xpressions
  - *AE-factoids* = { $x=y+z \mid x \in \text{Var}, y,z \in \text{Var} \cup \mathbf{Z}\} \cup \text{false}$
    $A = (2^{AE\text{-}factoids}, \supseteq, \cap, \cup, \mathit{false}, \varnothing)$

Lattice combinators reminder

- Cartesian Product
  - $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
    $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
  - $\text{Cart}(L_1, L_2) = (D_1 \times D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$
- Disjunctive completion
  - $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
  - $\text{Disj}(L) = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
- Relational Product
  - $\text{Rel}(L_1, L_2) = \text{Disj}(\text{Cart}(L_1, L_2))$

Cartesian product of complete lattices

- For two complete lattices
  $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- Define the poset
  $L_{cart} = (D_1 \times D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$
  as follows:
  - $(x_1, x_2) \sqsubseteq_{cart} (y_1, y_2)$ iff
    $x_1 \sqsubseteq_1 y_1$ and
    $x_2 \sqsubseteq_2 y_2$
  - $\sqcup_{cart} = ?$ $\sqcap_{cart} = ?$ $\bot_{cart} = ?$ $\top_{cart} = ?$
- **Lemma:** L is a complete lattice
- Define the Cartesian constructor $L_{cart} = \text{Cart}(L_1, L_2)$

Cartesian product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Cartesian Product

  $GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$

  – $\alpha^{C,A\times B}(X)$= ?

  – $\gamma^{A\times B,C}(Y)$ = ?

Cartesian product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Cartesian Product

  $$GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$$

  – $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$
  – $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- What about transformers?

Cartesian product transformers

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$ $F^A[st] : A \rightarrow A$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$ $F^B[st] : B \rightarrow B$
- Cartesian Product

  $GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$

  – $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$

  – $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- How should we define $F^{A\times B}[st] : A\times B \rightarrow A\times B$

Cartesian product transformers

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$ $F^A[st] : A \longrightarrow A$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$ $F^B[st] : B \longrightarrow B$
- Cartesian Product

  $GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$

  – $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$

  – $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- How should we define $F^{A\times B}[st] : A\times B \longrightarrow A\times B$
- Idea: $F^{A\times B}[st](a, b) = (F^A[st]\ a, F^B[st]\ b)$
- Are component-wise transformers precise?

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP analysis

```
a := 9;     {a=9}
b := 9;     {a=9, b=9}
c := a;     {a=9, b=9, c=9}
```

VE analysis

```
a := 9;     {}
b := 9;     {}
c := a;     {c=a}
```

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP analysis + VE analysis

```
a := 9;      {a=9}
b := 9;      {a=9, b=9}
c := a;      {a=9, b=9, c=9, c=a}
```

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP×VE analysis

```
a := 9;      {a=9}
b := 9;      {a=9, b=9}
c := a;      {a=9, b=9, c=9, c=a} {a=b, b=c}
```

Missing

Transformers for Cartesian product

- Naïve (component-wise) transformers do not utilize information from both components
  - Same as running analyses separately and then combining results
- Can we treat transformers from each analysis as black box and obtain best transformer for their combination?

Can we combine transformer modularly?

- No generic method for any abstract interpretations

Reducing values for CP×VE

- X = set of CP constraints of the form $x=c$ (e.g., **`a=9`**)
- Y = set of VE constraints of the form x=y
- $\text{Reduce}^{\text{CP}\times\text{VE}}(X, Y) = (X', Y')$ such that $(X', Y') \sqsubseteq (X', Y')$
- Ideas?

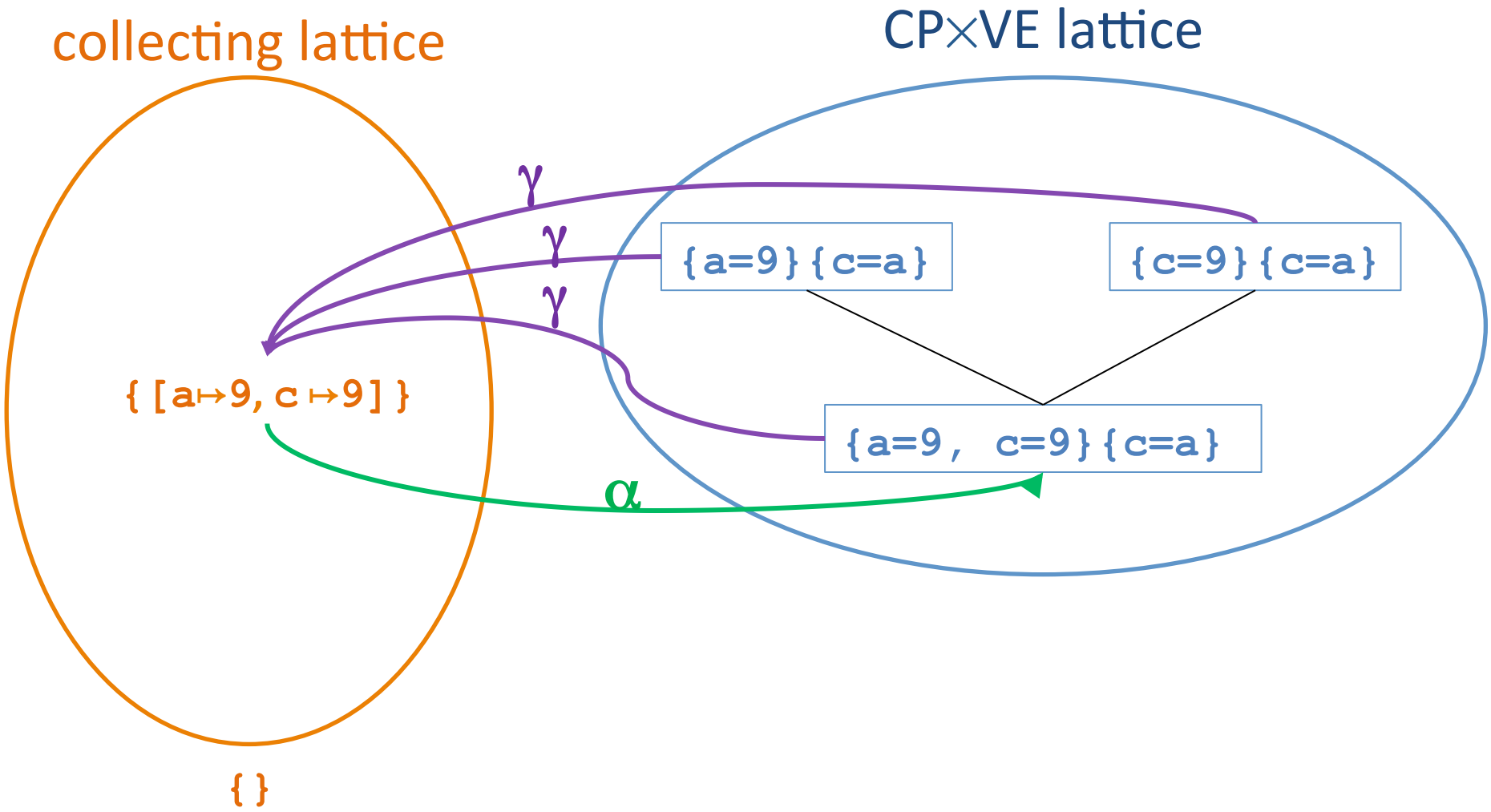
Reducing values for CP×VE

- X = set of CP constraints of the form $x=c$ (e.g., **`a=9`**)
- Y = set of VE constraints of the form x=y
- $\text{Reduce}^{CP\times VE}(X, Y) = (X', Y')$ such that $(X', Y') \sqsubseteq (X', Y')$
- ReduceRight:
  - if $a=b \in X$ and $a=c \in Y$ then add b=c to Y
- ReduceLeft:
  - If a=c and $b=c \in Y$ then add a=b to X
- Keep applying ReduceLeft and ReduceRight and reductions on each domain separately until reaching a fixed-point

Transformers for Cartesian product

- Do we get the best transformer by applying component-wise transformer followed by reduction?
  - Unfortunately, no (what's the intuition?)
  - Can we do better?
  - Logical Product [Gulwani and Tiwari, PLDI 2006]

Product vs. reduced product



Reduced product

- For two complete lattices

  $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$

  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$

- Define the reduced poset

  $D_1 \sqcap D_2 = \{(d_1,d_2) \in D_1 \times D_2 \mid (d_1,d_2) = \alpha \circ \gamma\ (d_1,d_2)\ \}$

  $L_1 \sqcap L_2 = (D_1 \sqcap D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$

Transformers for Cartesian product

- Do we get the best transformer by applying component-wise transformer followed by reduction?
  - Unfortunately, no (what's the intuition?)
  - Can we do better?
  - Logical Product [Gulwani and Tiwari, PLDI 2006]

Combining Abstract Interpreters

Sumit Gulwani
Microsoft Research
sumitg@microsoft.com

Ashish Tiwari
SRI International
tiwari@csl.sri.com

Abstract

We present a methodology for automatically combining abstract interpreters over given lattices to construct an abstract interpreter for the combination of those lattices. This lends modularity to the process of design and implementation of abstract interpreters.

We define the notion of logical product of lattices. This kind of combination is more precise than the reduced product combination. We give algorithms to obtain the join operator and the existential quantification operator for the combined lattice from the corresponding operators of the individual lattices. We also give a bound on the number of steps required to reach a fixed point across loops during analysis over the combined lattice in terms of the corresponding bounds for the individual lattices. We prove that our combination methodology yields the most precise abstract interpretation operators over the logical product of lattices when the individual lattices are over theories that are convex, stably infinite, and disjoint.

