Program Analysis and Verification

0368-4479

Noam Rinetzky

Lecture 8: Abstract Interpretation

Slides credit: Roman Manevich, Mooly Sagiv, Eran Yahav

Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis

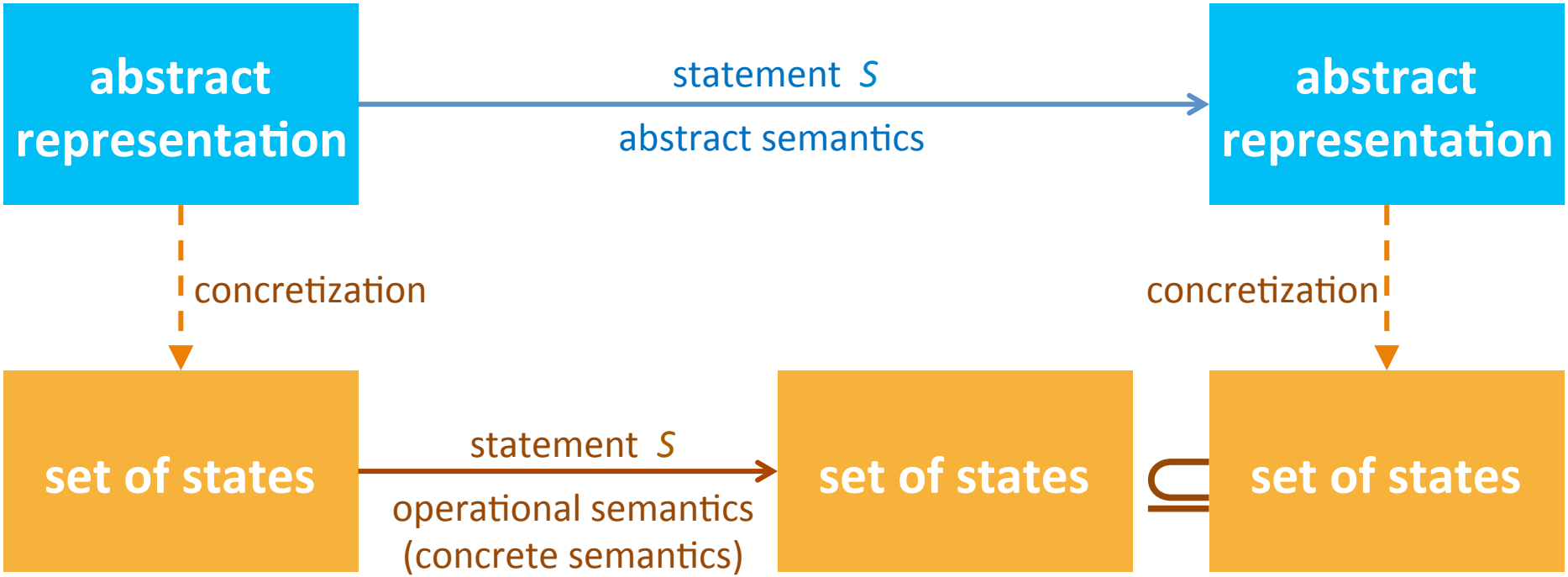
Order-Related Terminology

- Preorder
- Partial Order
- Pointed Posets
- Join/Meet
- Complete lattices

Complete lattices

- A complete lattice $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is
- A set of elements D
- A partial order $x \sqsubseteq y$
- A join operator $\sqcup$
- A meet operator $\sqcap$
- A bottom element
  $\bot = \sqcup \varnothing = \sqcap D$
- A top element
  $\top = \sqcup D = \sqcap \varnothing$

Abstract (conservative) interpretation



Domain Theory

- Monotone functions
- Chains
- Complete partial orders (CPO)
  - Every chain has a LUB
- Pointed CPOs
- Constructing CPOs
  - Cartesian product, relational product, disjunctive completion …

Continuity

- A monotonic function maps a chain of inputs into a chain of outputs:
  $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \Rightarrow f(x_0) \sqsubseteq f(x_1) \sqsubseteq \ldots$
- It is always true that:
  $\bigsqcup_i \langle f(x_i) \rangle \sqsubseteq f(\bigsqcup_i \langle x_i \rangle)$
- But
  $f(\bigsqcup_i \langle x_i \rangle) \sqsubseteq \bigsqcup_i \langle f(x_i) \rangle$
  is not always true

Fixed Points

- Solve equation:

$$W(\sigma) = \begin{cases} W(S[\![s]\!]\ \sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)= \perp \end{cases}$$

where $W: \sum_{\perp} \rightarrow \sum_{\perp}$; $W = S[\![\text{while b do S}]\!]$

Fixed Points

- Solve equation:

$$W(\sigma) = \begin{cases} W(S[\![s]\!]\,\sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

where $W: \sum_{\perp} \rightarrow \sum_{\perp}$; $W = S[\![\text{while b do S}]\!]$

- Alternatively, W = F(W) where:

$$F(W) = \lambda\sigma. \begin{cases} W(S[\![s]\!]\,\sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

Fixed Point (cont)

- Thus we are looking for a solution for W = F(W)
  - a fixed point of F
- Typically there are many fixed points
- We may argue that W ought to be continuous $W \in [\sum_{\perp} \rightarrow \sum_{\perp}]$
- Cut the number of solutions
- We will see how to find the least fixed point for such an equation provided that F itself is continuous

Fixed Point Theorem

- Define $F^k = \lambda x.\ F(\ F(\ldots\ F(\ x)\ldots))$ (F composed k times)
- If D is a pointed cpo and $F : D \rightarrow D$ is continuous, then
  - for any fixed-point x of F and $k \in N$
    $F^k(\bot) \sqsubseteq x$
  - The least of all fixed points is
    $\bigsqcup_k F^k(\bot)$
- Proof:
  i. By induction on k.
     - Base: $F^0(\bot) = \bot \sqsubseteq x$
     - Induction step: $F^{k+1}(\bot) = F(\ F^k(\bot)) \sqsubseteq F(\ x) = x$
  ii. It suffices to show that $\bigsqcup_k F^k(\bot)$ is a fixed-point
     - $F(\bigsqcup_k F^k(\bot)) = \bigsqcup_k F^{k+1}(\ \bot\ ) = \bigsqcup_k F^k(\bot)$

Fixed-Points (notes)

- If F is continuous on a pointed cpo, we know how to find the least fixed point
- All other fixed points can be regarded as refinements of the least one
  - They contain more information, they are more precise
  - In general, they are also more arbitrary

Fixed-Points (notes)

- If F is continuous on a pointed cpo, we know how to find the least fixed point
- All other fixed points can be regarded as refinements of the least one
  - They contain more information, they are more precise
  - In general, they are also more arbitrary
  - They also make less sense for our purposes

Complete Lattice

- Let $(D, \sqsubseteq)$ be a partial order

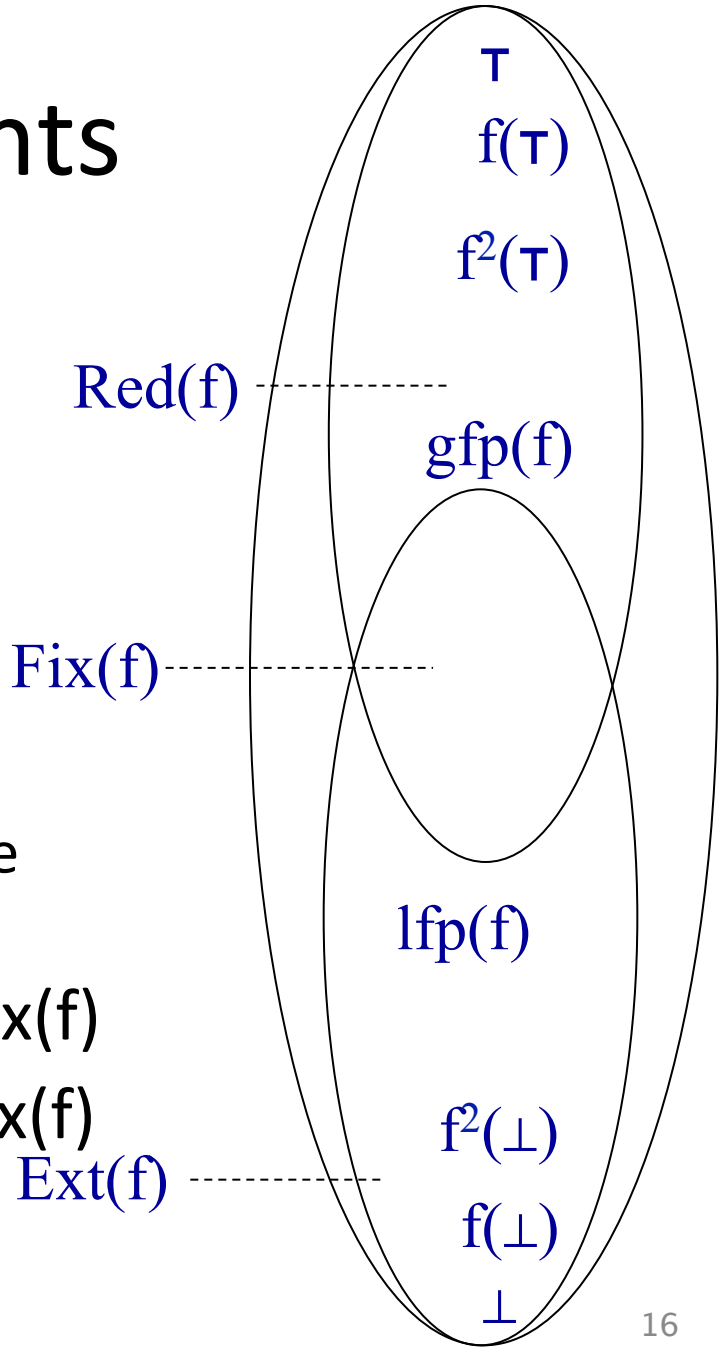
- D is a complete lattice if every subset has both greatest lower bounds and least upper bounds

Knaster-Tarski Theorem

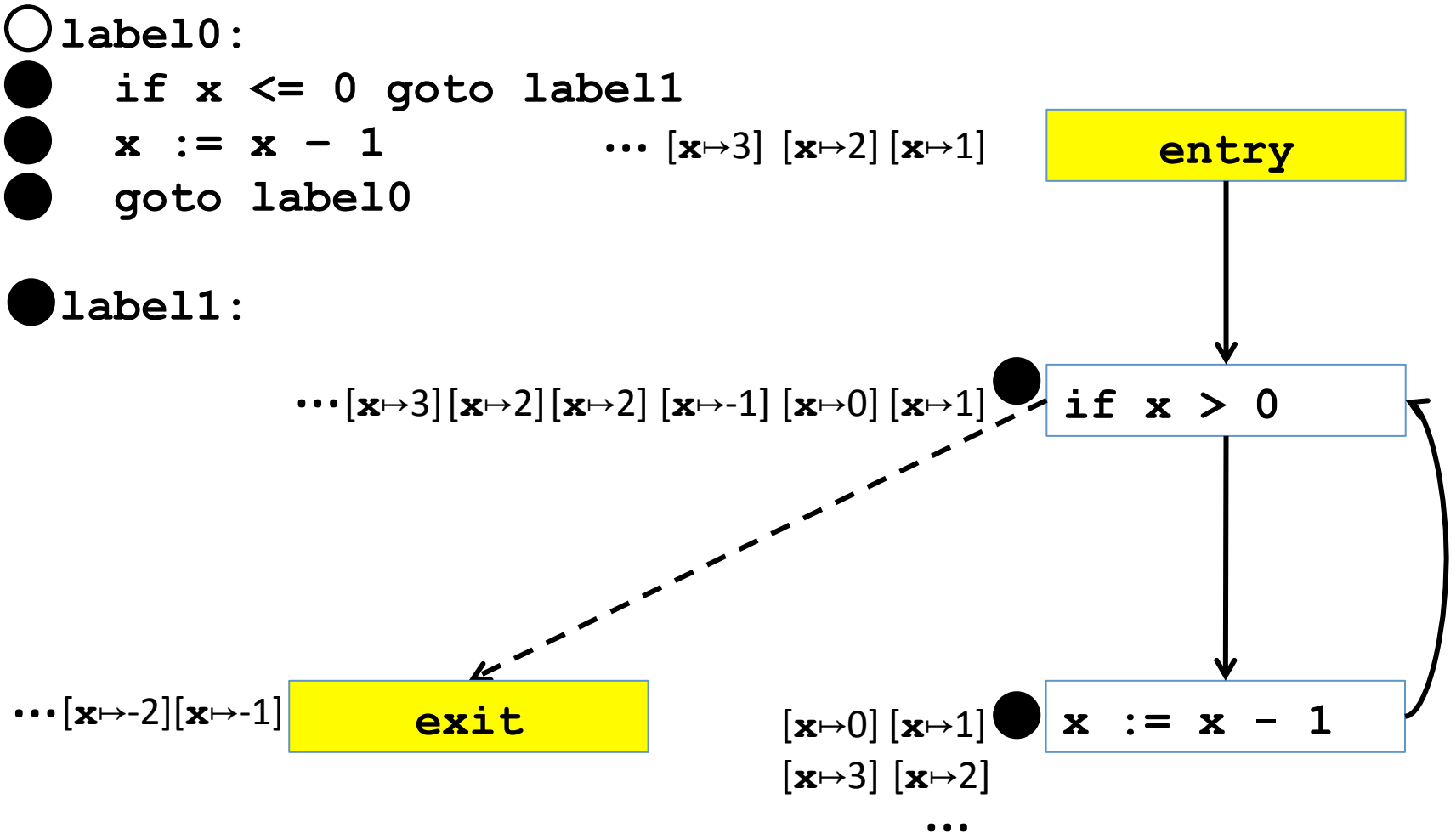
- Let f: L $\rightarrow$L be a monotonic function on a complete lattice L
- The least fixed point lfp(f) exists
  - lfp(f) = $\sqcap\{x \in L: f(x) \sqsubseteq x\}$

Fixed Points

- A monotone function f: L → L where (L, $\sqsubseteq$, $\sqcup$, $\sqcap$, $\bot$, $\top$) is a complete lattice
- Fix(f) = { l: l $\in$ L, f(l) = l}
- Red(f) = {l: l $\in$ L, f(l) $\sqsubseteq$ l}
- Ext(f) = {l: l $\in$ L, l $\sqsubseteq$ f(l)}
  - $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$
- Tarski's Theorem 1955: if f is monotone then:
  - lfp(f) = $\sqcap$ Fix(f) = $\sqcap$ Red(f) $\in$ Fix(f)
  - gfp(f) = $\sqcup$ Fix(f) = $\sqcup$ Ext(f) $\in$ Fix(f)



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*:

  ?

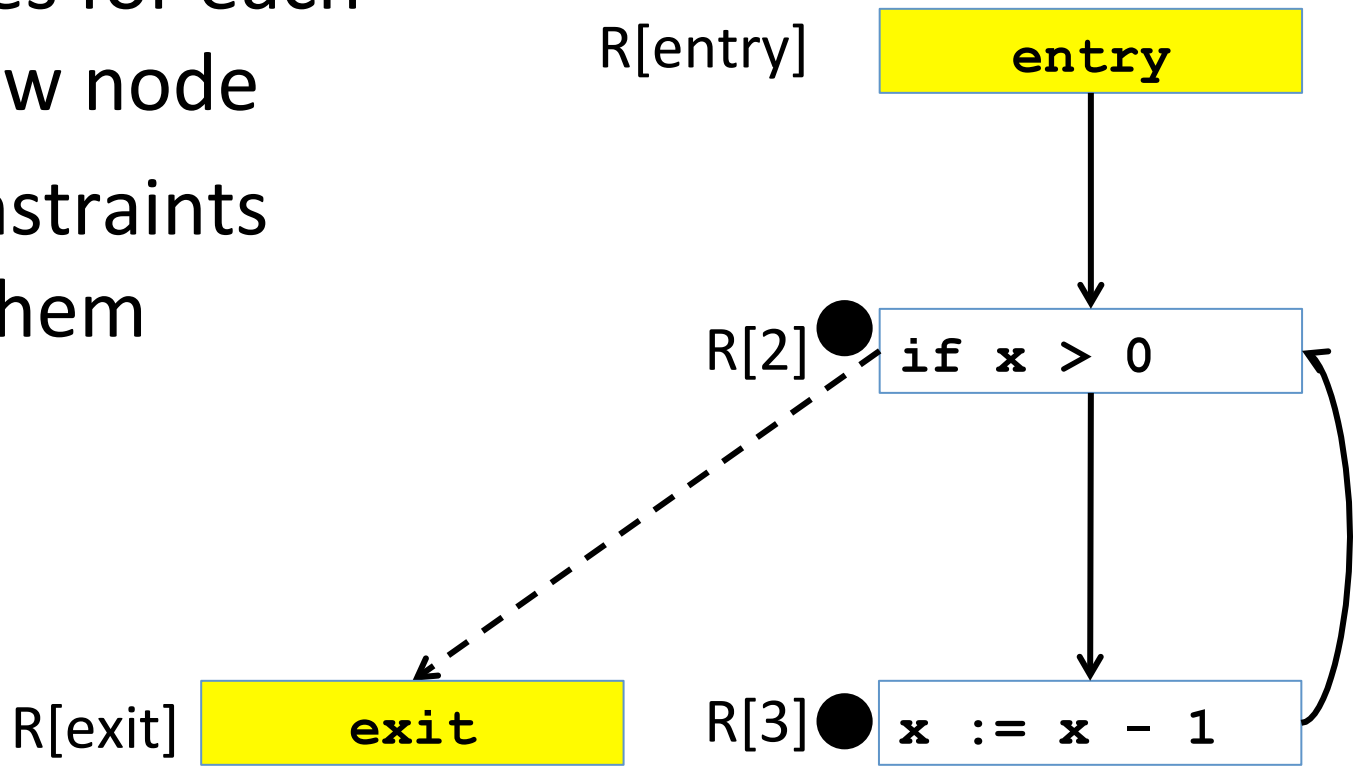
- We will use this lattice as a baseline for static analysis and define abstractions of its elements