We also present an interesting application of logical product wherein some lattices can be reduced to combination of other (unrelated) lattices with known abstract interpreters.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Program analysis

General Terms Algorithms, Theory, Verification

Keywords Abstract Interpreter, Logical Product, Reduced Product, Nelson-Oppen Combination

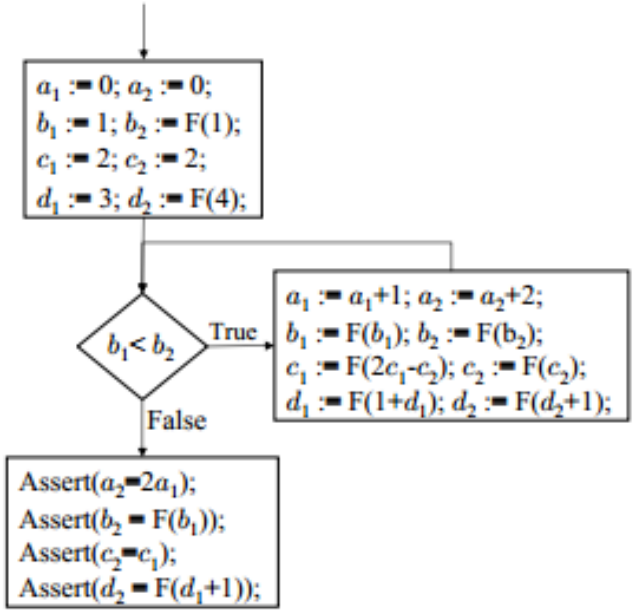


Figure 1. This program illustrates the difference between precision of performing analysis over *direct product*, *reduced product*, and *logical product* of the linear arithmetic lattice and uninterpreted functions lattice. Analysis over direct product can verify the first two assertions, while analysis over reduced product can verify the first three assertions. The analysis over logical product can verify all assertions. F denotes some function without any side-effects and can be modeled as an uninterpreted function for purpose of proving the assertions.

Logical product--

- Assume $A=(D,\ldots)$ is an abstract domain that supports two operations: for $x \in D$
  - inferEqualities(x) = { $a=b$ | $\gamma(x) \models a=b$ }
    returns a set of equalities between variables that are satisfied in all states given by x
  - refineFromEqualities(x, {$a=b$}) = y
    such that
    - $\gamma(x)=\gamma(y)$
    - $y \sqsubseteq x$

Developing a transformer for *EQ* - 1

- Input has the form $X = \bigwedge\{a=b\}$
- $sp(x:=expr,\ \varphi) = \exists v.\ x=expr[v/x] \wedge \varphi[v/x]$
- $sp(x:=y,\ X) = \exists v.\ x=y[v/x] \wedge \bigwedge\{a=b\}[v/x] = \ldots$
- Let's define helper notations:
  - $EQ(X, y) = \{y=a, b=y \in X\}$
    - Subset of equalities containing *y*
  - $EQc(X, y) = X \setminus EQ(X, y)$
    - Subset of equalities **not** containing *y*

Developing a transformer for *EQ* - 2

- sp(*x*:=*y*, *X*) = $\exists v.\ x{=}y[v/\mathrm{x}] \wedge \bigwedge\{a{=}b\}[v/\mathrm{x}]$ = …
- Two cases
  - *x* is *y*: sp(*x*:=*y*, *X*) = *X*
  - *x* is different from *y*:
    sp(*x*:=*y*, *X*) = $\exists v.\ x{=}y \wedge \mathrm{EQ}_{)}\mathrm{X}, \mathrm{x})[v/\mathrm{x}] \wedge \mathrm{EQc}(X, x)[v/\mathrm{x}]$
    = $x{=}y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}_{)}\mathrm{X}, \mathrm{x})[v/\mathrm{x}]$
    $\Rightarrow x{=}y \wedge \mathrm{EQc}(X, x)$
- Vanilla transformer: $[\![x{:=}y]\!]^{\#1}\ X = x{=}y \wedge \mathrm{EQc}(X, x)$
- Example: $[\![x{:=}y]\!]^{\#1} \bigwedge\{\mathrm{x{=}p,\ q{=}x,\ m{=}n}\} = \bigwedge\{\mathrm{x{=}y,\ m{=}n}\}$
  Is this the most precise result?

Developing a transformer for *EQ* - 3

- $[\![x{:=}y]\!]^{\#1}\ \bigwedge\{x{=}p, x{=}q, m{=}n\} = \bigwedge\{x{=}y, m{=}n\} \sqsupseteq \bigwedge\{x{=}y, m{=}n, p{=}q\}$
  - Where does the information p=q come from?
- sp($x{:=}y$, X) =

  $x{=}y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}_{)}(X, x)[v/x]$
- $\exists v.\ \mathrm{EQ}_{)}(X, x)[v/x]$ holds possible equalities between different *a*'s and *b*'s – how can we account for that?

Developing a transformer for *EQ* - 4

- Define a reduction operator:
  Explicate(X) = if exist {a=b, b=c}$\subseteq$X
  but not {a=c} $\subseteq$X then
  Explicate(X$\cup${a=c})
  else
  X
- Define $[\![x:=y]\!]^{\#2}$ = $[\![x:=y]\!]^{\#1} \circ$ Explicate
- $[\![x:=y]\!]^{\#2}(\bigwedge\{x=p, x=q, m=n\}) = \bigwedge\{x=y, m=n, p=q\}$
  is this the best transformer?

Developing a transformer for *EQ* - 5

- $[\![x{:=}y]\!]^{\#2}\ )\wedge\{y{=}z\}) = \{x{=}y,\ y{=}z\} \sqsupseteq \{x{=}y,\ y{=}z,\ x{=}z\}$
- Idea: apply reduction operator again after the vanilla transformer
- $[\![x{:=}y]\!]^{\#3}$ = Explicate $\circ$ $[\![x{:=}y]\!]^{\#1}$ $\circ$ Explicate

Logical Product-

The element E after an assignment node $x := e$ is the strongest postcondition of the element E' before the assignment node. It is computed by using an existential quantification operator $Q_{L_1 \bowtie L_2}$ as described below.

$$
\begin{aligned}
E &= Q_{L_1 \bowtie L_2}(E_1, \{x'\}) \\
\text{where } E_1 &= E'[x'/x] \wedge E_1' \\
\text{and } E_1' &= \begin{cases} x = e[x'/x] & \text{if } \mathit{Symbols}(e) \subseteq \Sigma_{\mathbb{T}_1 \cup \mathbb{T}_2} \\ \mathit{true} & \text{otherwise} \end{cases}
\end{aligned}
$$

safely abstracting the existential quantifier

basically the strongest postcondition

Abstracting the existential

Reduce the pair

Abstract away existential quantifier for each domain

$Q_{L_1 \ltimes L_2}(E, V)$ =

1 $\langle V^0, E_1^0, E_2^0 \rangle$:= $Purify_{\mathbb{T}_1,\mathbb{T}_2}(E)$;
2 $\langle E_1^1, E_2^1 \rangle$:= $NOSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^0, E_2^0)$;
3 V^1 := $V^0 \cup V$;
4 $\langle V^2, \texttt{Defs} \rangle$:= $QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^1, E_2^1, V^1)$;
5 E_1^2 := $Q_{L_1}(E_1^1, V^2)$;
6 E_2^2 := $Q_{L_2}(E_2^1, V^2)$;
7 E_1^3 := $E_1^2[\texttt{Defs}(y)/y]$ for all $y \in V^2 - V^1$;
8 E_2^3 := $E_2^2[\texttt{Defs}(y)/y]$ for all $y \in V^2 - V^1$;
9 return $E_1^3 \wedge E_2^3$;

$QSaturation_{\mathbb{T}_1,\mathbb{T}_2}(E_1^1, E_2^1, V^1)$ =

1 $V^2 := V^1$;
2 Defs := $\emptyset$;
3 repeat
4 for all $y \in V^1$
5 $t := Alternate_{\mathbb{T}_1}(E_1^1, y, V^2)$;
6 if $t = \bot$, then $t := Alternate_{\mathbb{T}_2}(E_2^1, y, V^2)$;
7 if $t \neq \bot$, then Defs := Defs $\wedge\, y = t$;
8 $V^2 := V^2 - \{y\}$;
9 until no change in V^2;
10 return $\langle V^2$,Defs$\rangle$;

Example

Information loss example

```
if (…)          {}
  b := 5        {b=5}
else
  b := -5       {b=-5}
                {b=⊤}
if (b>0)
  b := b-5      {b=⊤}
else
  b := b+5      {b=⊤}
assert b==0     can't prove
```

Disjunctive completion of a lattice

- For a complete lattice
  $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Define the powerset lattice
  $L_\vee = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
  $\sqsubseteq_\vee$ = ? $\sqcup_\vee$ = ? $\sqcap_\vee$ = ? $\bot_\vee$ = ? $\top_\vee$ = ?
- **Lemma:** $L_\vee$ is a complete lattice
- $L_\vee$ contains all subsets of D, which can be thought of as disjunctions of the corresponding predicates
- Define the disjunctive completion constructor
  $L_\vee$ = Disj(L)

Disjunctive completion for GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Disjunctive completion
  $GC^{C,P(A)} = (C, \alpha^{P(A)}, \gamma^{P(A)}, P(A))$
  - $\alpha^{C,P(A)}(X) = ?$
  - $\gamma^{P(A),C}(Y) = ?$

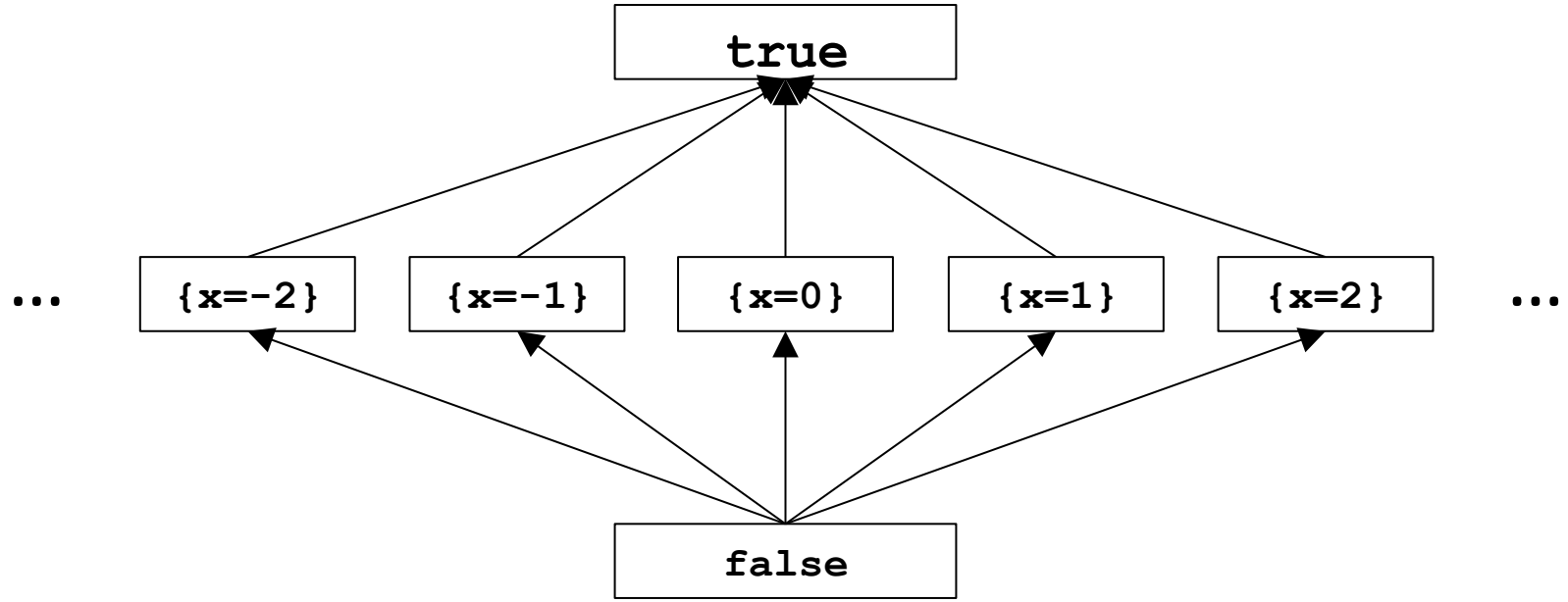
Disjunctive completion for GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Disjunctive completion
  $GC^{C,P(A)} = (C, \alpha^{P(A)}, \gamma^{P(A)}, P(A))$
  - $\alpha^{C,P(A)}(X) = \{\alpha^{C,A}(\{x\}) \mid x \in X\}$
  - $\gamma^{P(A),C}(Y) = \cup\{\gamma^{P(A)}(y) \mid y \in Y\}$
- What about transformers?

Information loss example

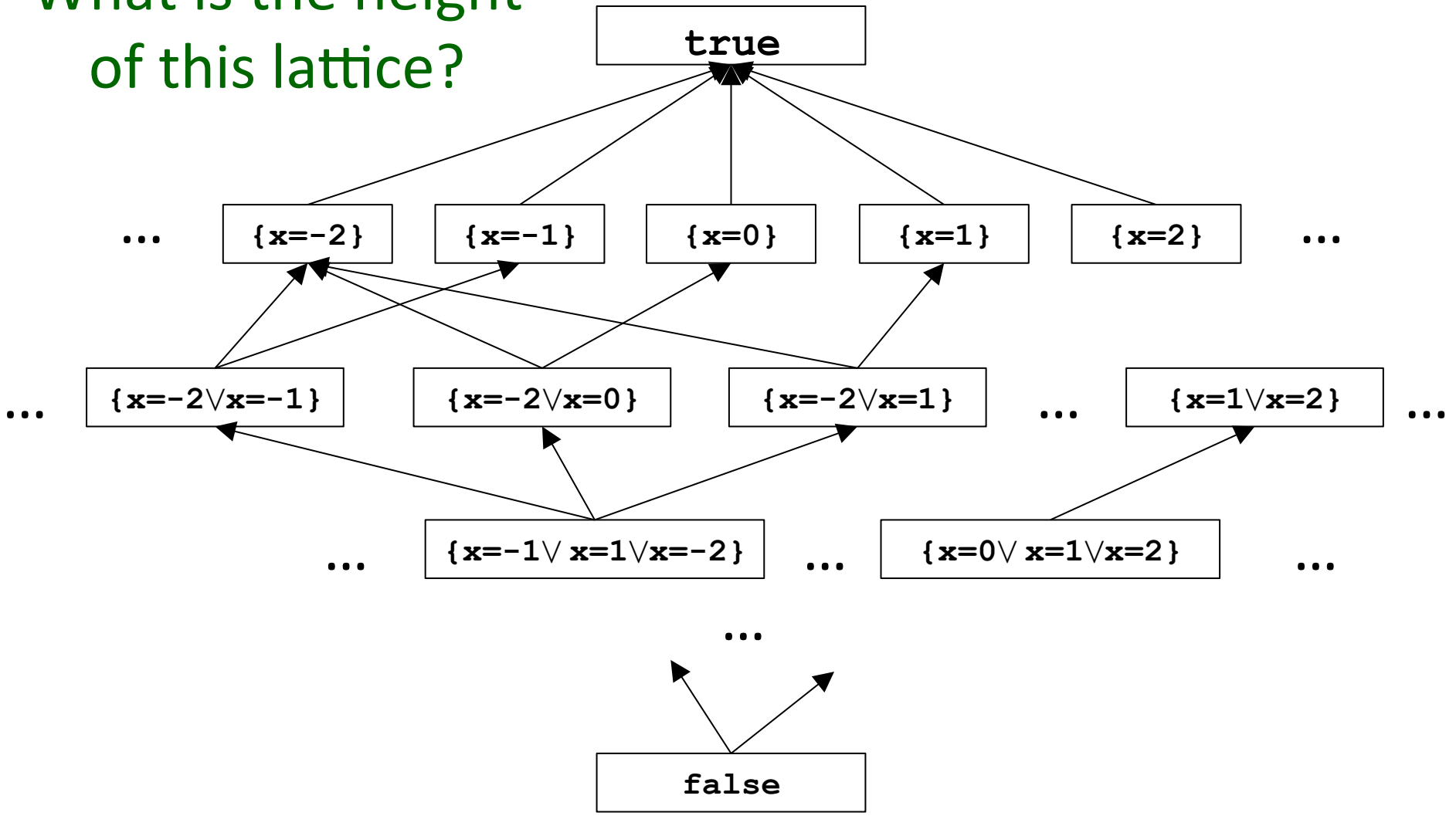
```
if (…)          {}
  b := 5        {b=5}
else
  b := -5       {b=-5}
                {b=5 ∨ b=-5}
if (b>0)
  b := b-5      {b=0}
else
  b := b+5      {b=0}
assert b==0     proved
```

The base lattice CP



The disjunctive completion of CP

What is the height of this lattice?



Taming disjunctive completion

- Disjunctive completion is very precise
  - Maintains correlations between states of different analyses
  - Helps handle conditions precisely
  - But very expensive – number of abstract states grows exponentially
  - May lead to non-termination
- Base analysis (usually product) is less precise
  - Analysis terminates if the analyses of each component terminates
- How can we combine them to get more precision yet ensure termination and state explosion?

Taming disjunctive completion

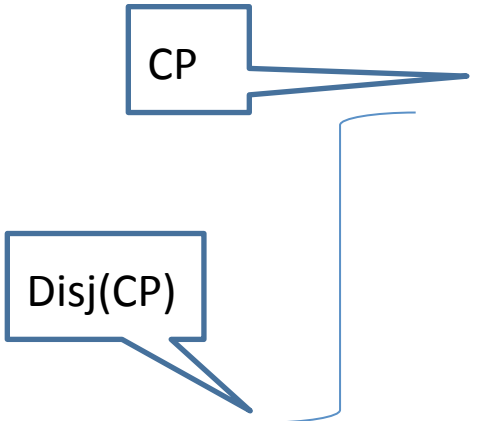
- Use different abstractions for different program locations
  - At loop heads use coarse abstraction (base)
  - At other points use disjunctive completion
- Termination is guaranteed (by base domain)
- Precision increased **inside** loop body

With Disj(CP)

```
while (…) {
  if (…)
    b := 5
  else
    b := -5

  if (b>0)
    b := b-5
  else
    b := b+5
  assert b==0
}
```

Doesn't terminate

With tamed Disj(CP)




```
while (…) {
  if (…)
    b := 5
  else
    b := -5

  if (b>0)
    b := b-5
  else
    b := b+5
  assert b==0
}
```

terminates

What **`MultiCartDomain`** implements

Reducing disjunctive elements

- A disjunctive set X may contain within it an ascending chain $Y=a \sqsubseteq b \sqsubseteq c\ldots$
- We only need max(Y) – remove all elements below

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = ?

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = Disj(Cart(L_1, L_2))
- **Lemma:** L is a complete lattice
- What does it buy us?
  - How is it relative to Cart(Disj(L_1), Disj(L_2))?
- What about transformers?