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

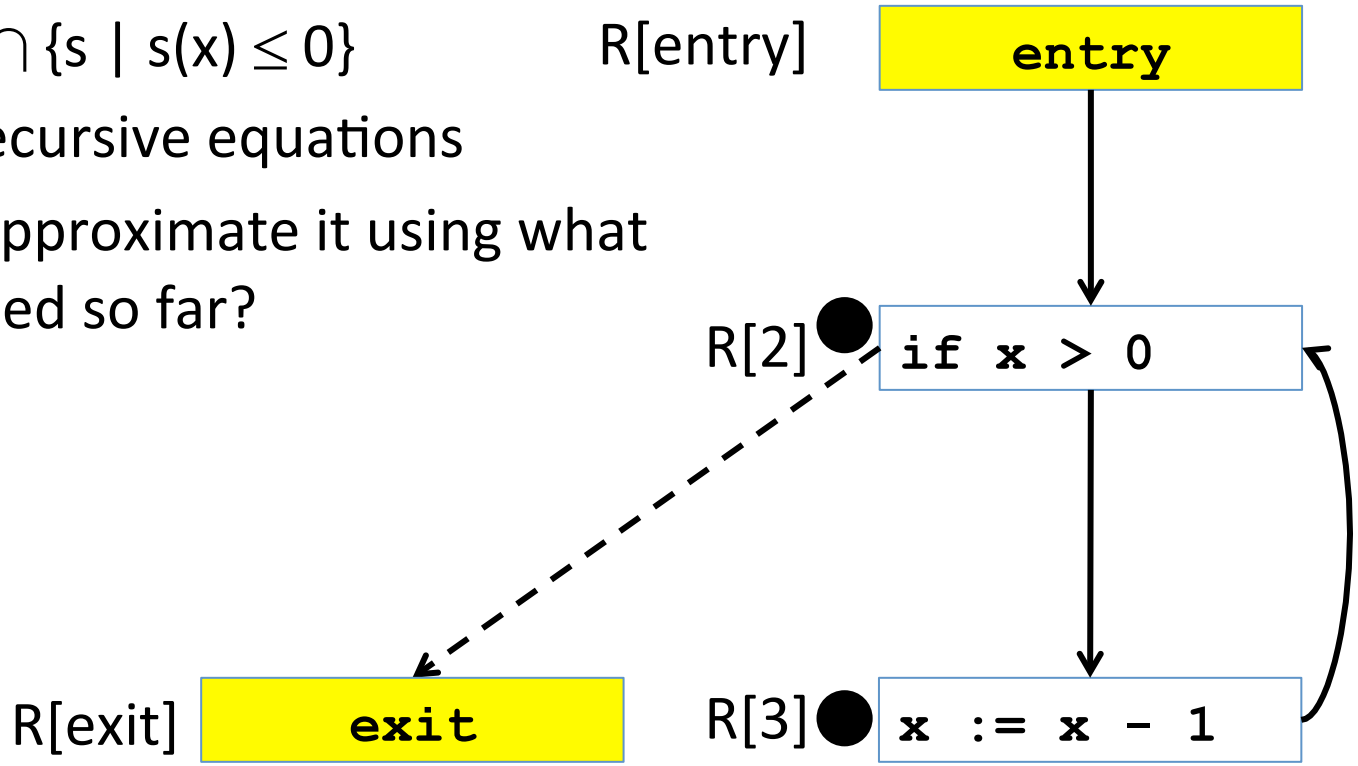
Equational definition of the semantics

- 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?

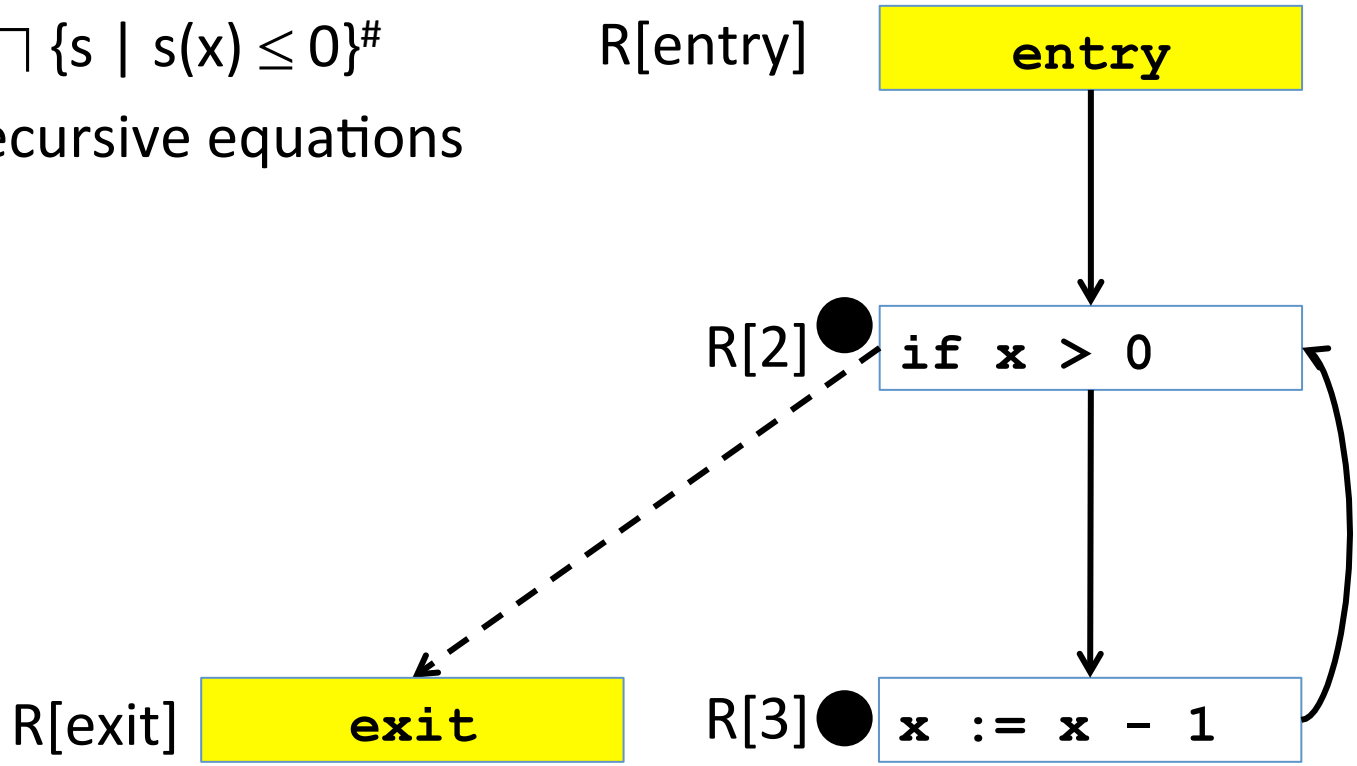


An abstract semantics

- $R[2] = R[entry] \sqcup [\![\texttt{x:=x-1}]\!]^{\#} R[3]$
- $R[3] = R[2] \sqcap \{s \mid s(x) > 0\}^{\#}$
- $R[exit] = R[2] \sqcap \{s \mid s(x) \leq 0\}^{\#}$
- A system of recursive equations

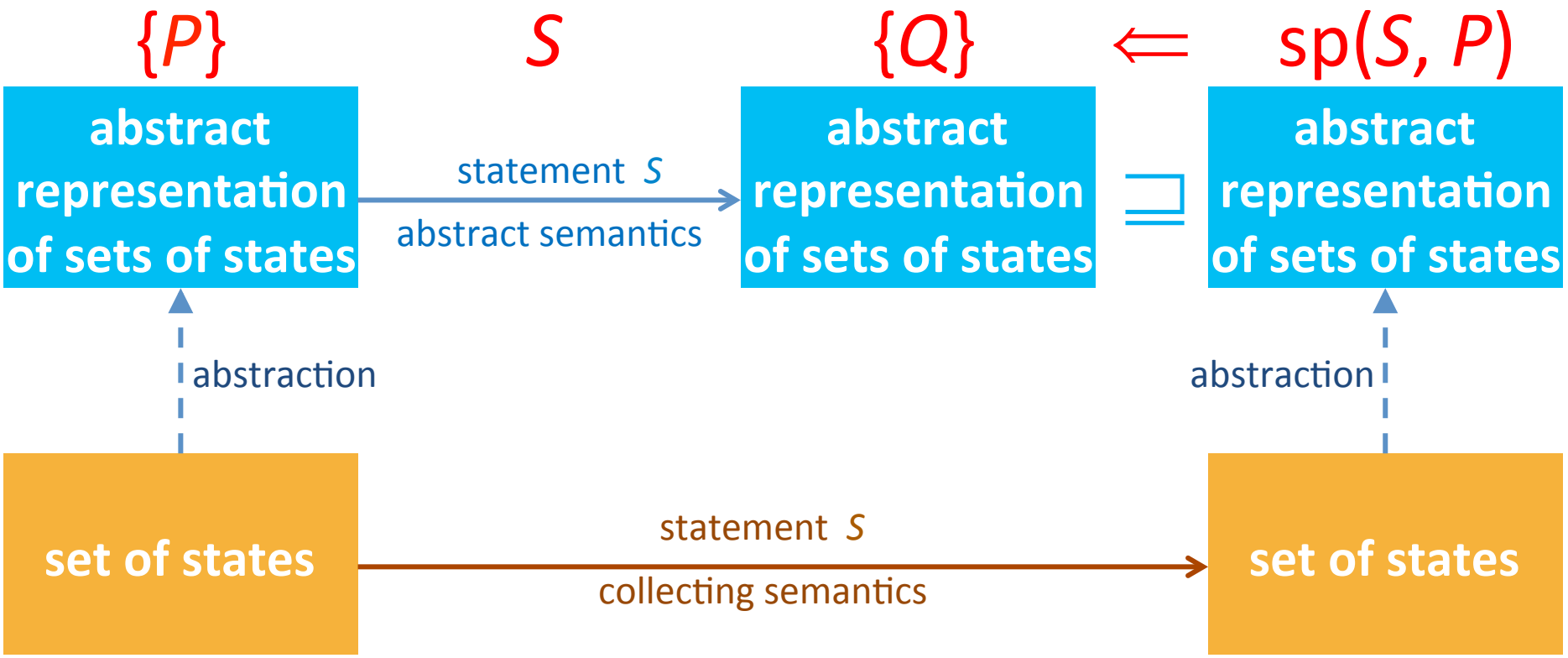
Abstract transformer for `x:=x-1`

Abstract representation of $\{s \mid s(x) < 0\}$

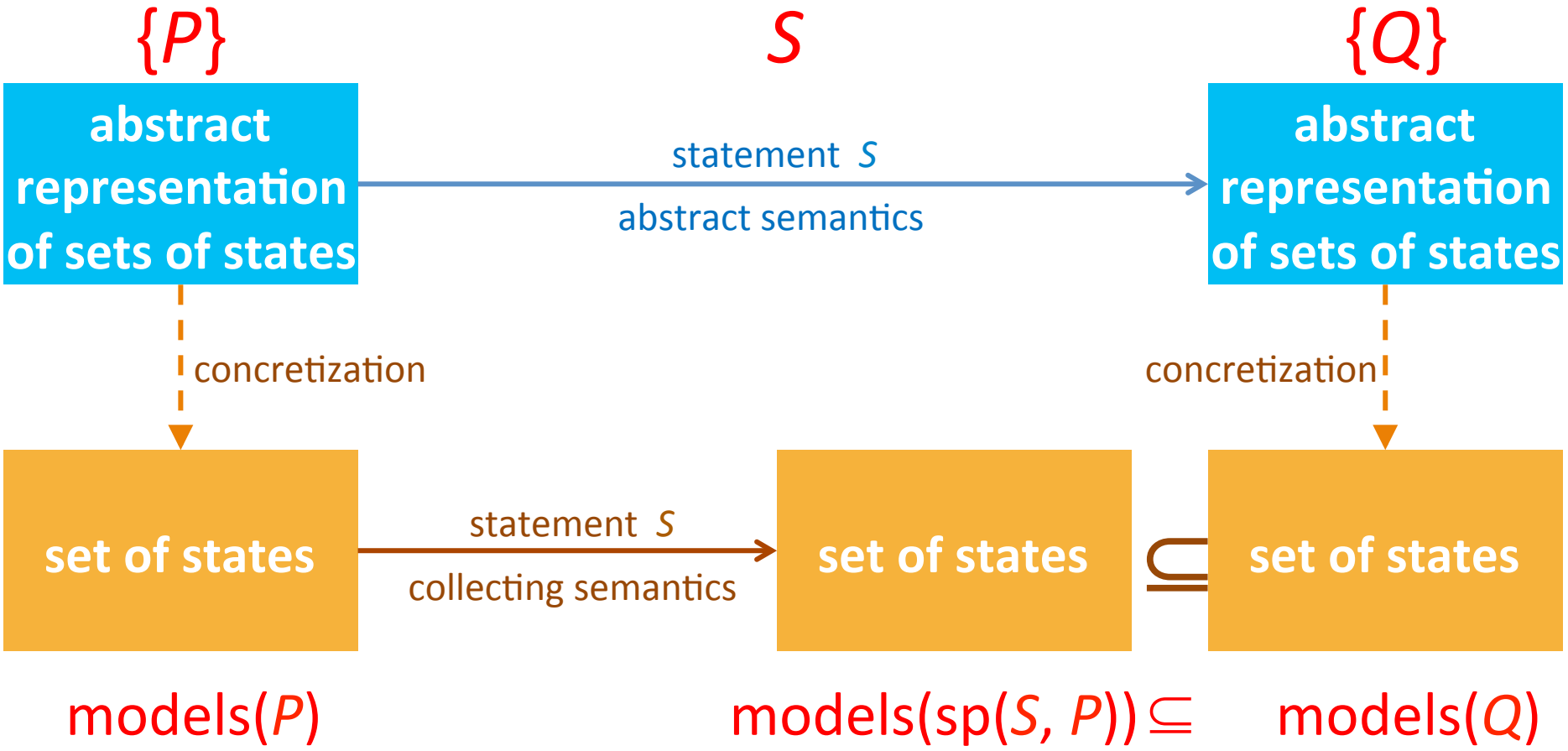


Abstract interpretation via abstraction

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

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

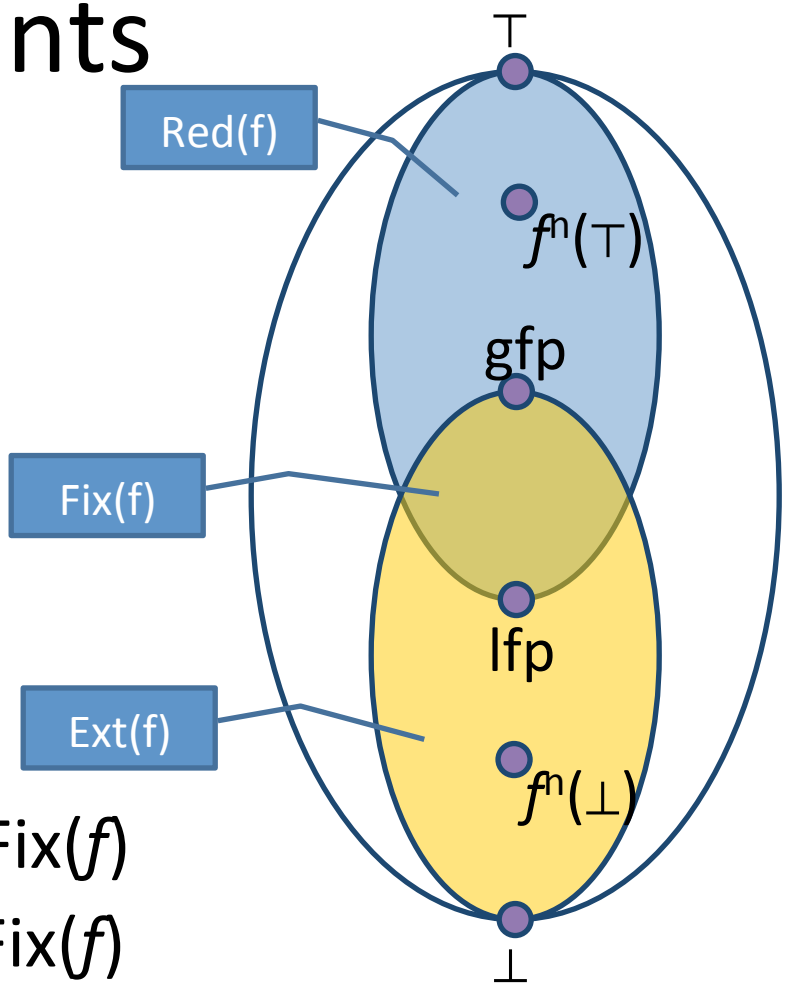
- Join: $f(X, Y) = X \sqcup Y$
  Prove it!
- For a set X and any function g
  $F(X) = \{ g(x) \mid x \in X \}$
  Prove it!
- Notice that the collecting semantics function is defined in terms of
  - Join (set union)
  - Semantic function for atomic statements lifted to sets of states

Extensive/reductive functions

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

Fixed-points

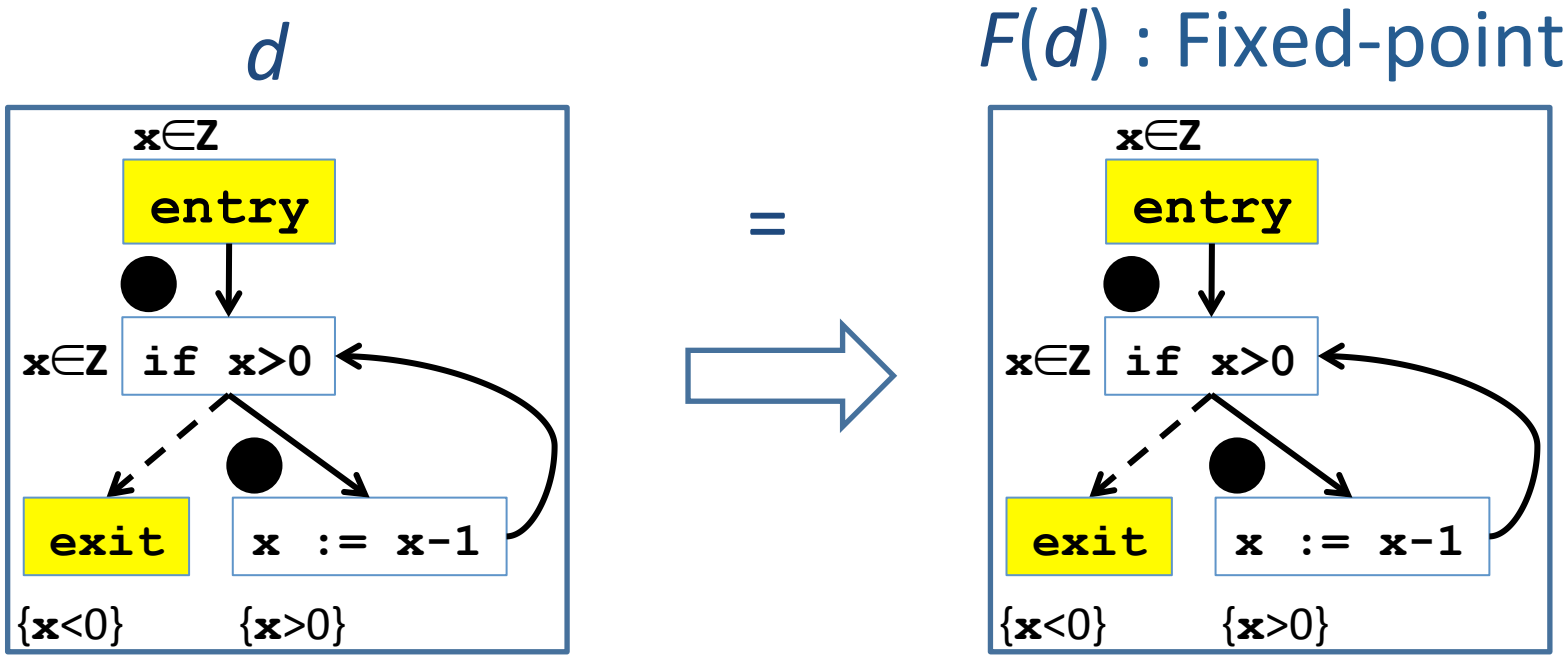
- $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- $f : D \rightarrow D$ **monotone**
- $\text{Fix}(f) = \{ d \mid f(d) = d \}$
- $\text{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\text{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\text{lfp}(f) = \sqcap\text{Fix}(f)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



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

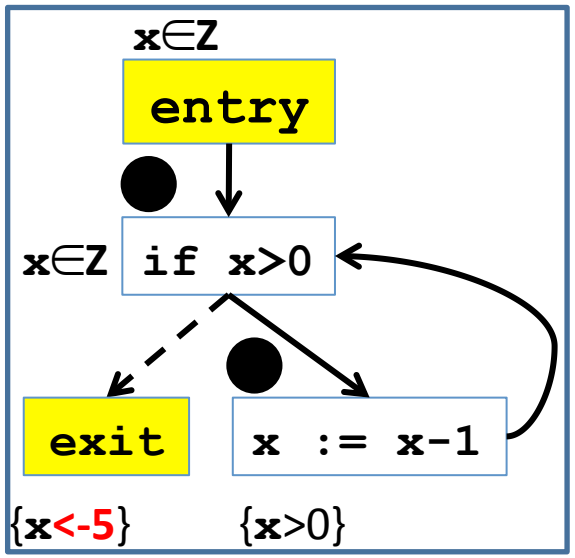
- R[0] = {**x**∈**Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]



Fixed point example for program

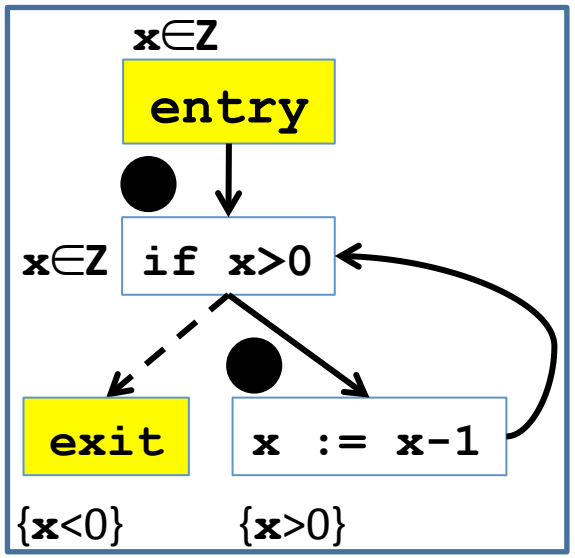
- R[0] = {**x∈Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]

d



⊑

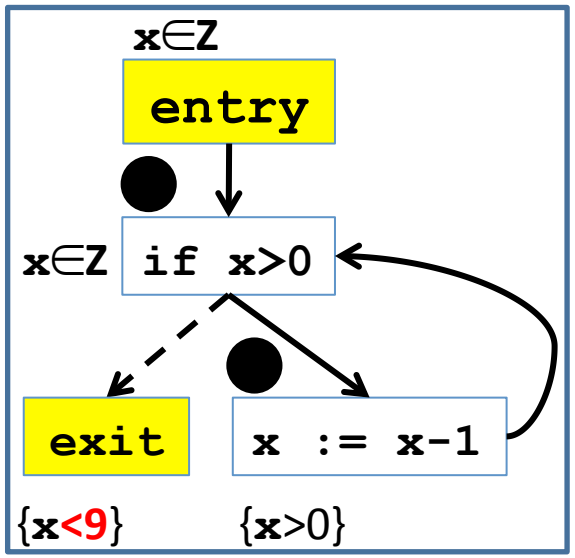
F(*d*) : pre Fixed-point



Fixed point example for program

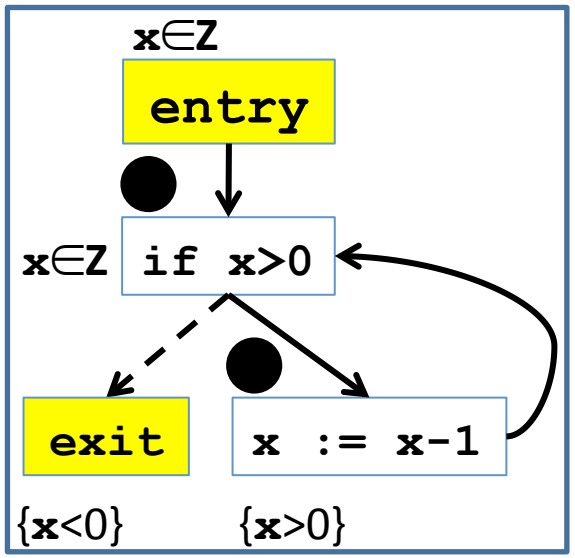
- R[0] = {**x∈Z**}
  R[1] = R[0] ∪ R[4]
  R[2] = R[1] ∩ {s | s(x) > 0}
  R[3] = R[1] ∩ {s | s(x) ≤ 0}
  R[4] = ⟦**x:=x-1**⟧ R[2]

d



⊒

F(*d*) : post Fixed-point



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 $f: D \rightarrow D$ then

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

- **Lemma:** Monotone functions on posets satisfying ACC are continuous
  Proof:

Resulting algorithm

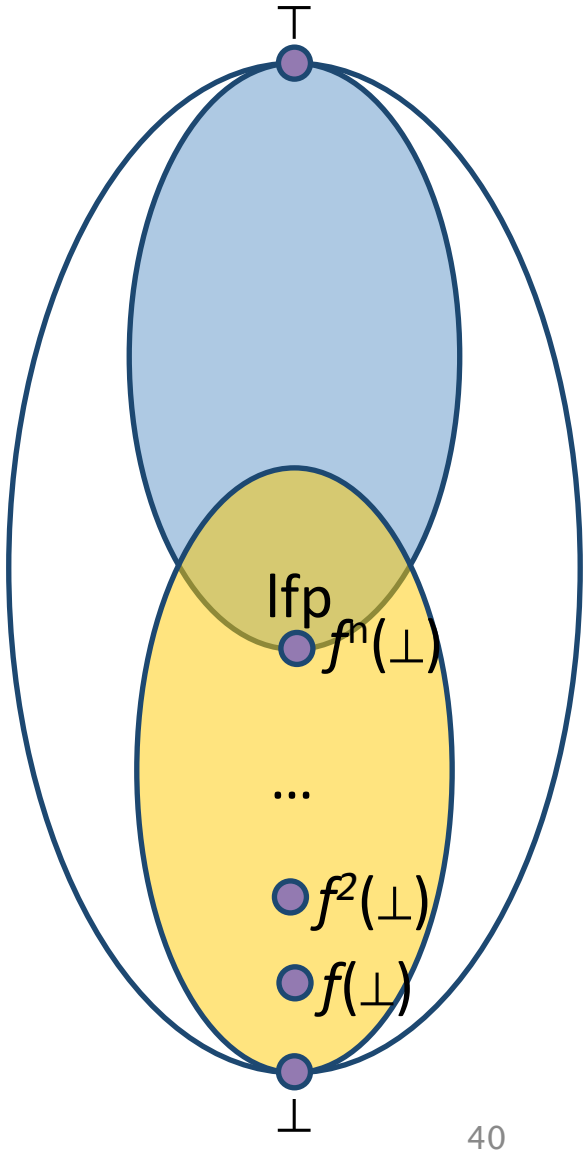
- Kleene's fixed point theorem gives a constructive method for computing the lfp

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

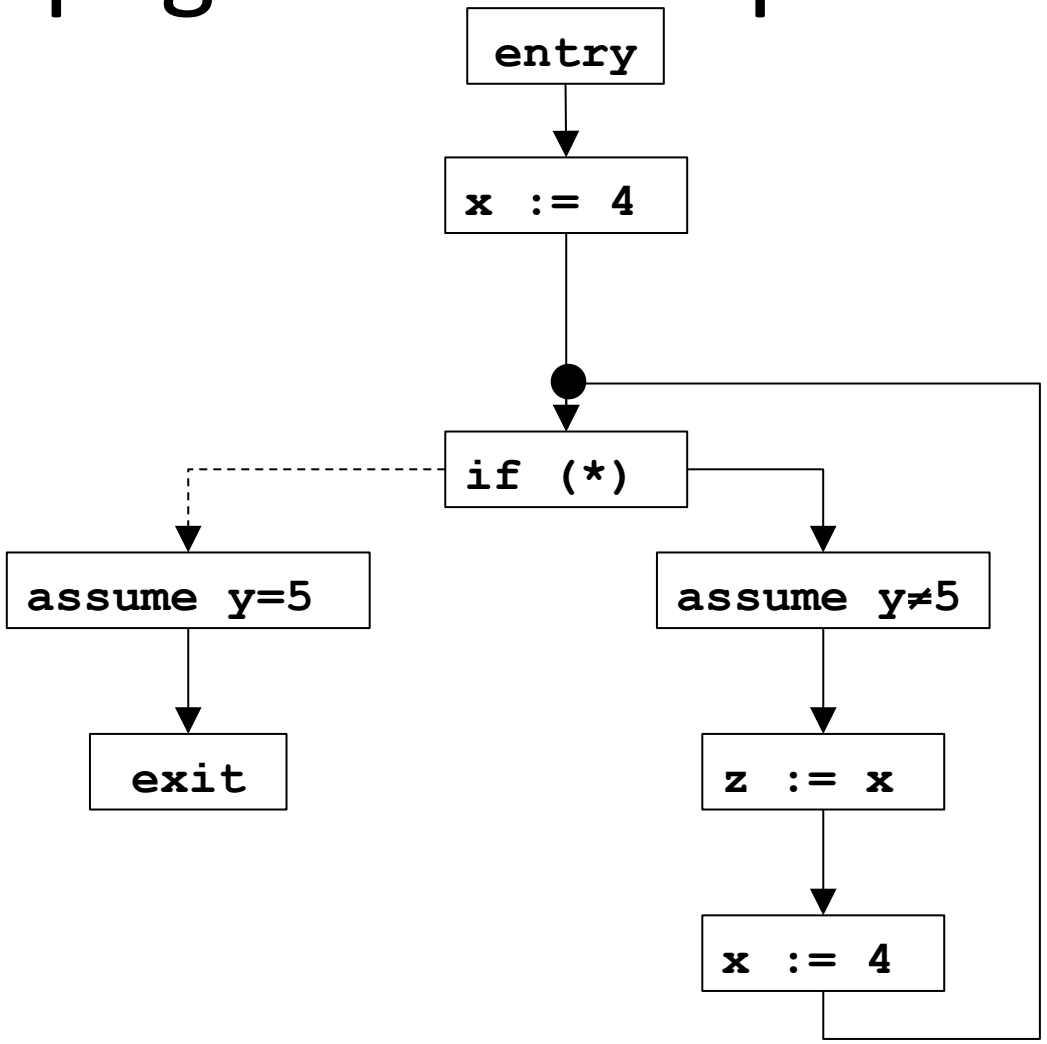
```
for i:=1 to n do
   X[i] := ⊥
WL = {1,…,n}
while WL ≠ ∅ do
   i := pop WL // choose index non-deterministically
   N := F[i](X)
   if N ≠ X[i] then
      X[i] := N
      add all the indexes that directly depend on i to WL
      (X[j] depends on X[i] if F[j] contains X[i])
return X
```