Relational product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Relational Product
  $GC^{C,P(A\times B)} = (C, \alpha^{C,P(A\times B)}, \gamma^{P(A\times B),C}, P(A\times B))$
  - $\alpha^{C,P(A\times B)}(X) = ?$
  - $\gamma^{P(A\times B),C}(Y) = ?$

Relational product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Relational Product
  $GC^{C,P(A\times B)} = (C, \alpha^{C,P(A\times B)}, \gamma^{P(A\times B),C}, P(A\times B))$
  - $\alpha^{C,P(A\times B)}(X) = \{(\alpha^{C,A}(\{x\}), \alpha^{C,B}(\{x\})) \mid x\in X\}$
  - $\gamma^{P(A\times B),C}(Y) = \cup\{\gamma^{A,C}(y_A) \cap \gamma^{B,C}(y_B) \mid (y_A,y_B)\in Y\}$

Cartesian product example

```
V[10] = V[9] // goto [?= (branch)]
V[11] = P(Reduce_([AssignConstantToVarTransformer, Id]))(V[6]) // b = 9
V[12] = P(Reduce_([AssignVarToVarTransformer, Reduce_VEDomain(AssignVarToVarTransformer)]))(V[11]) // a = d
V[15] = Join_DisjunctiveDomain(V[10], V[12]) // if b != 8 goto (branch)
```

```
public void relationalProductExample(int a, int b, int c, int d) {
    if (a > 5) {
        b = 8;
        a = c;
    } else {
        b = 9;
        a = d;
    }

    if (b == 8) {
        if (a != c)
            error("Unable to prove a==c!");
    }
    else if (b == 9) {
        if (a != d)
            error("Unable to prove a==d!");
    }
    else {
        error("Can't get here");
    }
```

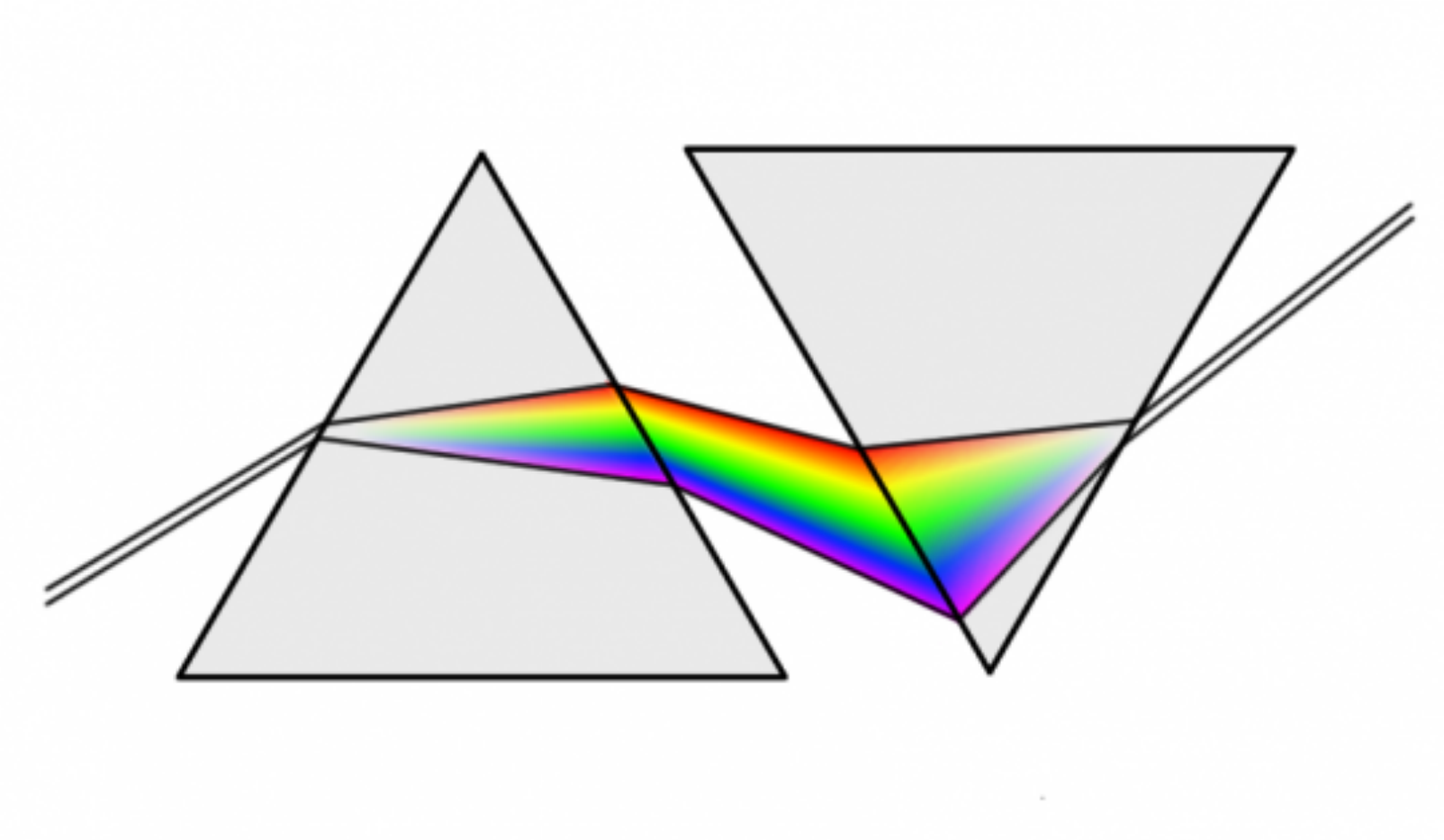
Correlations preserved

```
Reached fixed-point after 28 iterations.
Solution = {
  V[0] : (true, true)
  V[1] : (true, true)
  V[2] : (true, true)
  V[3] : (true, true)
  V[4] : (true, true)
  V[5] : (true, true)
  V[6] : (true, true)
  V[7] : (true, true)
  V[8] : (b=8, true)
  V[9] : (b=8, a=c)
  V[10] : (b=8, a=c)
  V[11] : (b=9, true)
  V[12] : (b=9, a=d)
  V[15] : or((b=9, a=d), (b=8, a=c))
  V[13] : (b=9, a=d)
  V[14] : (b=8, a=c)
  V[16] : (b=8, a=c)
  V[17] : false
  V[18] : false
  V[19] : false
  V[20] : false
  V[21] : (b=9, a=d)
  V[22] : (b=9, a=d)
  V[23] : false
  V[24] : false
  V[25] : false
  V[26] : false
  V[28] : or((b=9, a=d), (b=8, a=c))
  V[27] : or((b=9, a=d), (b=8, a=c))
}
0 possible errors found.
```

Function space

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Denote the set of monotone functions from A to B by $A \longrightarrow B$
- Define $\sqcup$ for elements of $A \longrightarrow B$ as follows
  $(a_1, b_1) \sqcup (a_2, b_2)$ = if $a_1=a_2$ then $\{(a_1, b_1 \sqcup_B b_1)\}$
  else $\{(a_1, b_1), (a_2, b_2)\}$
- Reduced cardinal power
  $GC^{C,A \to B} = (C, \alpha^{C,A \to B}, \gamma^{A \to B,C}, A \longrightarrow B)$
  - $\alpha^{C,A \to B}(X) = \sqcup\{(\alpha^{C,A}(\{x\}), \alpha^{C,B}(\{x\})) \mid x \in X\}$
  - $\gamma^{A \to B,C}(Y) = \cup\{\gamma^{A,C}(y_A) \cap \gamma^{B,C}(y_B) \mid (y_A, y_B) \in Y\}$
- Useful when A is small and B is much larger
  - E.g., typestate verification

Widening/Narrowing



How can we prove this automatically?

```
public void loopExample() {
    int x = 7;
    while (x < 1000) {
        ++x;
    }
    if (!(x == 1000))
        error("Unable to prove x == 1000!");
}
```

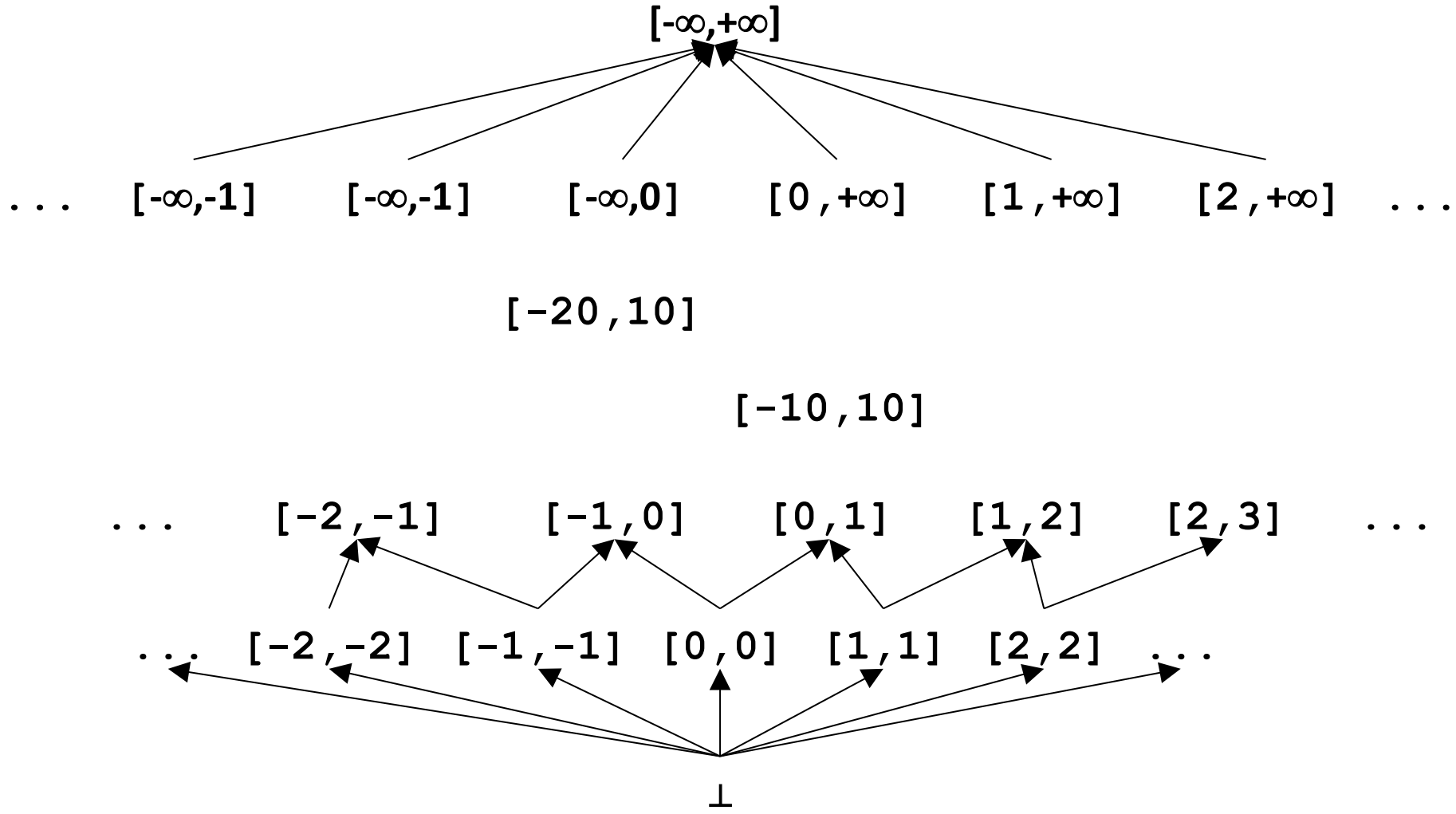
RelProd(CP, VE)

```
Reached fixed-point after 19 iterations.
Solution = {
  V[0] : (true, true)
  V[1] : (true, true)
  V[2] : (x=7, true)
  V[3] : (x=7, true)
  V[4] : (true, true)
  V[7] : (true, true)
  V[5] : (true, true)
  V[6] : (true, true)
  V[8] : (true, true)
  V[9] : (true, true)
  V[10] : (true, true)
  V[12] : (true, true)
  V[11] : (true, true)
}
1 possible errors found.
```

Intervals domain

- One of the simplest numerical domains
- Maintain for each variable *x* an interval [*L*,*H*]
  - *L* is either an integer of -∞
  - *H* is either an integer of +∞
- A (non-relational) numeric domain

Intervals lattice for variable *x*



Intervals lattice for variable *x*

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z} \text{ and } H \in \mathbf{Z},+\infty \text{ and } L \leq H \}$
- $\bot$
- $\top = [-\infty,+\infty]$
- $\sqsubseteq$ = ?
  - $[1,2] \sqsubseteq [3,4]$?
  - $[1,4] \sqsubseteq [1,3]$?
  - $[1,3] \sqsubseteq [1,4]$?
  - $[1,3] \sqsubseteq [-\infty,+\infty]$?
- What is the lattice height?

Intervals lattice for variable *x*

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z} \text{ and } H \in \mathbf{Z},+\infty \text{ and } L \leq H \}$
- $\bot$
- $\top = [-\infty,+\infty]$
- $\sqsubseteq$ = ?
  - $[1,2] \sqsubseteq [3,4]$ no
  - $[1,4] \sqsubseteq [1,3]$ no
  - $[1,3] \sqsubseteq [1,4]$ yes
  - $[1,3] \sqsubseteq [-\infty,+\infty]$ yes
- What is the lattice height? Infinite

Joining/meeting intervals

- $[a,b] \sqcup [c,d] = ?$
  - $[1,1] \sqcup [2,2] = ?$
  - $[1,1] \sqcup [2, +\infty] = ?$
- $[a,b] \sqcap [c,d] = ?$
  - $[1,2] \sqcap [3,4] = ?$
  - $[1,4] \sqcap [3,4] = ?$
  - $[1,1] \sqcap [1,+\infty] = ?$
- Check that indeed $x \sqsubseteq y$ if and only if $x \sqcup y = y$

Joining/meeting intervals

- $[a,b] \sqcup [c,d] = [\min(a,c), \max(b,d)]$
  - $[1,1] \sqcup [2,2] = [1,2]$
  - $[1,1] \sqcup [2,+\infty] = [1,+\infty]$
- $[a,b] \sqcap [c,d] = [\max(a,c), \min(b,d)]$ if a proper interval and otherwise $\bot$
  - $[1,2] \sqcap [3,4] = \bot$
  - $[1,4] \sqcap [3,4] = [3,4]$
  - $[1,1] \sqcap [1,+\infty] = [1,1]$
- Check that indeed $x \sqsubseteq y$ if and only if $x \sqcup y = y$

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z} \text{ and } H \in \mathbf{Z},+\infty \text{ and } L \leq H \}$
- For a program with variables $Var=\{x_1,\ldots,x_k\}$
- $D^{int}[Var] = ?$

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z} \text{ and } H \in \mathbf{Z},+\infty \text{ and } L \leq H \}$
- For a program with variables $Var=\{x_1,\ldots,x_k\}$
- $D^{int}[Var] = D^{int}[x_1] \times \ldots \times D^{int}[x_k]$
- How can we represent it in terms of formulas?

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty, \mathbf{Z} \text{ and } H \in \mathbf{Z}, +\infty \text{ and } L \leq H \}$
- For a program with variables $Var = \{x_1, \ldots, x_k\}$
- $D^{int}[Var] = D^{int}[x_1] \times \ldots \times D^{int}[x_k]$
- How can we represent it in terms of formulas?
  - Two types of factoids $x \geq c$ and $x \leq c$
  - Example: $S = \wedge\{x \geq 9, y \geq 5, y \leq 10\}$
  - Helper operations
    - $c + +\infty = +\infty$
    - remove(S, x) = S without any x-constraints
    - lb(S, x) =

Assignment transformers

- $[\![x := c]\!]^\# S = ?$
- $[\![x := y]\!]^\# S = ?$
- $[\![x := y+c]\!]^\# S = ?$
- $[\![x := y+z]\!]^\# S = ?$
- $[\![x := y*c]\!]^\# S = ?$
- $[\![x := y*z]\!]^\# S = ?$

Assignment transformers

- $[\![x := c]\!]^\# S = \text{remove}(S,x) \cup \{x \geq c, x \leq c\}$
- $[\![x := y]\!]^\# S = \text{remove}(S,x) \cup \{x \geq \text{lb}(S,y), x \leq \text{ub}(S,y)\}$
- $[\![x := y+c]\!]^\# S = \text{remove}(S,x) \cup \{x \geq \text{lb}(S,y)+c, x \leq \text{ub}(S,y)+c\}$
- $[\![x := y+z]\!]^\# S = \text{remove}(S,x) \cup \{x \geq \text{lb}(S,y)+\text{lb}(S,z),$
  $x \leq \text{ub}(S,y)+\text{ub}(S,z)\}$
- $[\![x := y*c]\!]^\# S = \text{remove}(S,x) \cup \text{if } c>0 \{x \geq \text{lb}(S,y)*c, x \leq \text{ub}(S,y)*c\}$
  else $\{x \geq \text{ub}(S,y)*-c, x \leq \text{lb}(S,y)*-c\}$
- $[\![x := y*z]\!]^\# S = \text{remove}(S,x) \cup ?$

assume transformers

- $[\![\textbf{assume}\ x{=}c]\!]^\#\ S$ = ?
- $[\![\textbf{assume}\ x{<}c]\!]^\#\ S$ = ?
- $[\![\textbf{assume}\ x{=}y]\!]^\#\ S$ = ?
- $[\![\textbf{assume}\ x{\neq}c]\!]^\#\ S$ = ?

assume transformers

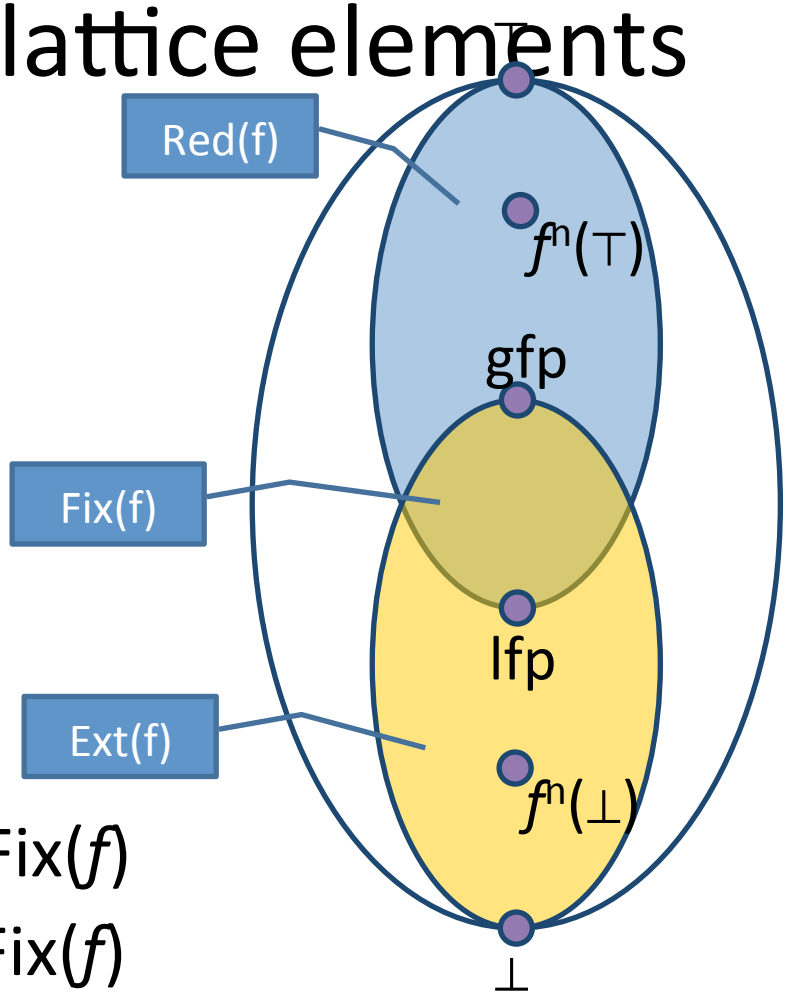
- $[\![\textbf{assume}\ x{=}c]\!]^\#\ S = S \sqcap \{x \geq c, x \leq c\}$
- $[\![\textbf{assume}\ x{<}c]\!]^\#\ S = S \sqcap \{x \leq c\text{-}1\}$
- $[\![\textbf{assume}\ x{=}y]\!]^\#\ S = S \sqcap \{x \geq \mathrm{lb}(S,y), x \leq \mathrm{ub}(S,y)\}$
- $[\![\textbf{assume}\ x{\neq}c]\!]^\#\ S = ?$