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

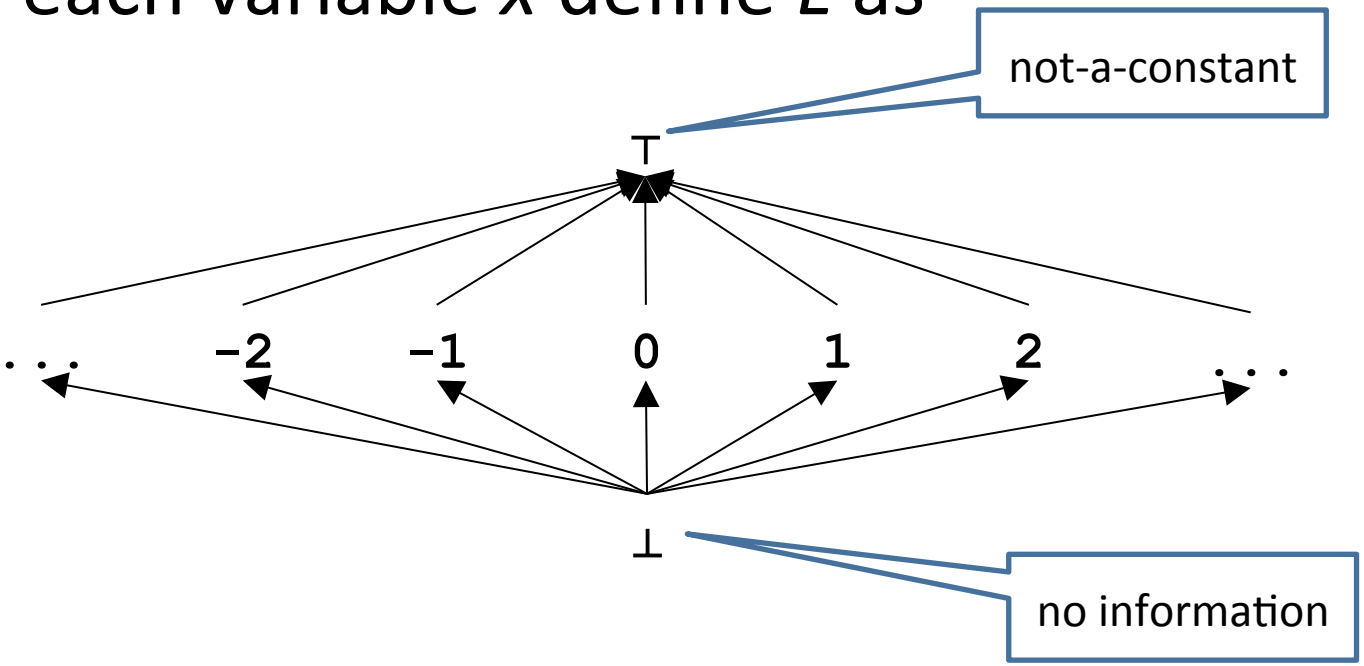
Constant propagation example

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



Constant propagation lattice

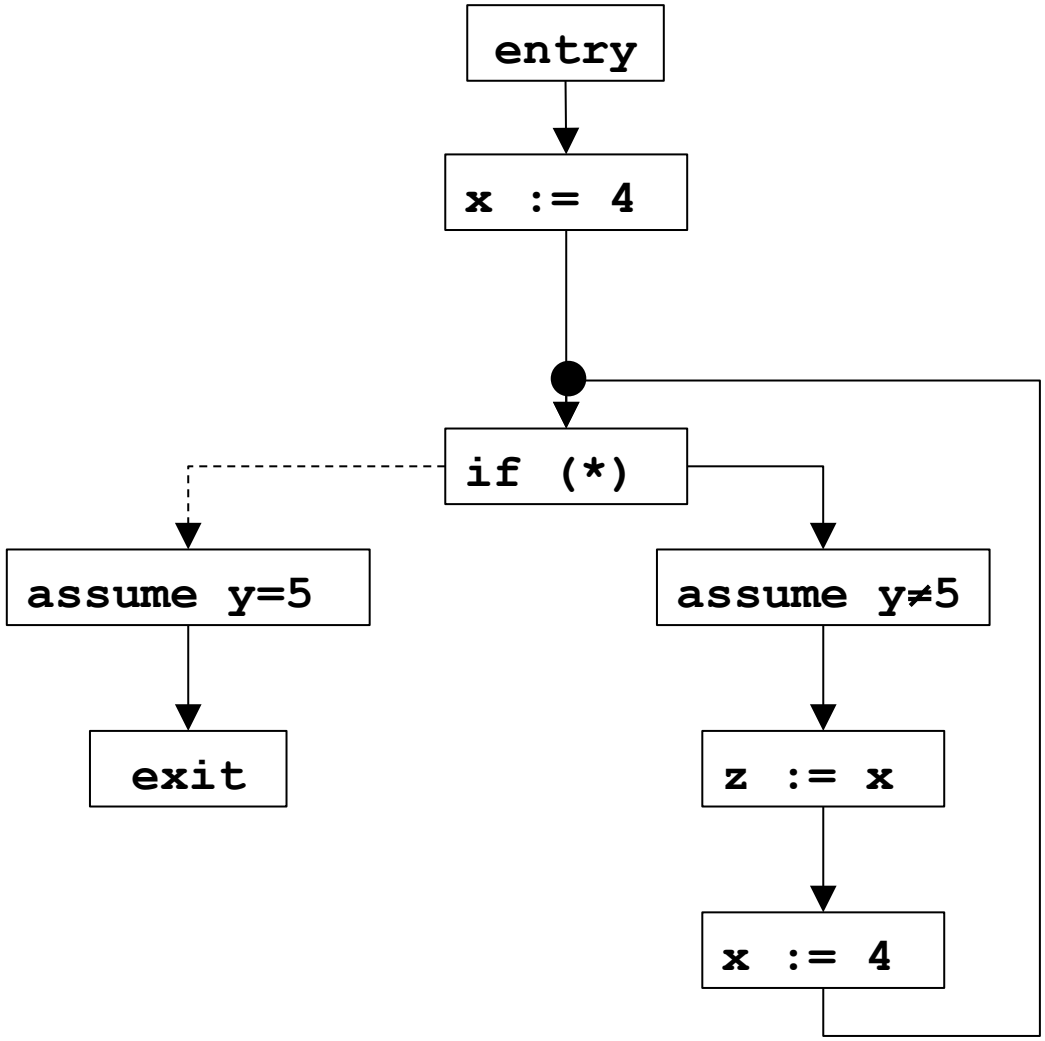
- For each variable *x* define *L* as



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

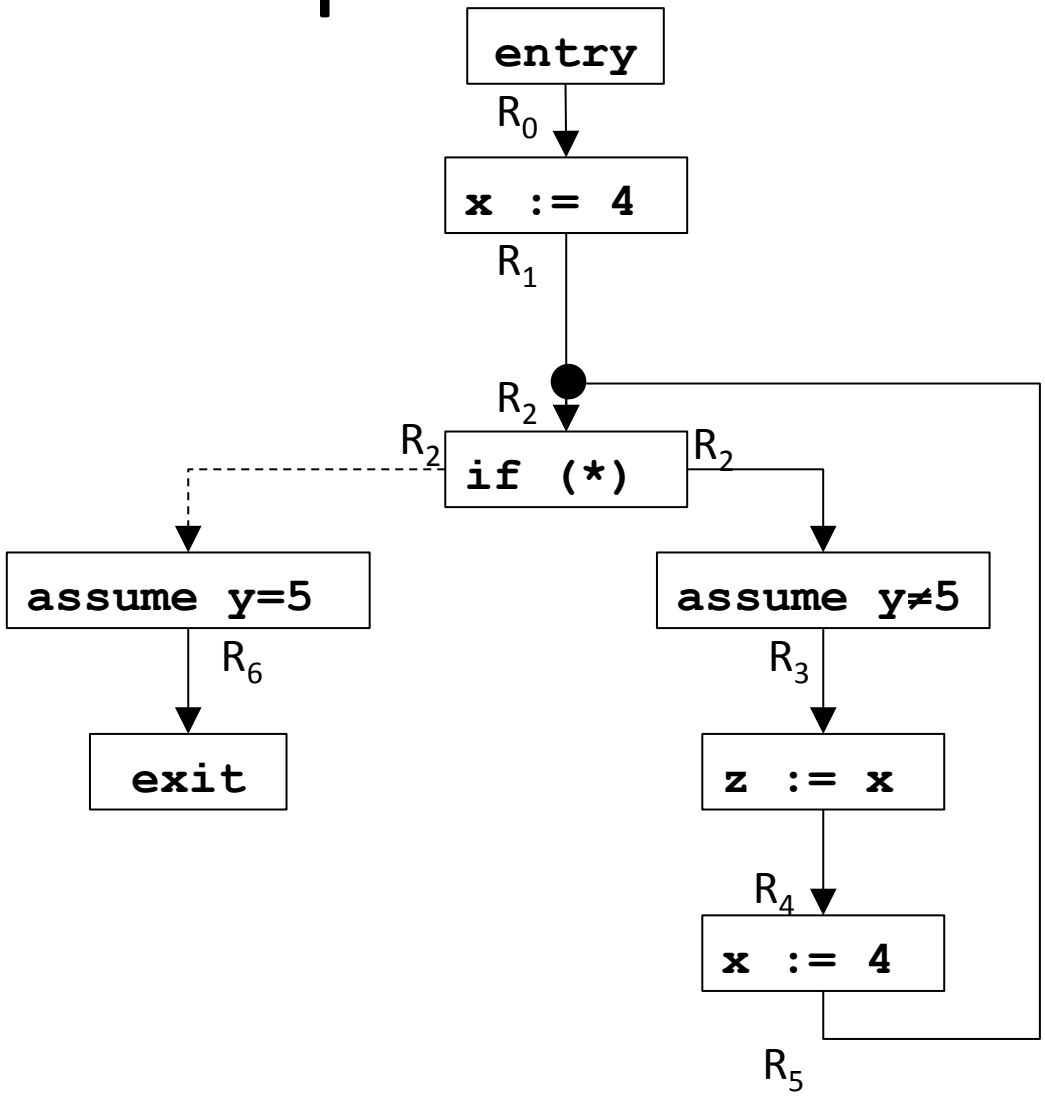
Write down variables

```
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**

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

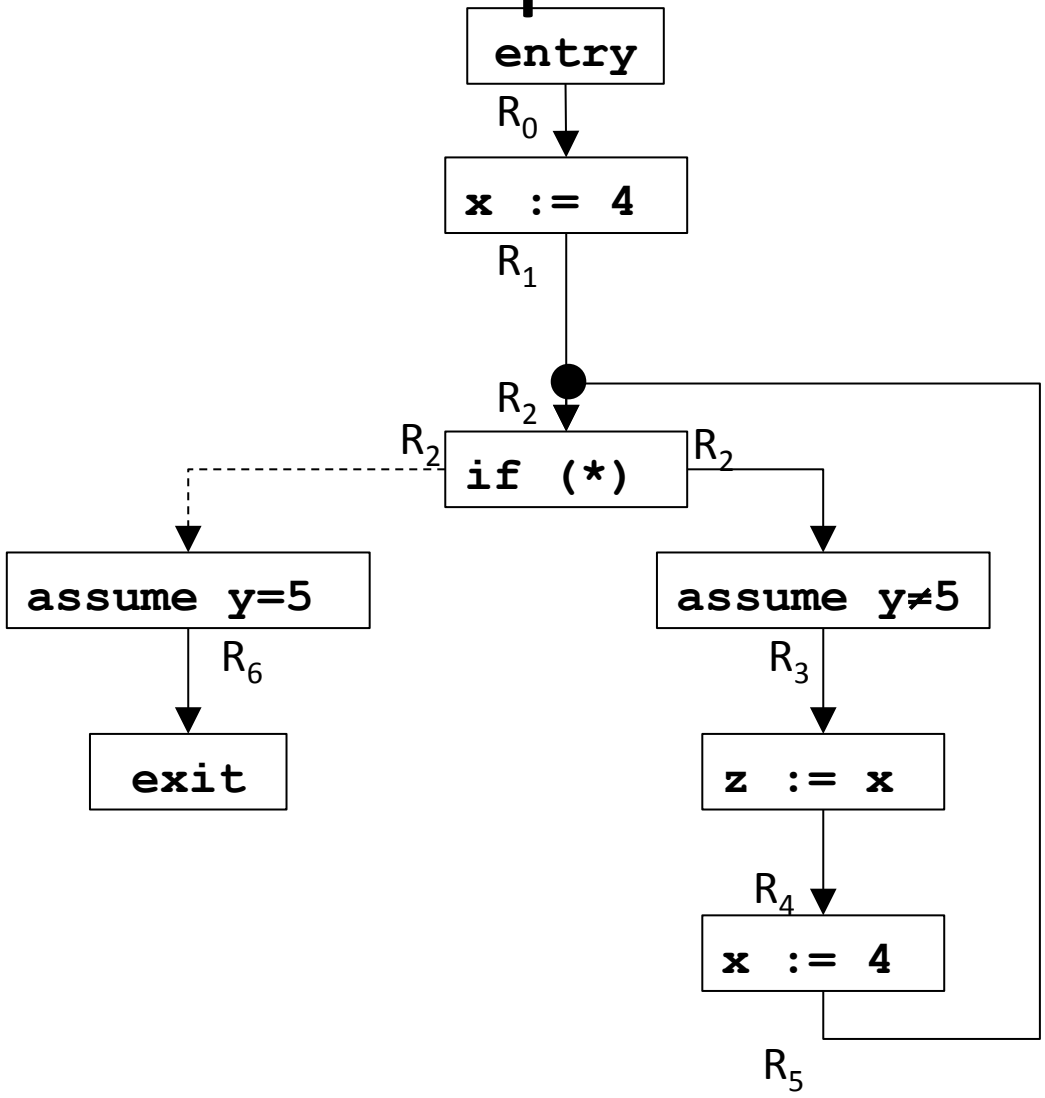
$R_2 = R_1 \cup R_5$

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

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

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

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



Constant propagation equations

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

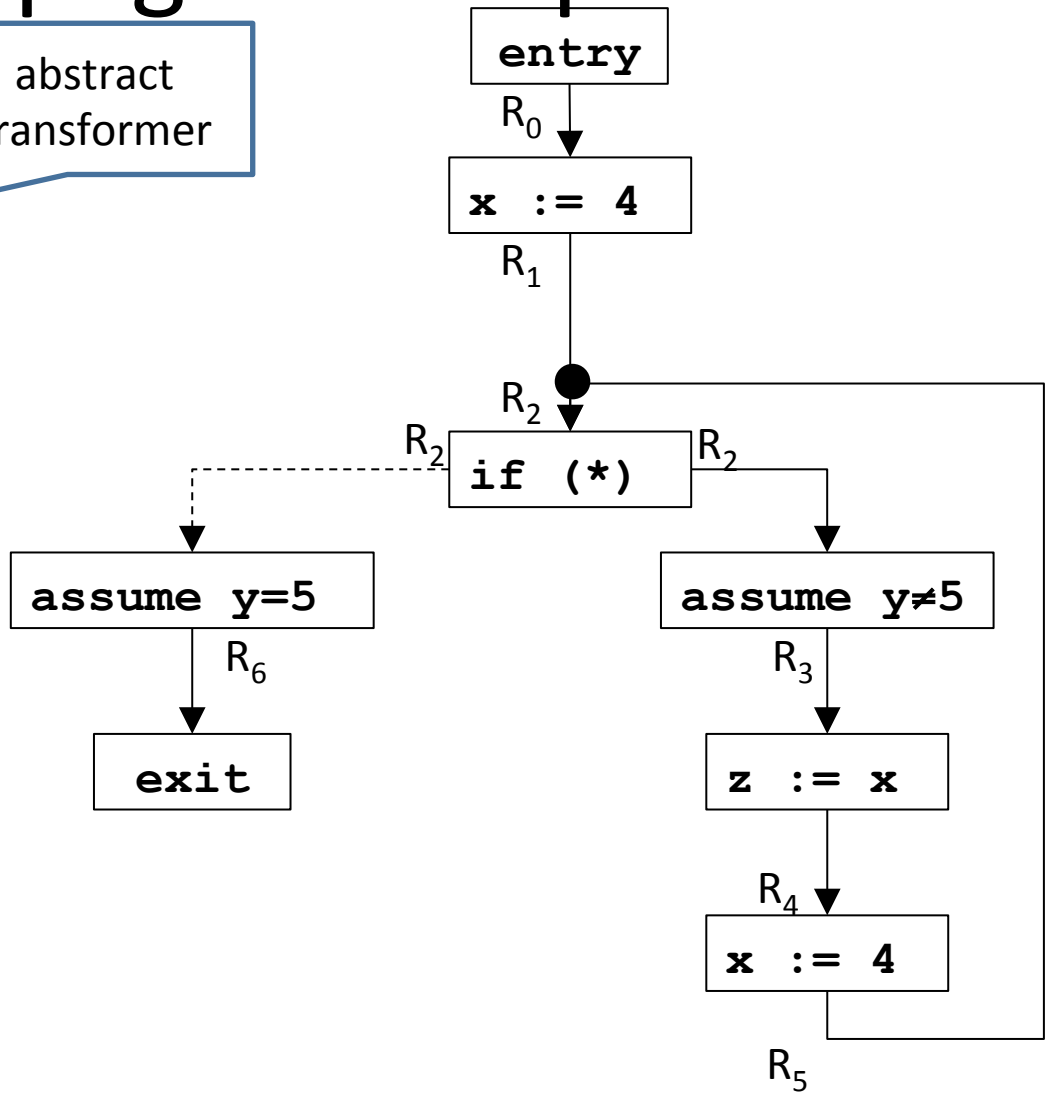
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

abstract transformer



Abstract operations for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

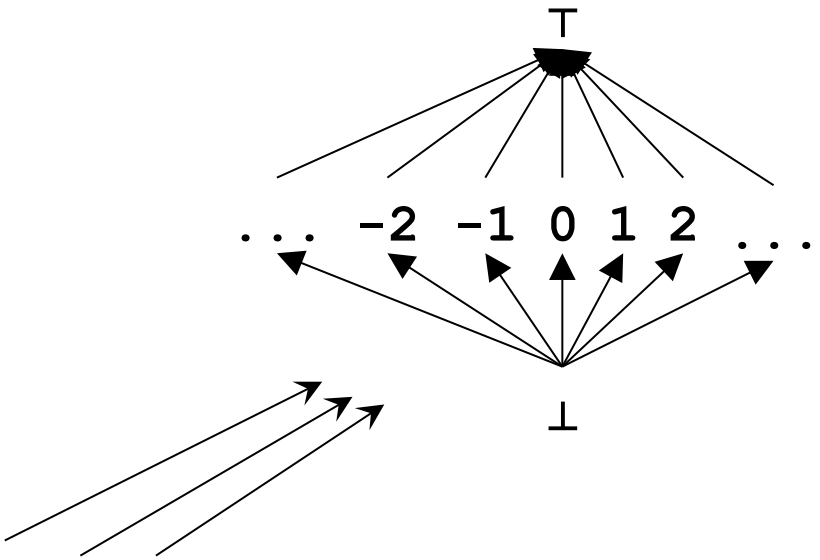
$R_3 = [\![\mathbf{assume\ y\neq 5}]\!]^\# R_2$

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

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

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

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 $v_y = k \neq \mathbf{5}$ 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

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

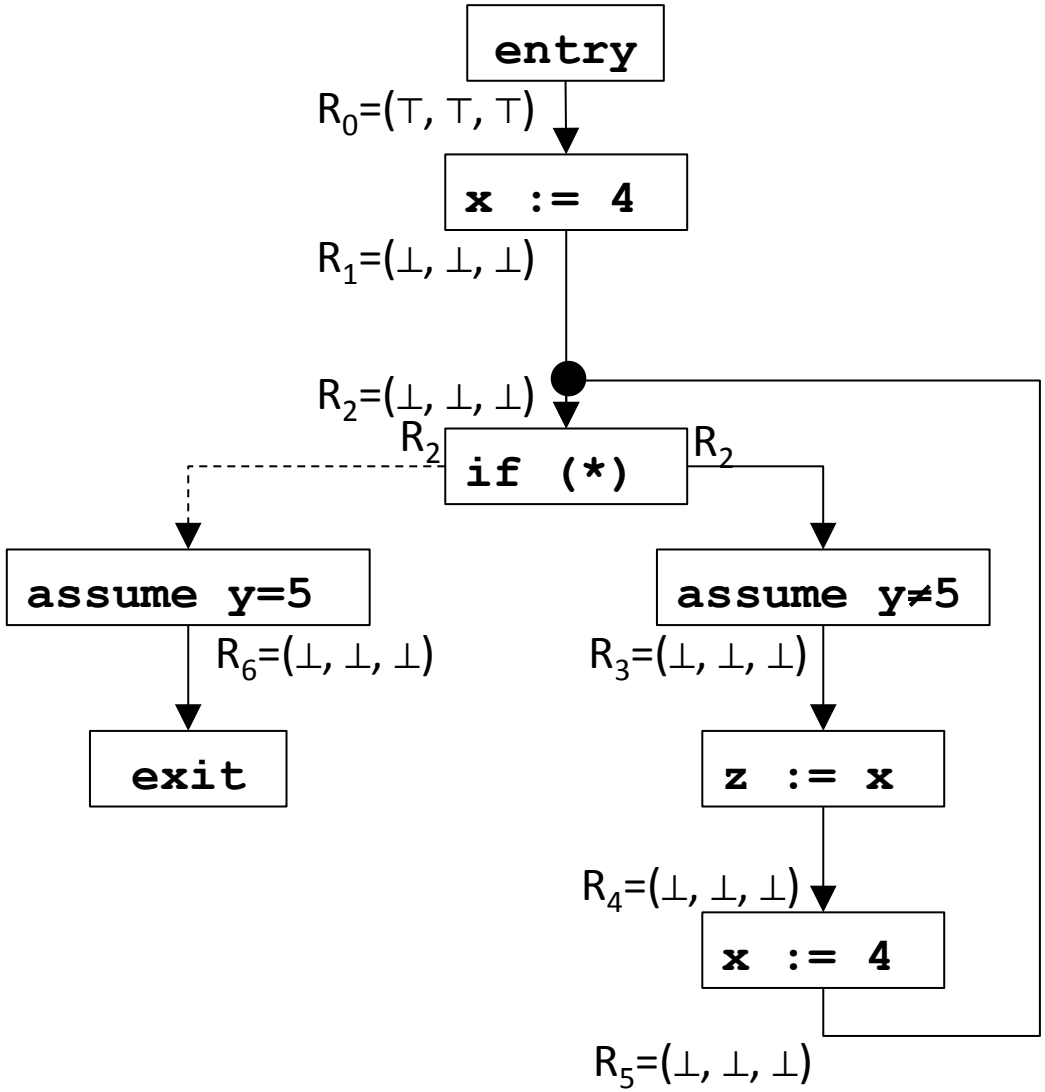
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

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



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

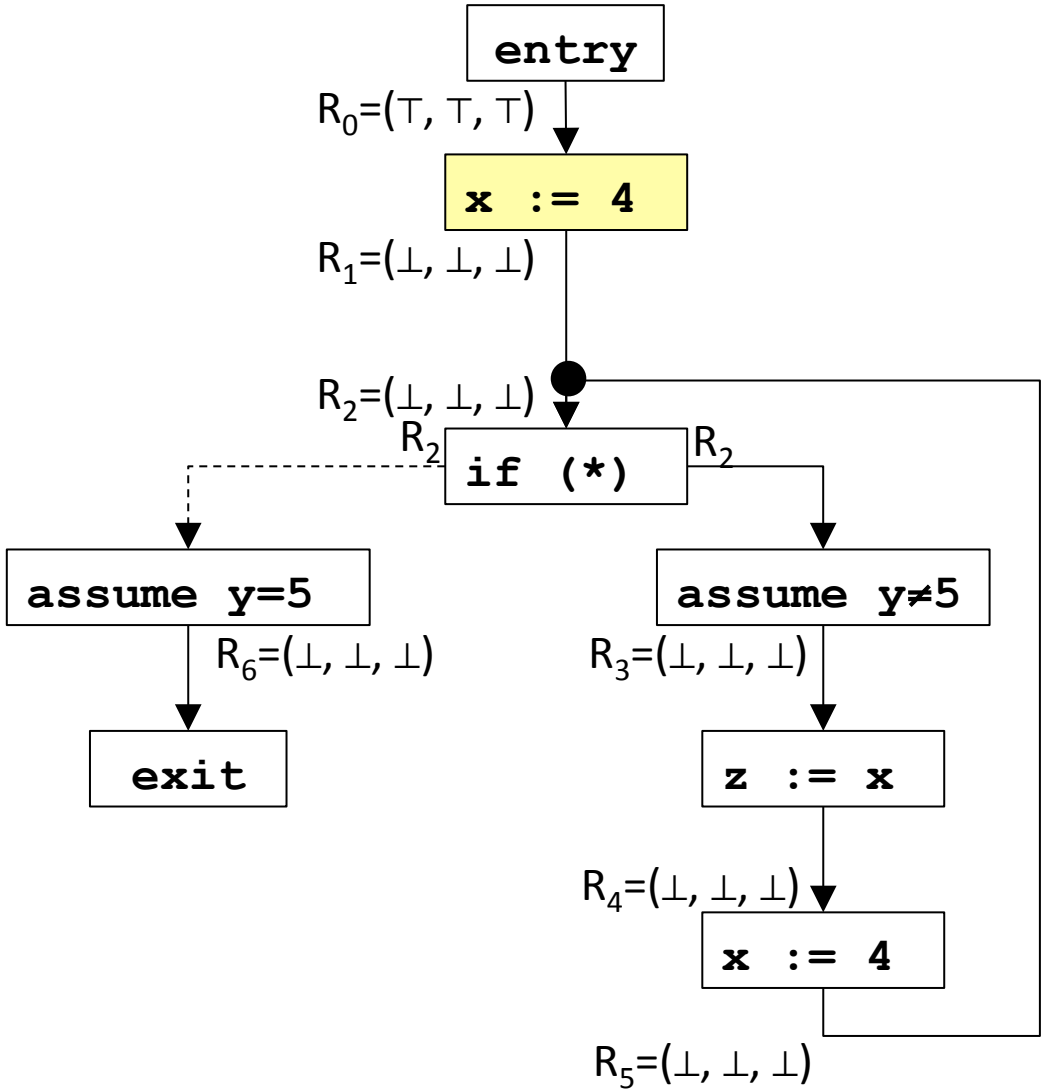
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

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

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

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

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



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

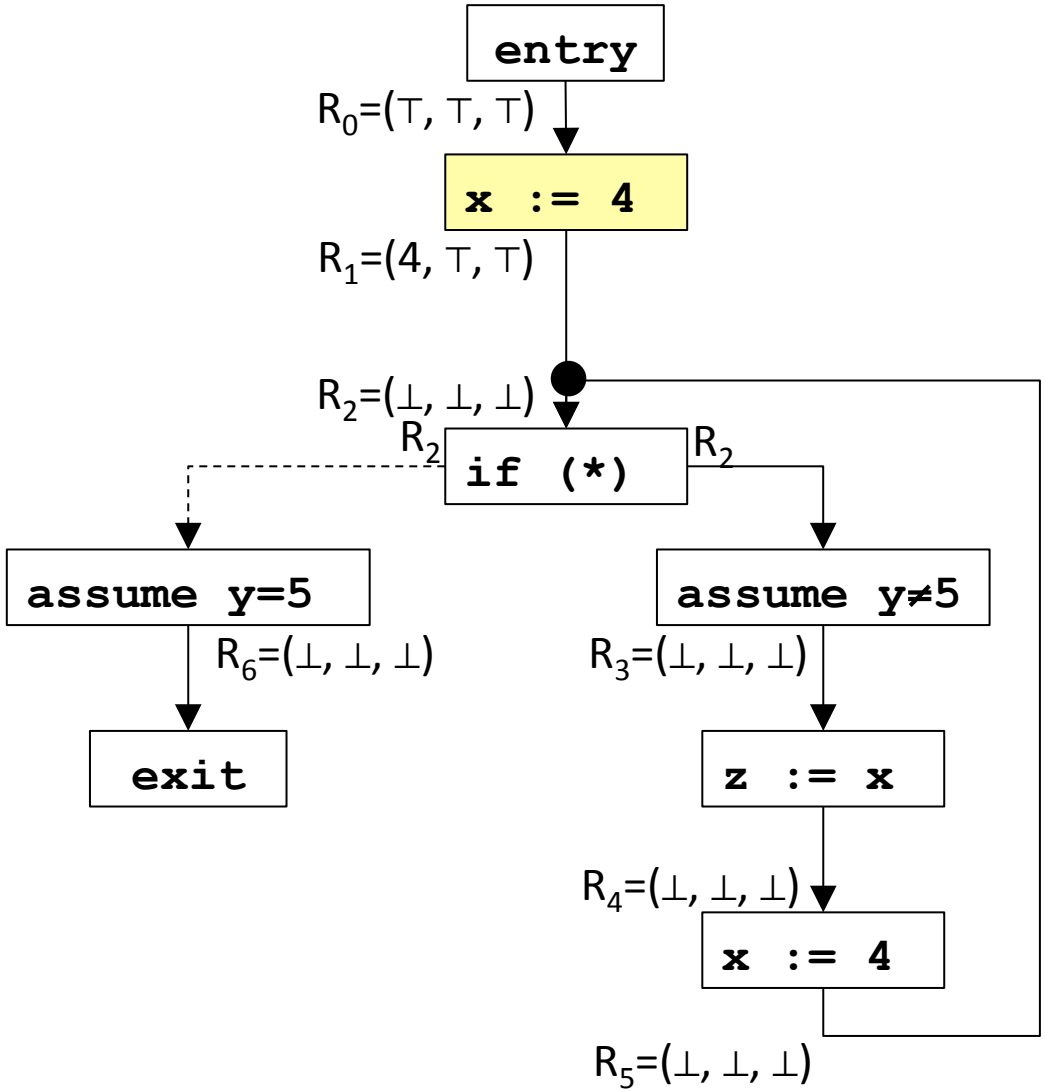
$R_3 = [\![\texttt{assume y≠5}]\!]^{\#} R_2$

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

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

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

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



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

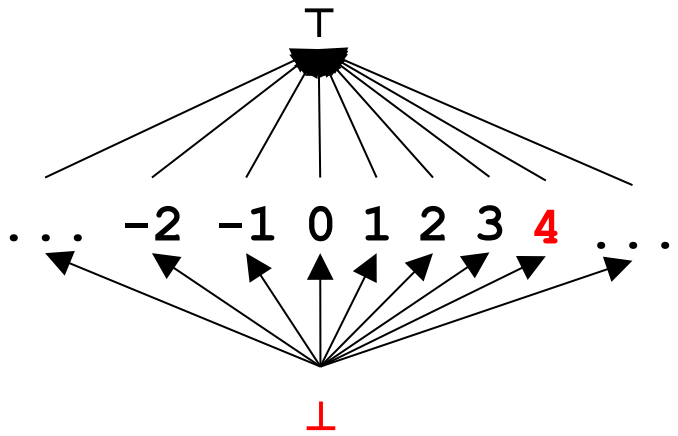
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

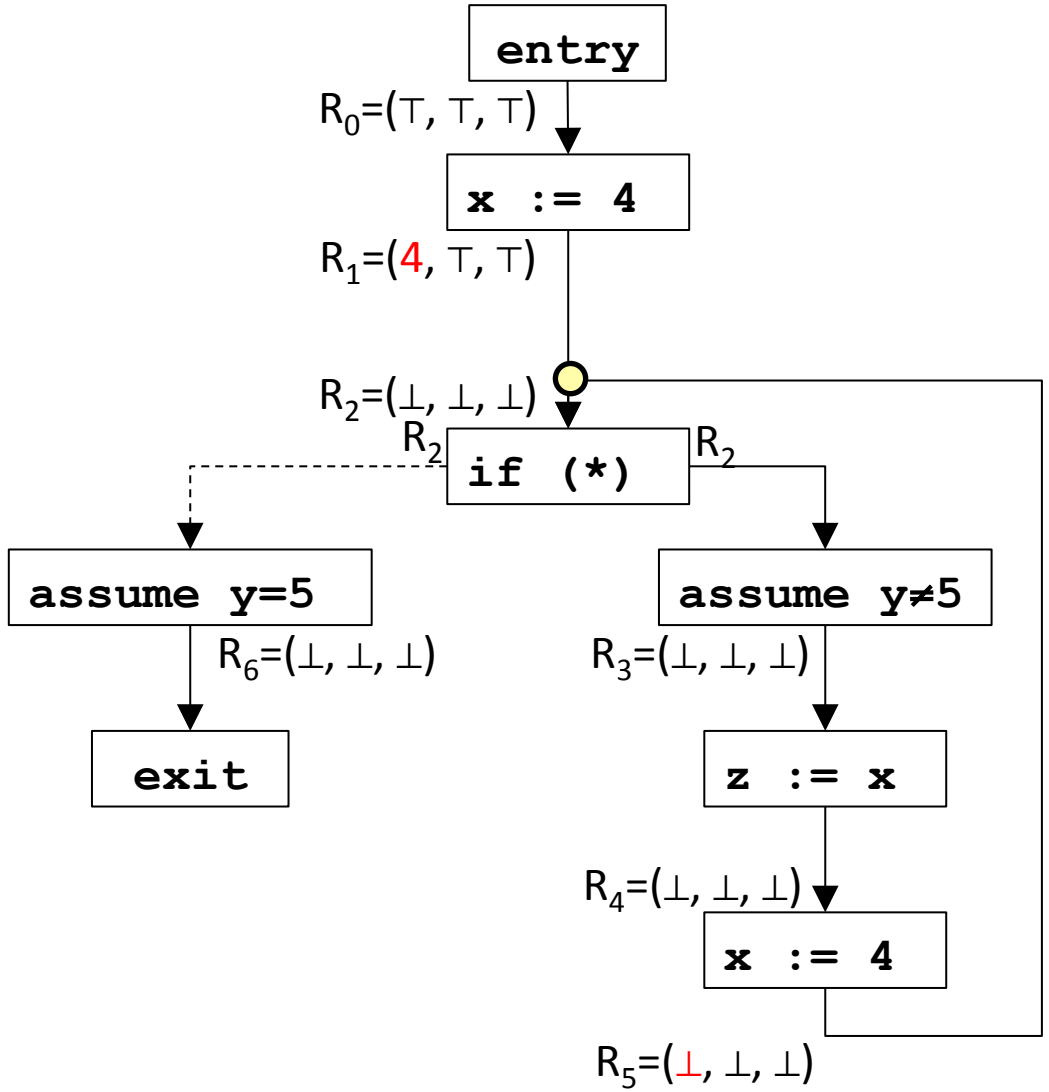
$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