assume transformers

- $[\![\textbf{assume}\ x{=}c]\!]^\#\ S = S \sqcap \{x \geq c, x \leq c\}$
- $[\![\textbf{assume}\ x{<}c]\!]^\#\ S = S \sqcap \{x \leq c\text{-}1\}$
- $[\![\textbf{assume}\ x{=}y]\!]^\#\ S = S \sqcap \{x \geq \mathrm{lb}(S,y), x \leq \mathrm{ub}(S,y)\}$
- $[\![\textbf{assume}\ x{\neq}c]\!]^\#\ S = (S \sqcap \{x \leq c\text{-}1\}) \sqcup (S \sqcap \{x \geq c{+}1\})$

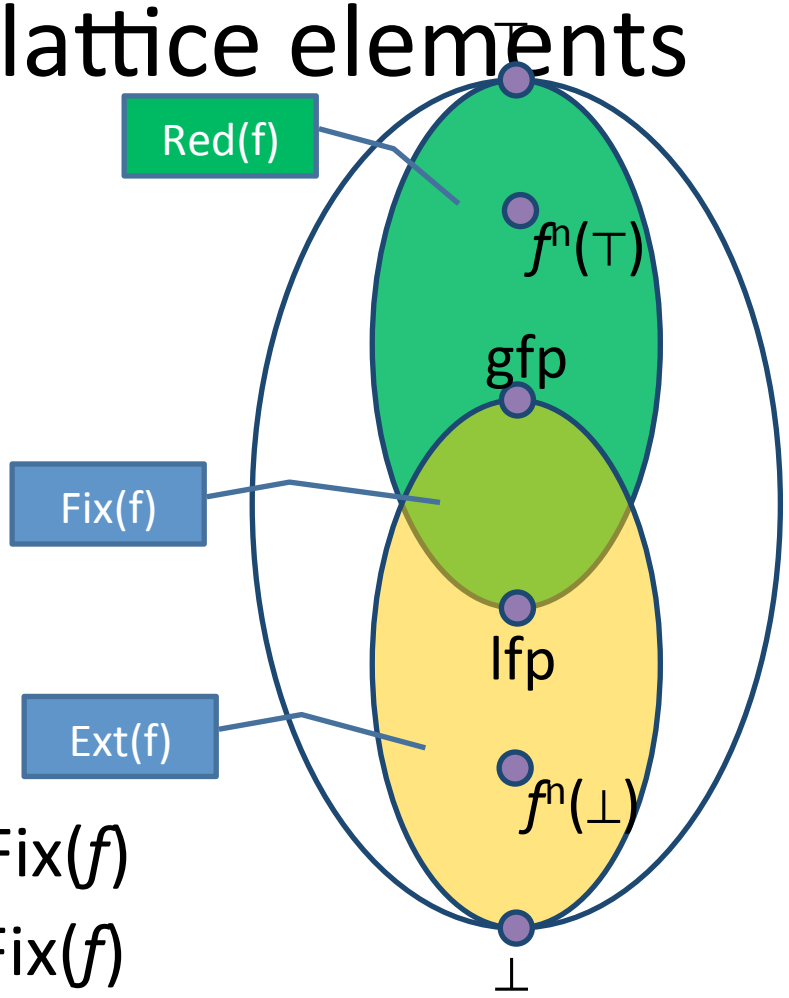
Effect of function *f* on lattice elements

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- $f : D \rightarrow D$ **monotone**
- $\text{Fix}(f) = \{ d \mid f(d) = d \}$
- $\text{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\text{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\text{lfp}(f) = \sqcap\text{Fix}(f) = \sqcap\text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \sqcup\text{Fix}(f) = \sqcup\text{Ext}(f) \in \text{Fix}(f)$



Effect of function *f* on lattice elements

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- $f : D \rightarrow D$ **monotone**
- $\text{Fix}(f) = \{ d \mid f(d) = d \}$
- $\text{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\text{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\text{lfp}(f) = \sqcap\text{Fix}(f) = \sqcap\text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \sqcup\text{Fix}(f) = \sqcup\text{Ext}(f) \in \text{Fix}(f)$



Continuity and ACC condition

- Let $L = (D, \sqsubseteq, \sqcup, \bot)$ be a complete partial order
  - Every ascending chain has an upper bound
- A function f is continuous if for every increasing chain $Y \subseteq D^*$,

$$f(\sqcup Y) = \sqcup\{ f(y) \mid y \in Y \}$$

- L satisfies the ascending chain condition (ACC) if every ascending chain eventually stabilizes:

$$d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n = d_{n+1} = \ldots$$

Fixed-point theorem [Kleene]

- Let $L = (D, \sqsubseteq, \bigsqcup, \bot)$ be a complete partial order and a **continuous** function $f: D \to D$ then

$$\mathrm{lfp}(f) = \bigsqcup_{n \in N} f^n(\bot)$$

Resulting algorithm

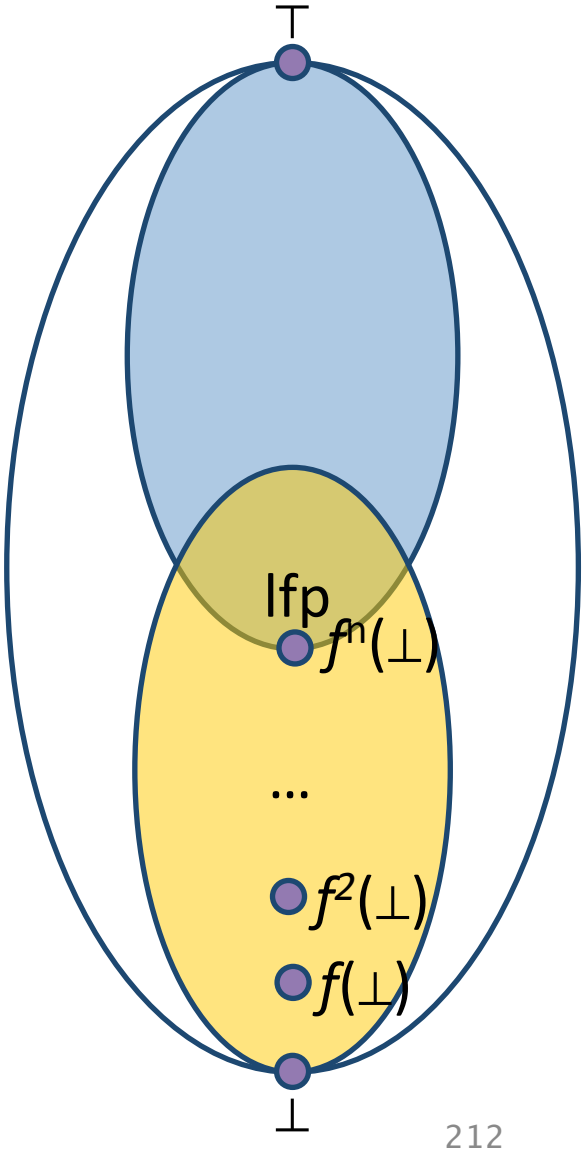
- Kleene's fixed point theorem gives a constructive method for computing the lfp

Mathematical definition

$$\text{lfp}(f) = \bigsqcup_{n \in N} f^n(\bot)$$

Algorithm

$d := \bot$
while $f(d) \neq d$ **do**
  $d := d \sqcup f(d)$
return d



Chaotic iteration

- Input:
  - A cpo $L = (D, \sqsubseteq, \sqcup, \bot)$ satisfying ACC
  - $L^n = L \times L \times \ldots \times L$
  - A monotone function $f : D^n \rightarrow D^n$
  - A system of equations $\{ X[i] \mid f(X) \mid 1 \leq i \leq n \}$
- Output: lfp(f)
- A worklist-based algorithm

```
for i:=1 to n do
   X[i] := ⊥
WL = {1,…,n}
while WL ≠ ∅ do
   j := pop WL // choose index non-deterministically
   N := F[i](X)
   if N ≠ X[i] then
      X[i] := N
      add all the indexes that directly depend on i to WL
      (X[j] depends on X[i] if F[j] contains X[i])
return X
```

Concrete semantics equations

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]      ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]      error("Unable to prove x == 1000!");
}
```

- R[0] = {**x**∈**Z**}
  R[1] = ⟦**x**:=**7**⟧
  R[2] = R[1] ∪ R[4]
  R[3] = R[2] ∩ {s | s(x) < 1000}
  R[4] = ⟦**x**:=**x+1**⟧ R[3]
  R[5] = R[2] ∩ {s | s(x) ≥ 1000}
  R[6] = R[5] ∩ {s | s(x) ≠ 1001}

Abstract semantics equations

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]       ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]       error("Unable to prove x == 1000!");
}
```

- $R[0] = \alpha(\{\mathbf{x} \in \mathbf{Z}\})$
  $R[1] = [\![\mathbf{x := 7}]\!]^{\#}$
  $R[2] = R[1] \sqcup R[4]$
  $R[3] = R[2] \sqcap \alpha(\{s \mid s(x) < 1000\})$
  $R[4] = [\![\mathbf{x := x+1}]\!]^{\#} R[3]$
  $R[5] = R[2] \sqcap \alpha(\{s \mid s(x) \geq 1000\})$
  $R[6] = R[5] \sqcap \alpha(\{s \mid s(x) \geq 1001\}) \sqcup R[5] \sqcap \alpha(\{s \mid s(x) \leq 999\})$

Abstract semantics equations

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]      ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]      error("Unable to prove x == 1000!");
}
```

- R[0] = ⊤
  R[1] = [7,7]
  R[2] = R[1] ⊔ R[4]
  R[3] = R[2] ⊓ [-∞,999]
  R[4] = R[3] + [1,1]
  R[5] = R[2] ⊓ [1000,+∞]
  R[6] = R[5] ⊓ [999,+∞] ⊔ R[5] ⊓ [1001,+∞]

Too many iterations to converge

```
Iteration 3981: processing V[8] = Interval[x==1000](V[6]) // if x == 1000 goto return
              V[8] : false
              V[6] : and(x=1000)
              V[8]' : and(x=1000)
              Adding [V[12] = Join_IntervalDomain(V[8], V[10]) // return]
              workSet = {V[12]}
Iteration 3982: processing V[12] = Join_IntervalDomain(V[8], V[10]) // return
              V[12] : false
              V[8] : and(x=1000)
              V[10] : false
              V[12]' : and(x=1000)
              Adding [V[11] = V[12] // return]
              workSet = {V[11]}
Iteration 3983: processing V[11] = V[12] // return
              V[11] : false
              V[12] : and(x=1000)
              V[11]' : and(x=1000)
              Adding []
Reached fixed-point after 3983 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[5] : and(7<=x<=999)
  V[6] : and(x=1000)
  V[8] : and(x=1000)
  V[9] : false
  V[10] : false
  V[12] : and(x=1000)
  V[11] : and(x=1000)
}
0 possible errors found.
Writing to sootOutput\IntervalExample.jimple
Soot finished on Wed Jun 12 06:24:14 IDT 2013
Soot has run for 0 min. 1 sec.
```

How many iterations for this one?

```
public void loopExample2(int y) {
    int x = 7;
    if (x < y) {
        while (x < y) {
            ++x;
        }

        if (x != y)
            error("Unable to prove x = y!");
    }
}
```

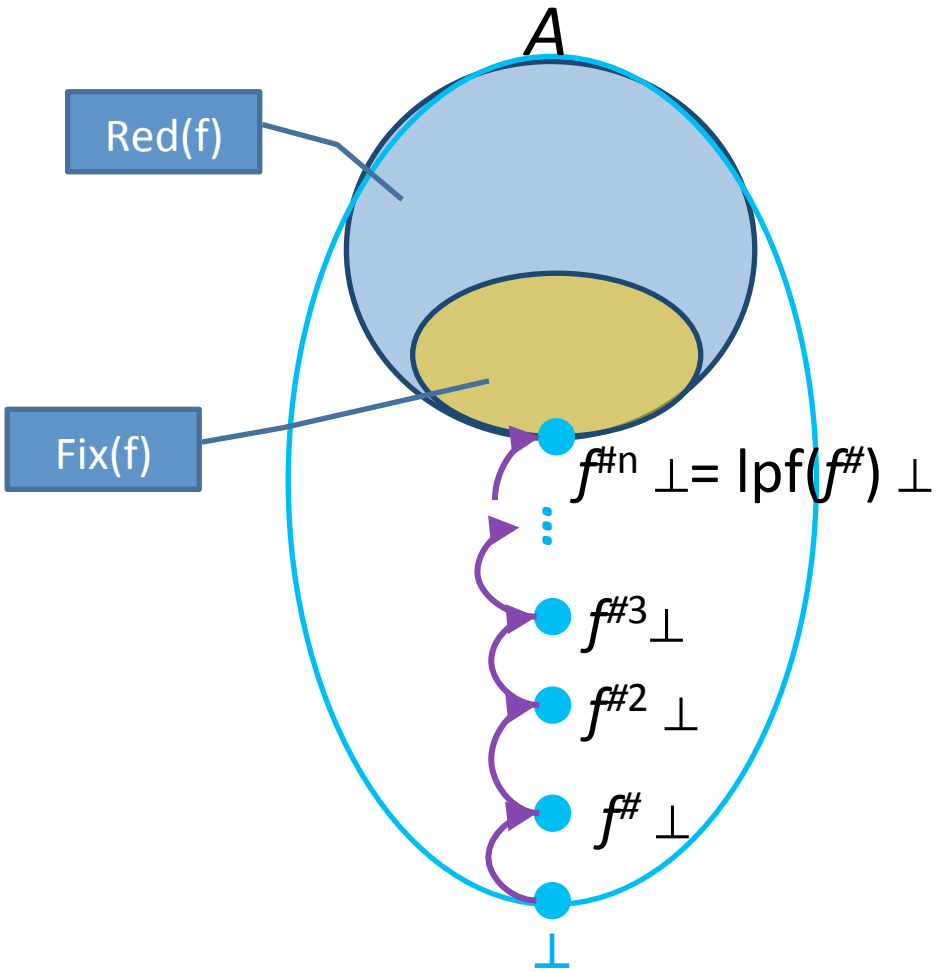
Widening

- Introduce a new binary operator to ensure termination
  - A kind of extrapolation
- Enables static analysis to use infinite height lattices
  - Dynamically adapts to given program
- Tricky to design
- Precision less predictable then with finite-height domains (widening non-monotone)

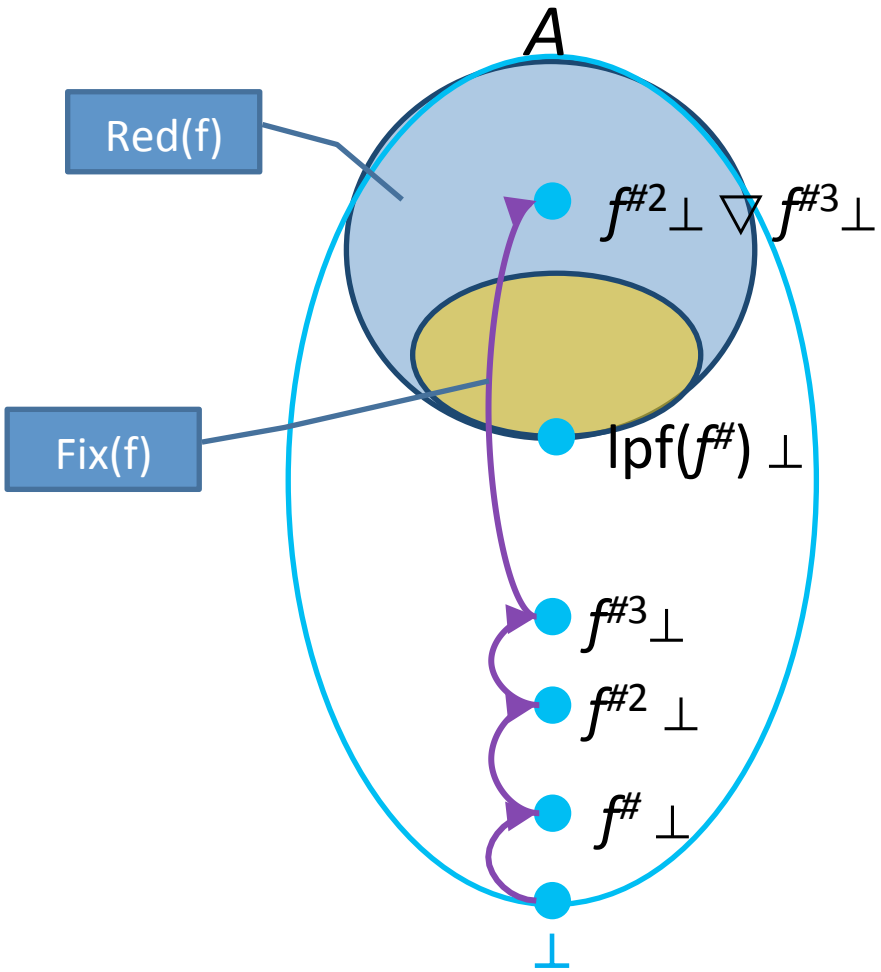
Formal definition

- For all elements $d_1 \sqcup d_2 \sqsubseteq d_1 \bigtriangledown d_2$
- For all ascending chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ the following sequence is finite
  - $y_0 = d_0$
  - $y_{i+1} = y_i \bigtriangledown d_{i+1}$
- For a monotone function $f : D \rightarrow D$ define
  - $x_0 = \bot$
  - $x_{i+1} = x_i \bigtriangledown f(x_i)$
- Theorem:
  - There exits k such that $x_{k+1} = x_k$
  - $x_k \in \text{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$

Analysis with finite-height lattice



Analysis with widening



Widening for Intervals Analysis

- $\perp \triangledown [c, d] = [c, d]$
- $[a, b] \triangledown [c, d] = [$
  if $a \leq c$
  then a
  else $-\infty$,
  if $b \geq d$
  then b
  else ∞

Semantic equations with widening

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]       ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]       error("Unable to prove x == 1000!");
}
```