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

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

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





Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

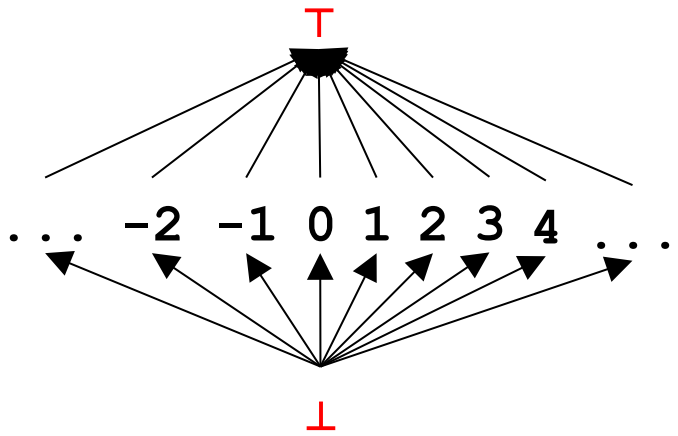
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

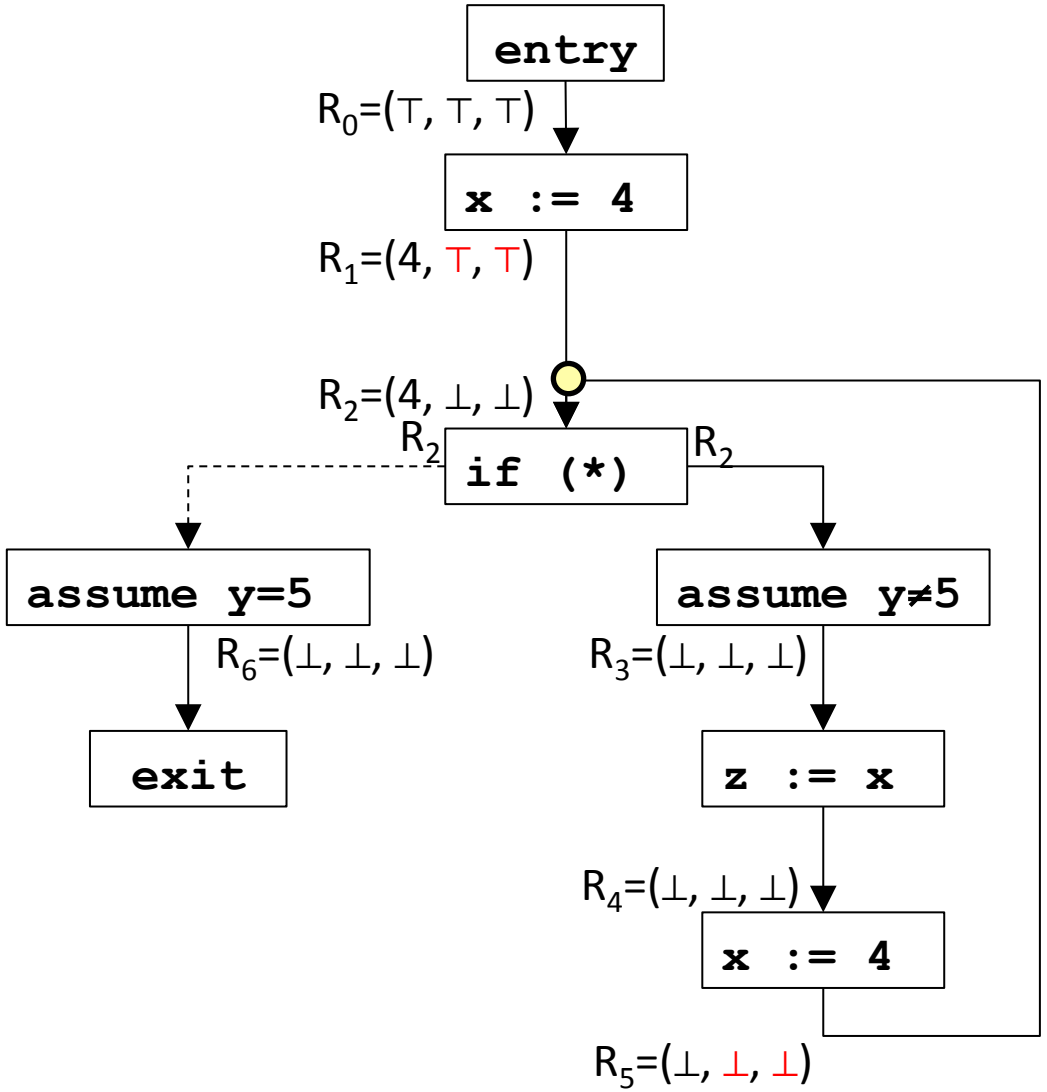
$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

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

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

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





Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

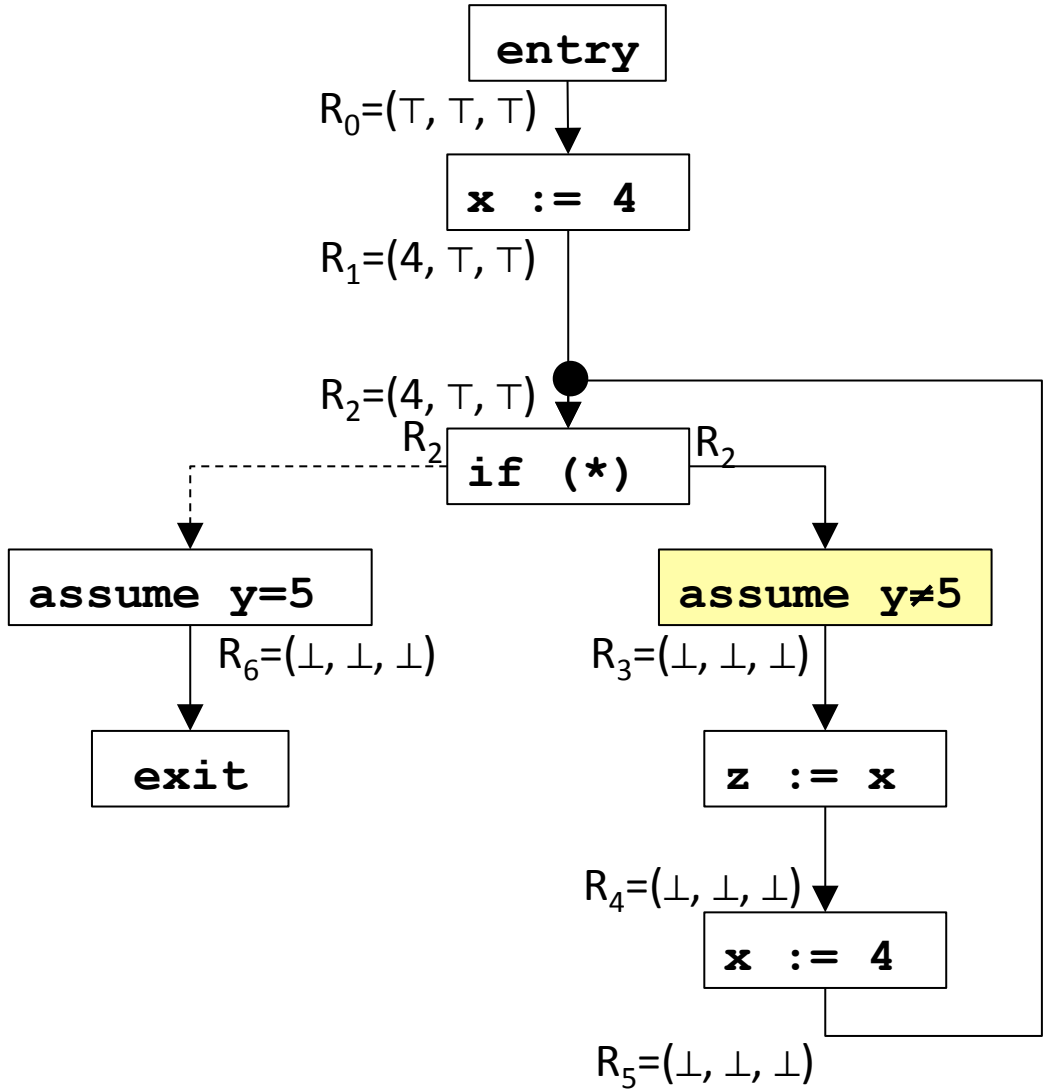
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

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



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

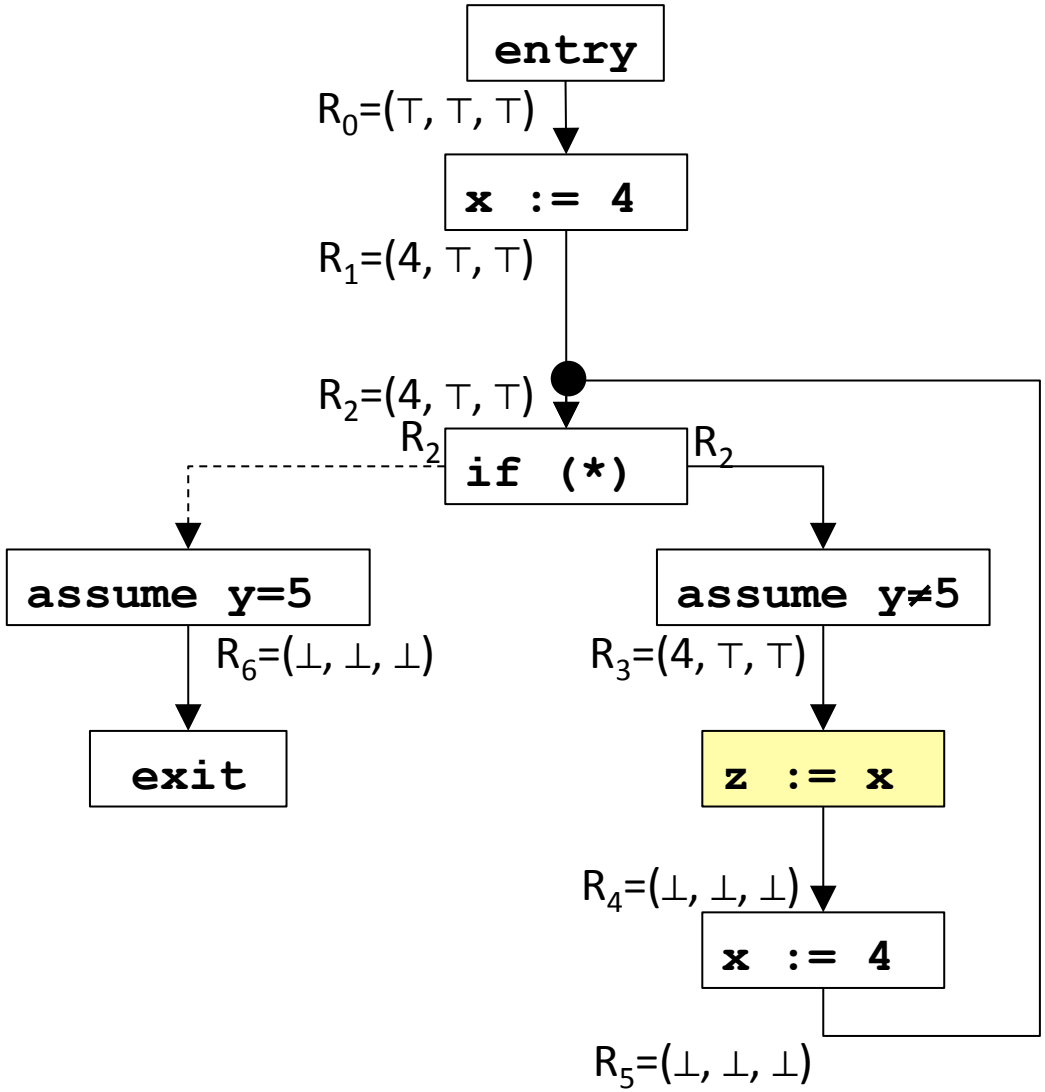
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

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



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

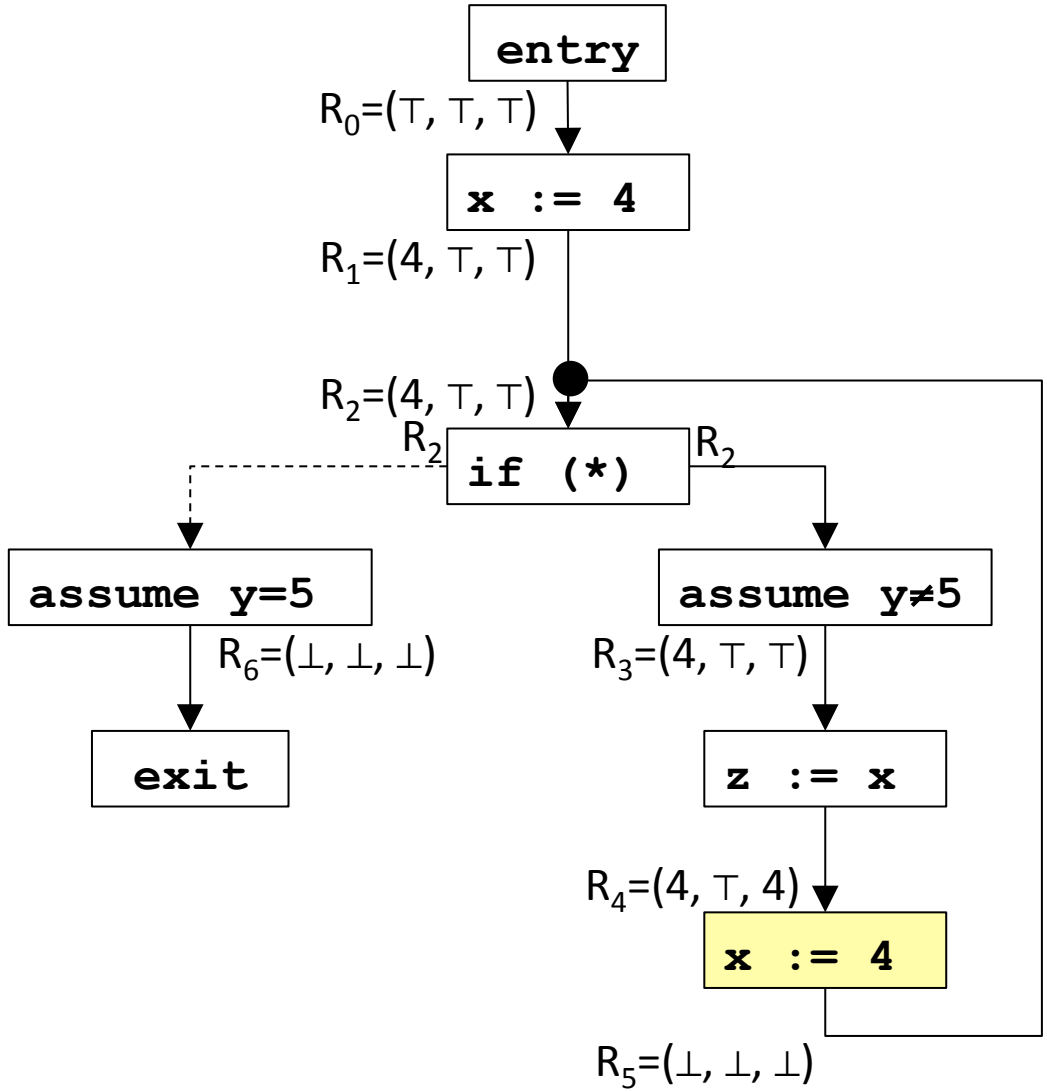
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

WL = $\{R_5, R_6\}$



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

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

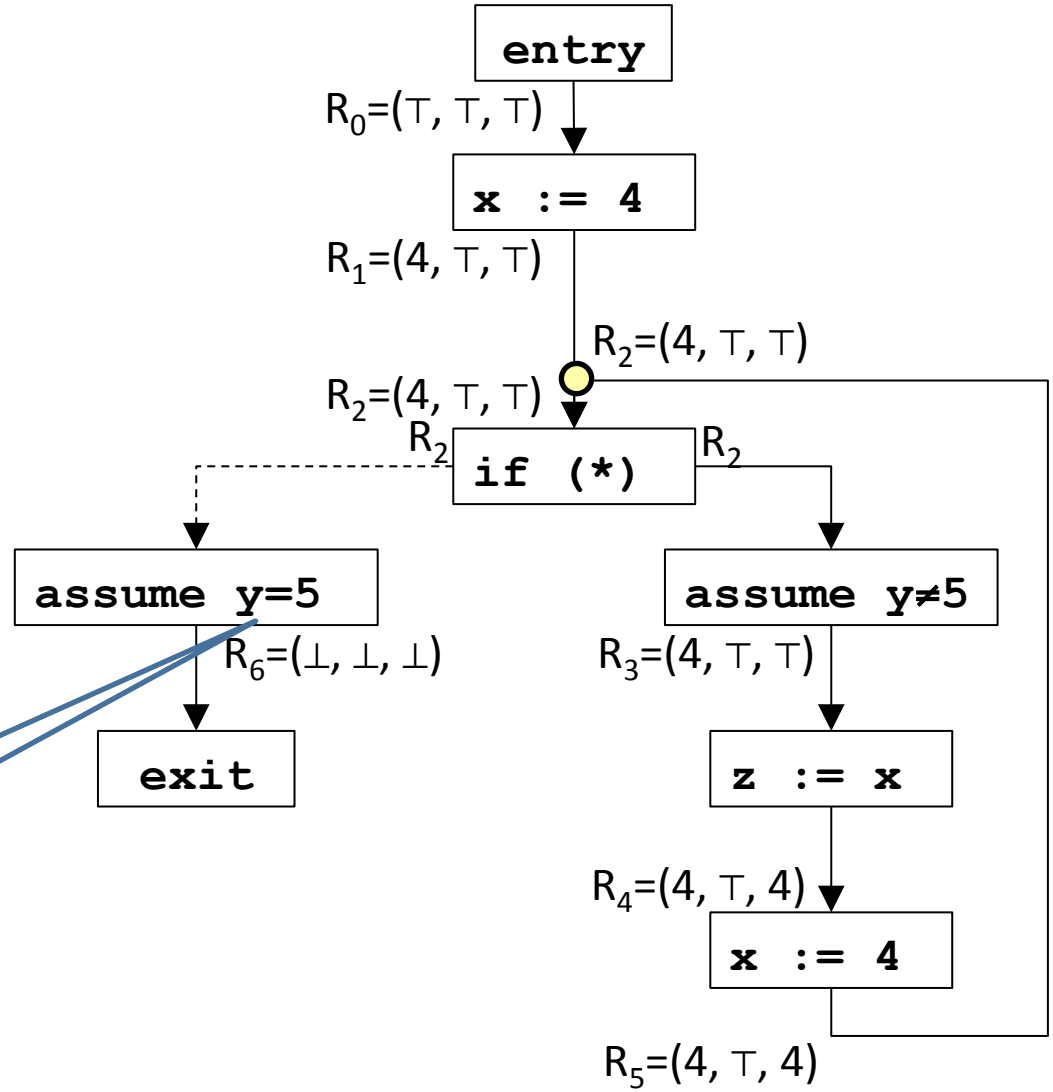
$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

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

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

WL = {R_2, R_6}

added R_2 back to worklist since it depends on R_5



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

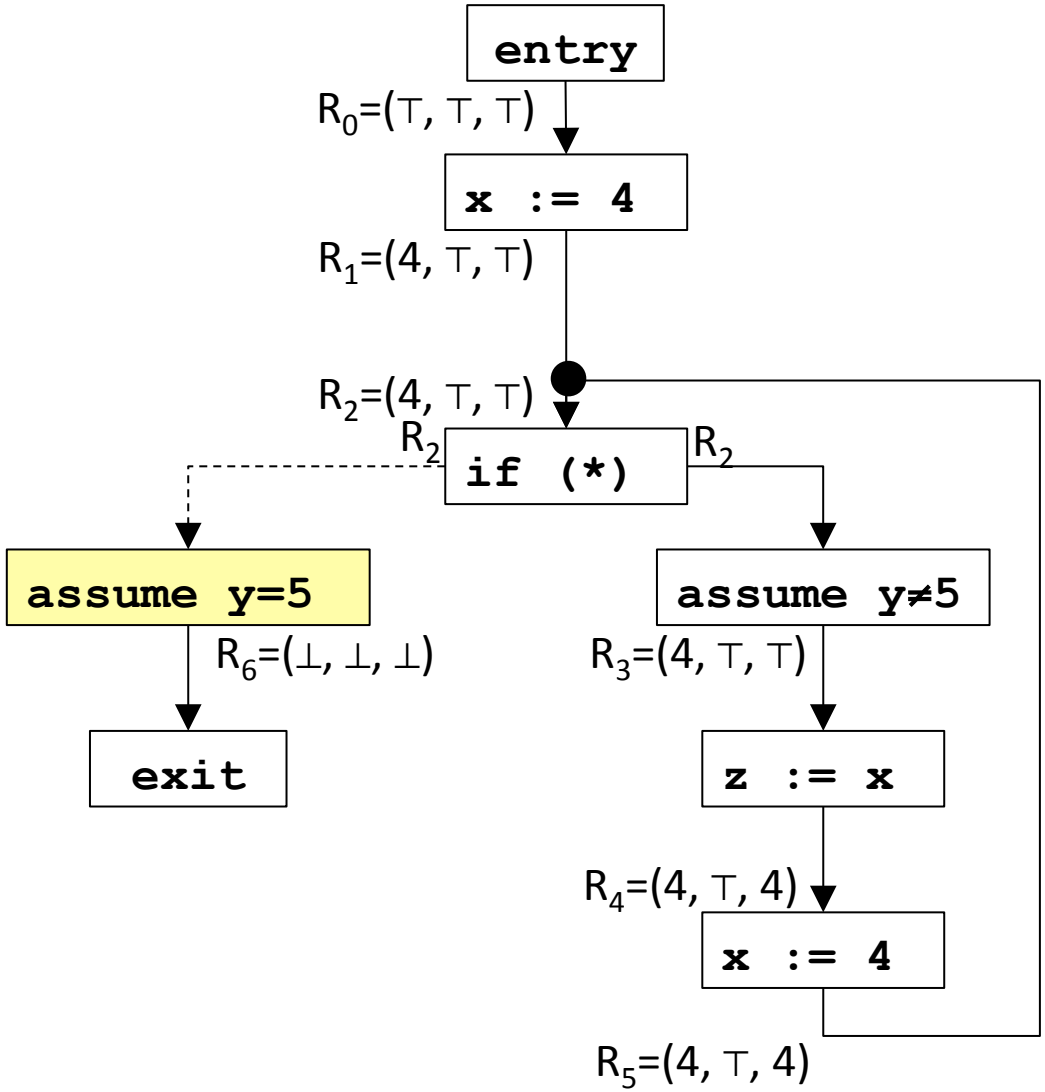
$R_3 = [\![\texttt{assume y≠5}]\!]^\# R_2$

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

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

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

WL = $\{R_6\}$



Chaotic iteration for CP

$R_0 = \top$

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

$R_2 = R_1 \sqcup R_5$

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

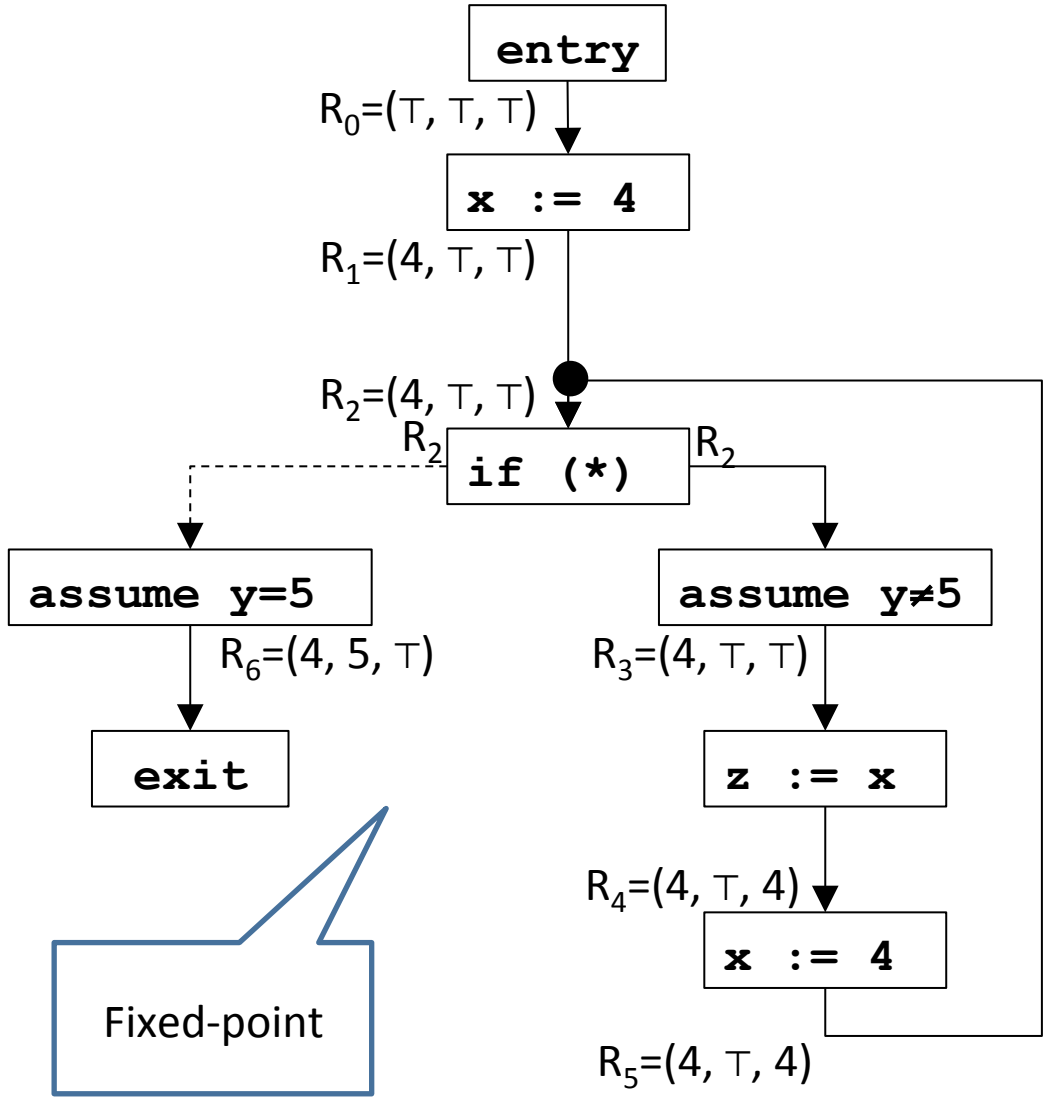
$R_4 = [\![\texttt{z:=x}]\!]^\# R_3$

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

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

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:
  $v_{out} = F[v](\sqcup u \mid (u,v) \text{ is a CFG edge})$
  - 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

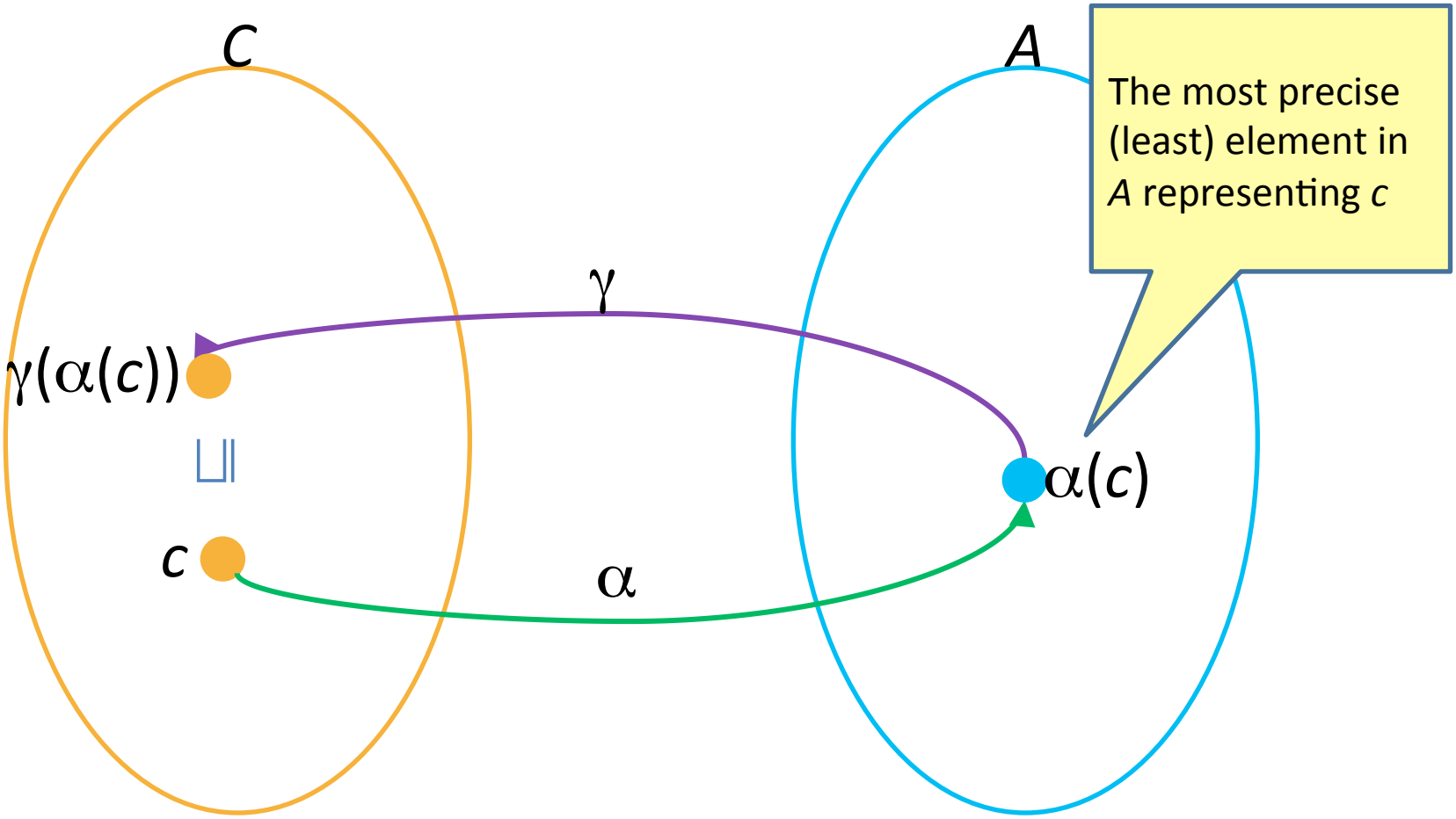
- We defined a reference semantics – the collecting semantics
- We defined an abstract semantics for a given lattice and abstract transformers
- We defined an algorithm to compute abstract least fixed-point when transformers are monotone and lattice obeys ACC