- $R[0] = \top$
  $R[1] = [7,7]$
  $R[2] = R[1] \sqcup R[4]$
  $R[2.1] = R[2.1] \triangledown R[2]$
  $R[3] = R[2.1] \sqcap [-\infty,999]$
  $R[4] = R[3] + [1,1]$
  $R[5] = R[2] \sqcap [1001,+\infty]$
  $R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Choosing analysis with widening

```
/**
 * Adds the Interval analysis transform to Soot.
 *
 * @author romanm
 */
public class IntervalMain {
    public static void main(String[] args) {
        PackManager
                .v()
                .getPack("jtp")
                .add(new Transform("jtp.IntervalAnalysis",
                        new IntervalAnalysis()));
        soot.Main.main(args);
    }

    public static class IntervalAnalysis extends BaseAnalysis<IntervalState> {
        public IntervalAnalysis() {
            super(new IntervalDomain());
            useWidening(true);
        }
    }
}
```

Enable widening

Non monotonicity of widening

- $[0,1] \triangledown [0,2] = ?$
- $[0,2] \triangledown [0,2] = ?$

Non monotonicity of widening

- $[0,1] \triangledown [0,2] = [0, \infty]$
- $[0,2] \triangledown [0,2] = [0,2]$

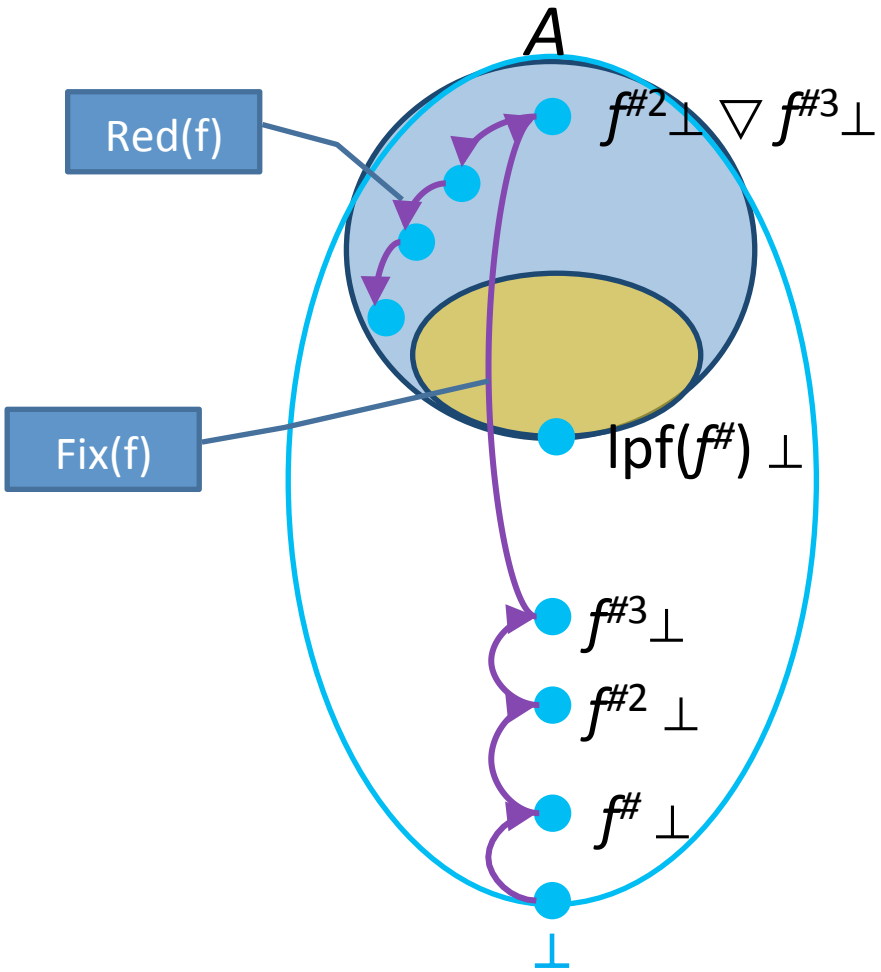
Analysis results with widening

```
Analyzing method loopExample
----------------------------------------
Solving the following equation system =
V[0] = true // this := @this: IntervalExample
V[1] = AssignTopTransformer(V[0]) // this := @this: IntervalExample
V[2] = AssignConstantToVarTransformer(V[1]) // x = 7
V[3] = V[2] // goto [?= (branch)]
V[4] = AssignAddExprToVarTransformer(V[5]) // x = x + 1
V[7] = JoinLoop_IntervalDomain(V[3], V[4]) // if x < 1000 goto x = x + 1
V[8] = IntervalDomain[Widening|Narrowing](V[8], V[7]) // if x < 1000 goto x = x + 1
V[5] = Interval[x<1000](V[8]) // if x < 1000 goto x = x + 1
V[6] = Interval[x>=1000](V[8]) // if x < 1000 goto x = x + 1
V[9] = Interval[x==1000](V[6]) // if x == 1000 goto return
V[10] = Interval[x!=1000](V[6]) // if x == 1000 goto return
V[11] = V[10] // specialinvoke this.<IntervalExample: void error(java.lang.String)>("Unable to prove x == 1000!")
V[13] = Join_IntervalDomain(V[9], V[11]) // return
V[12] = V[13] // return
```

```
Reached fixed-point after 23 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(x>=7)
  V[5] : and(7<=x<=999)
  V[6] : and(x>=1000)
  V[9] : and(x=1000)
  V[10] : and(x>=1001)
  V[11] : and(x>=1001)
  V[13] : and(x>=1000)
  V[12] : and(x>=1000)
}
```

Did we prove it?

Analysis with narrowing



Formal definition of narrowing

- Improves the result of widening
- $y \sqsubseteq x \Rightarrow y \sqsubseteq (x \triangle y) \sqsubseteq x$
- For all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$ the following sequence is finite
  - $y_0 = x_0$
  - $y_{i+1} = y_i \triangle x_{i+1}$
- For a monotone function f: $D \rightarrow D$ and $x_k \in \mathrm{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$ define
  - $y_0 = x$
  - $y_{i+1} = y_i \triangle f(y_i)$
- Theorem:
  - There exits k such that $y_{k+1} = y_k$
  - $y_k \in \mathrm{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$

Narrowing for Interval Analysis

- $[a, b] \triangle \perp = [a, b]$
- $[a, b] \triangle [c, d] = [$

  if $a = -\infty$

  then c

  else a,

  if $b = \infty$

  then d

  else b

  $]$

Semantic equations with narrowing

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]      ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]      error("Unable to prove x == 1000!");
}
```

- $R[0] = \top$
  $R[1] = [7,7]$
  $R[2] = R[1] \sqcup R[4]$
  $R[2.1] = R[2.1] \triangle R[2]$
  $R[3] = R[2.1] \sqcap [-\infty,999]$
  $R[4] = R[3]+[1,1]$
  $R[5] = R[2]^{\#} \sqcap [1000,+\infty]$
  $R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Analysis with widening/narrowing

- Two phases
  - Phase 1: analyze with widening until converging
  - Phase 2: use values to analyze with narrowing

```
public void loopExample() {
    int x = 7;
    while (x < 1000) {
        ++x;
    }
    if (!(x == 1000))
        error("Unable to prove x == 1000!");
}
```

Phase 1:

$R[0] = \top$

$R[1] = [7,7]$

$R[2] = R[1] \sqcup R[4]$

$R[2.1] = R[2.1] \triangledown R[2]$

$R[3] = R[2.1] \sqcap [-\infty,999]$

$R[4] = R[3] + [1,1]$

$R[5] = R[2] \sqcap [1001,+\infty]$

$R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Phase 2:

$R[0] = \top$

$R[1] = [7,7]$

$R[2] = R[1] \sqcup R[4]$

$R[2.1] = R[2.1] \triangle R[2]$

$R[3] = R[2.1] \sqcap [-\infty,999]$

$R[4] = R[3]+[1,1]$

$R[5] = R[2]^{\#} \sqcap [1000,+\infty]$

$R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Analysis with widening/narrowing

```
Reached fixed-point after 23 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(x>=7)
  V[5] : and(7<=x<=999)
  V[6] : and(x>=1000)
  V[9] : and(x=1000)
  V[10] : and(x>=1001)
  V[11] : and(x>=1001)
  V[13] : and(x>=1000)
  V[12] : and(x>=1000)
}

Starting chaotic iteration: narrowing phase...
              workSet = {V[0], V[1], V[2], V[3], V[4], V[7], V[8], V[5], V[6], V[9], V[10], V[11], V[13], V[12]}
Iteration 24: processing V[0] = true // this := @this: IntervalExample
              V[0] : true
              V[0]' : true
              workSet = {V[12], V[1], V[2], V[3], V[4], V[7], V[8], V[5], V[6], V[9], V[10], V[11], V[13]}
```

Analysis results widening/narrowing

```
Iteration 44: processing V[1] = AssignTopTransformer(V[0]) // this := @this: IntervalExample
              V[1] : true
              V[0] : true
              V[1]' : true
Reached fixed-point after 44 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(7<=x<=1000)
  V[5] : and(7<=x<=999)
  V[6] : and(x=1000)
  V[9] : and(x=1000)
  V[10] : false
  V[11] : false
  V[13] : and(x=1000)
  V[12] : and(x=1000)
}
0 possible errors found.
Writing to sootOutput\IntervalExample.jimple
Soot finished on Wed Jun 12 06:47:24 IDT 2013
Soot has run for 0 min. 0 sec.
```