- Questions:

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

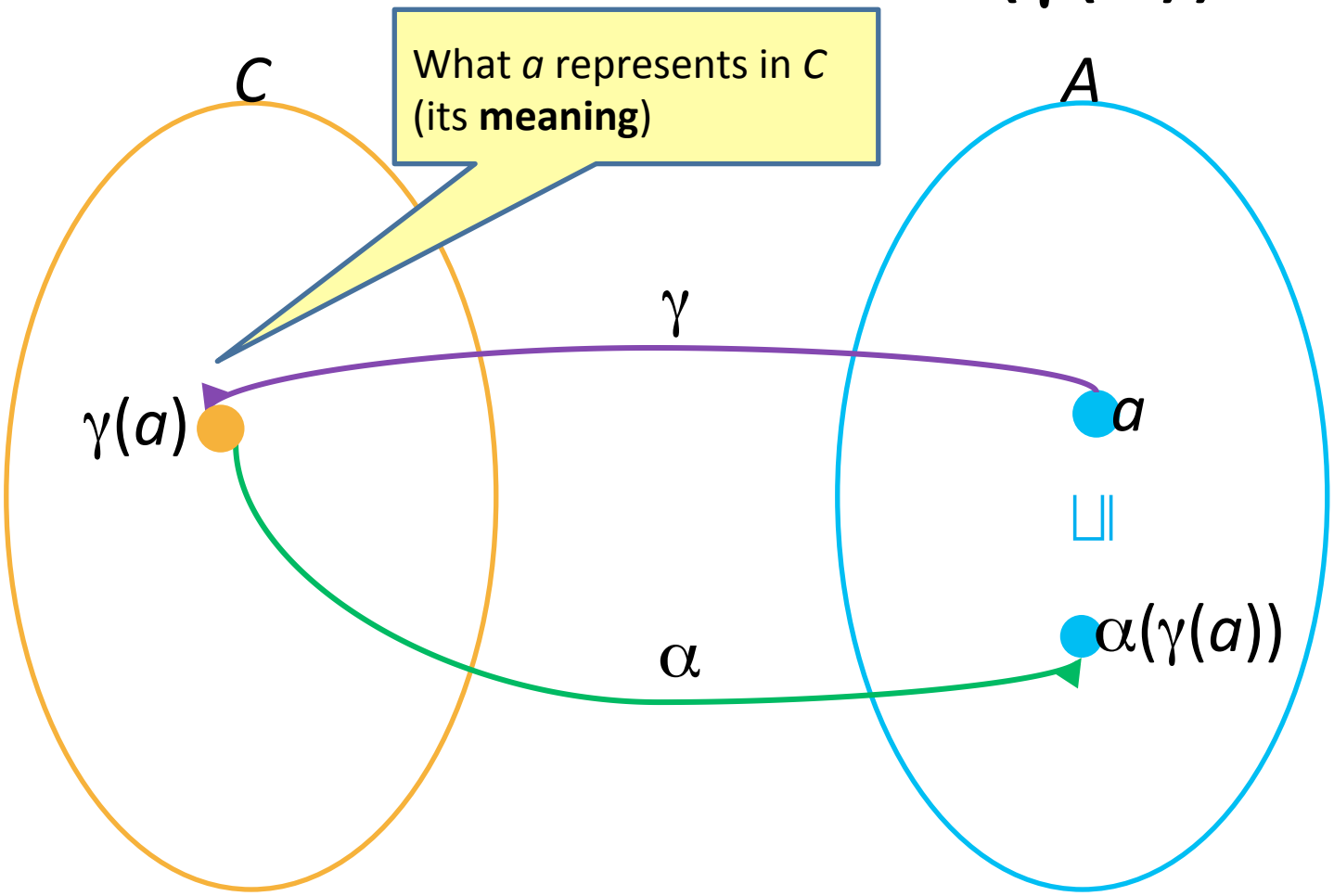
Galois Connection

- 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: $\alpha(\gamma(a)) \sqsubseteq a$



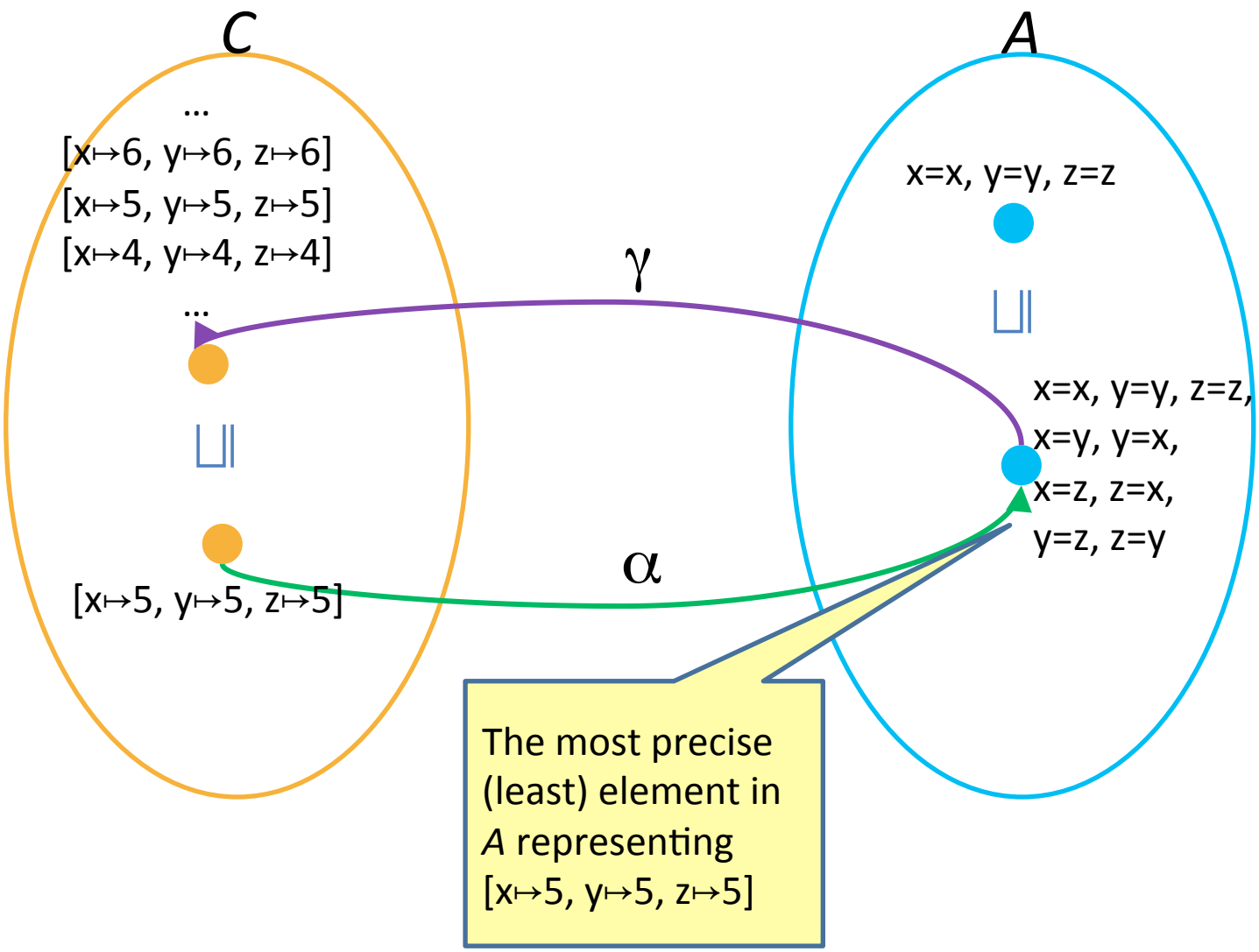
Example: lattice of equalities

- 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

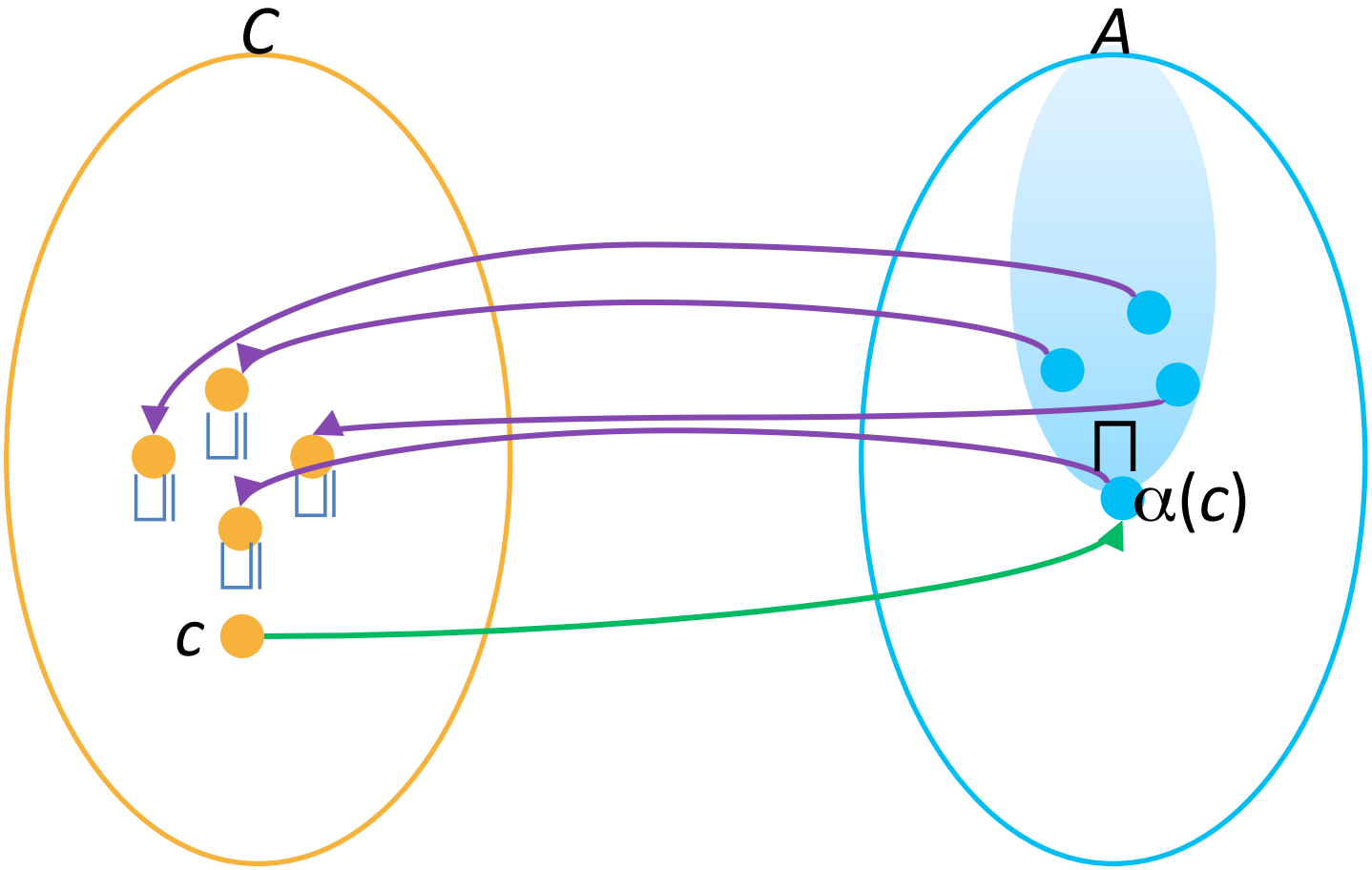
- Concrete lattice:
  $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- Abstract lattice:
  $EQ = \{ x{=}y \mid x, y \in \mathrm{Var}\}$
  $A = (2^{EQ}, \supseteq, \cap, \cup, EQ\ , \varnothing)$
  - Treat elements of A as both formulas and sets of constraints
- Useful 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: $c \sqsubseteq \gamma(\alpha(c))$



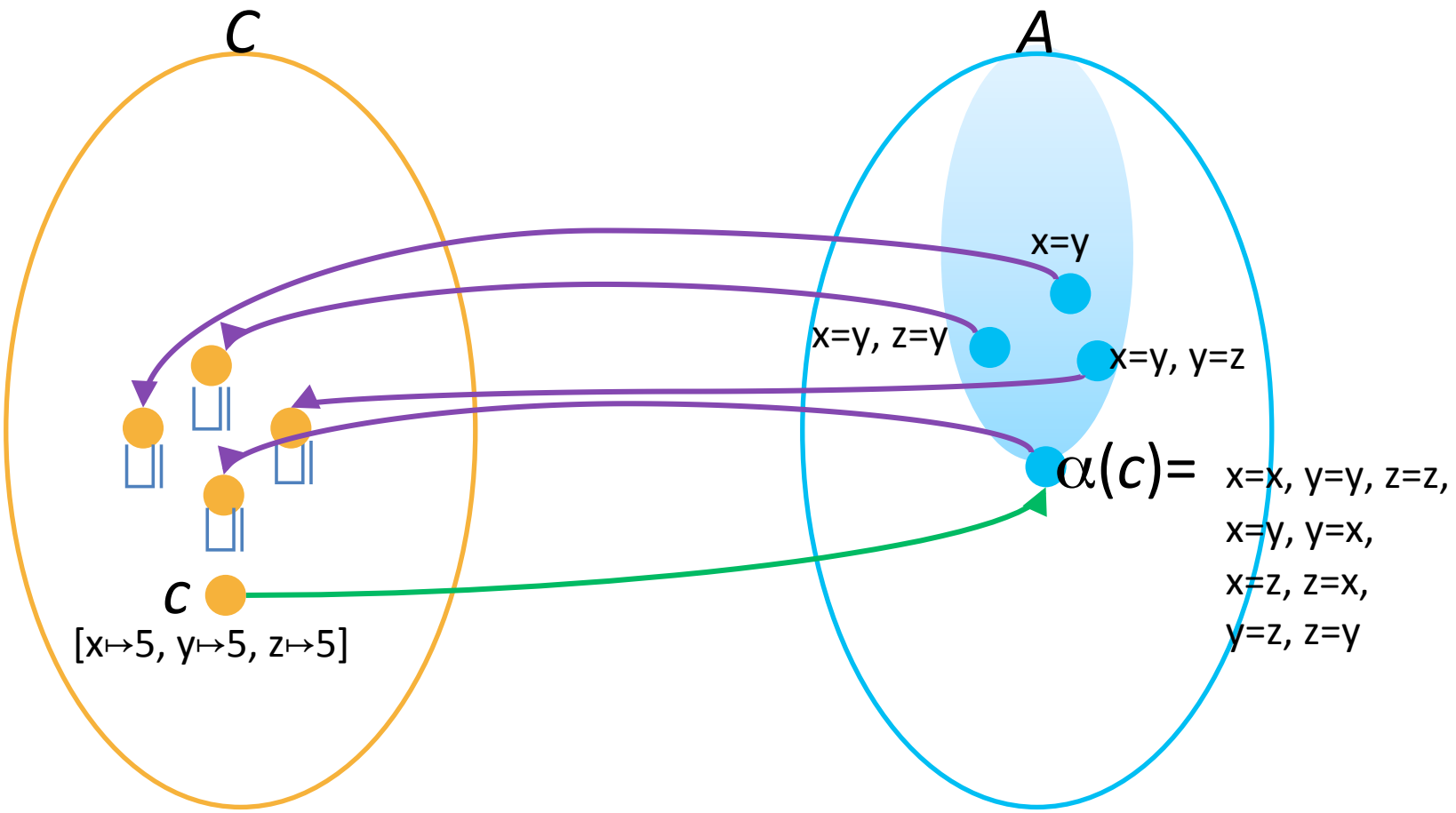
Most precise abstract representation

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

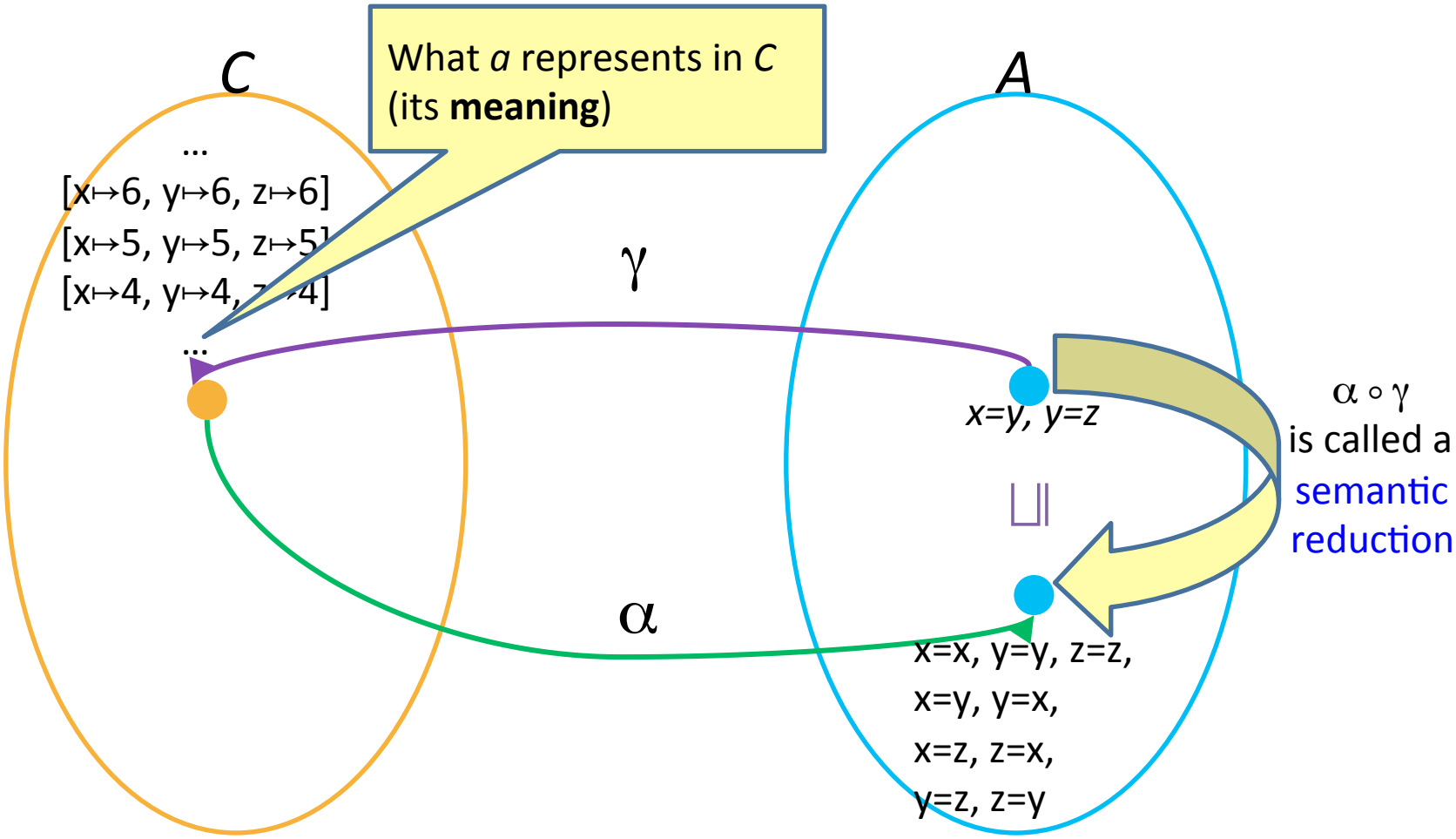


Most precise abstract representation

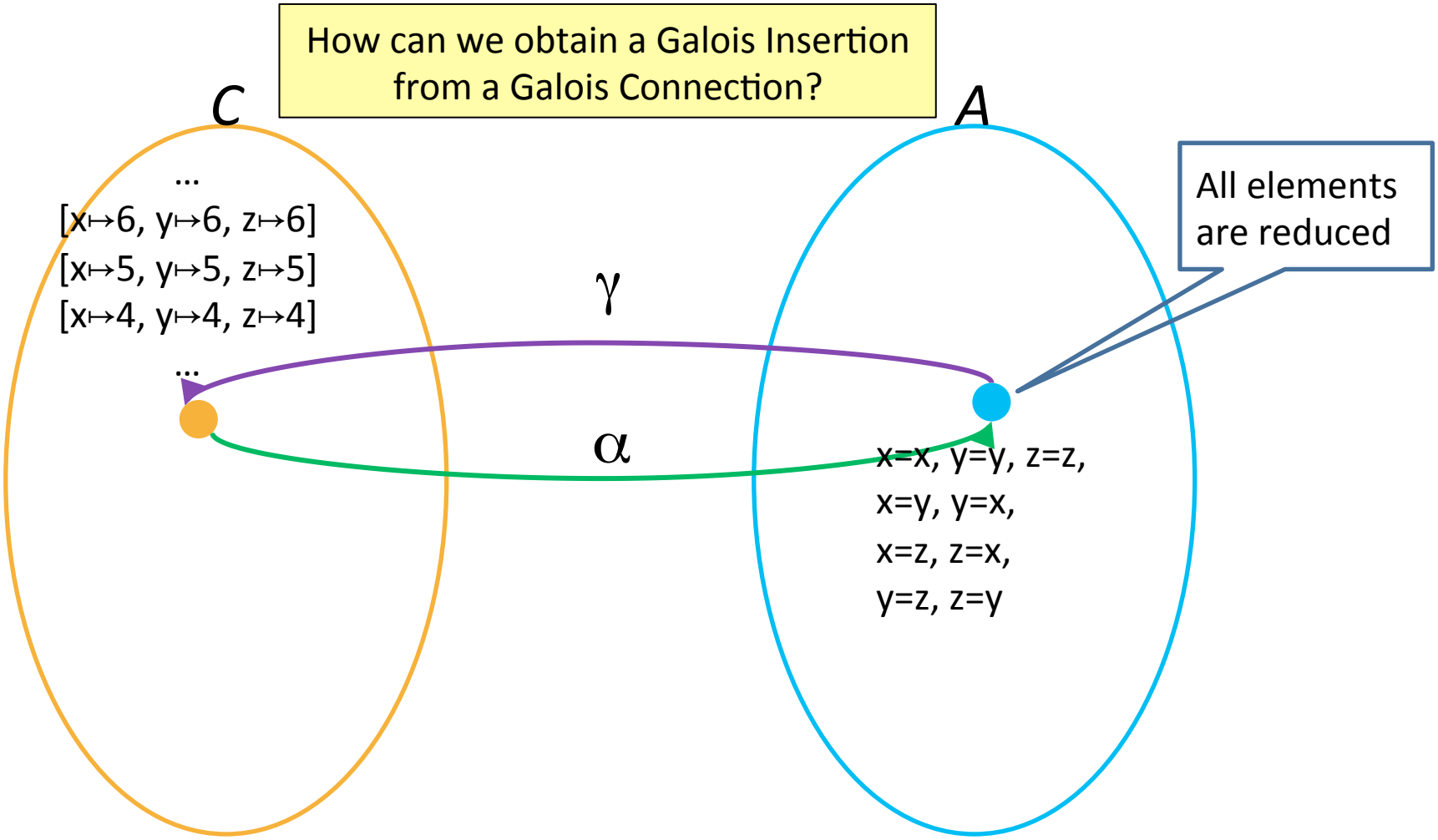
$$\alpha(c) = \sqcap\{c' \mid c \sqsubseteq \gamma(c')\}$$



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



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



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

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

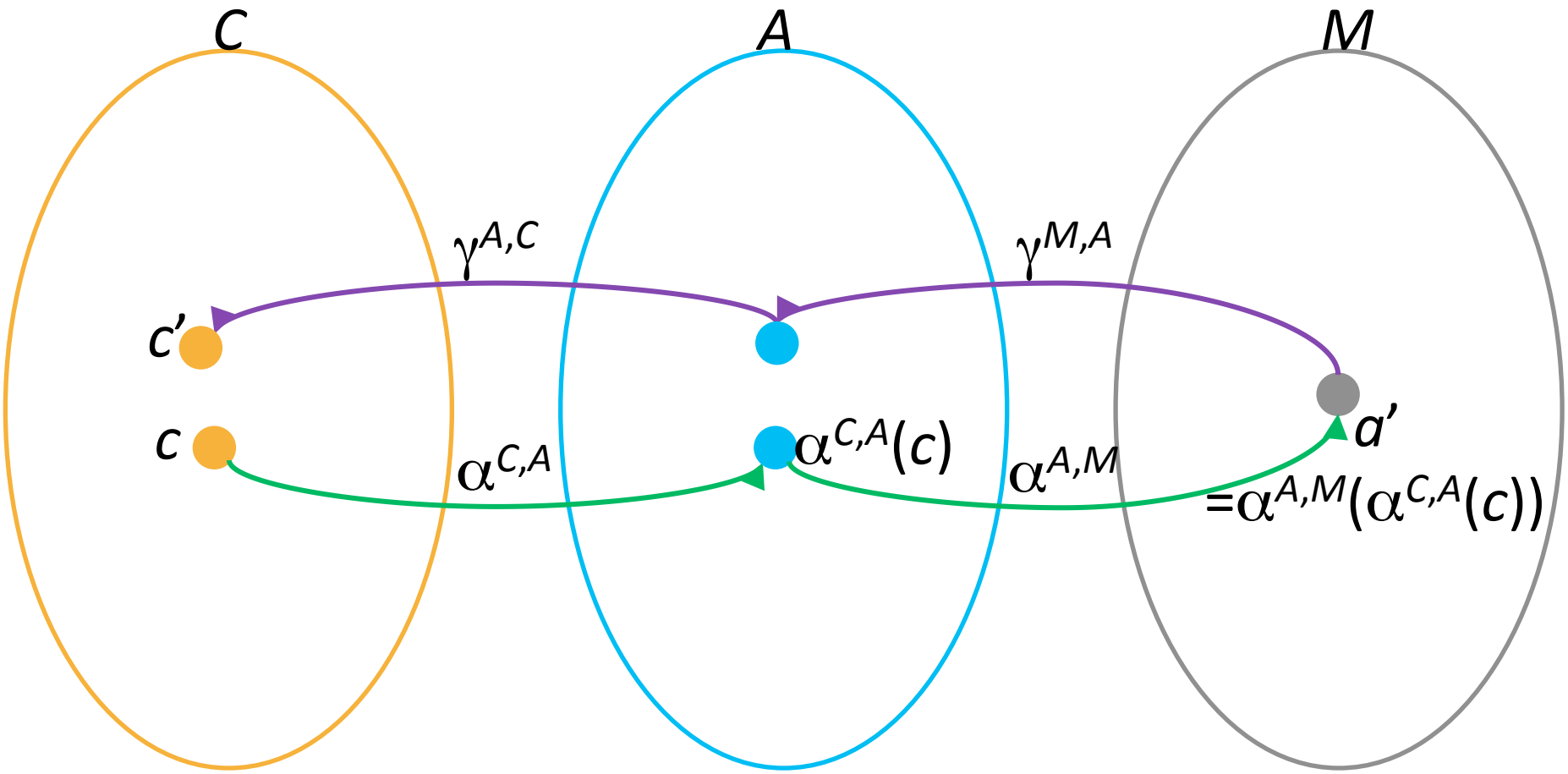
The case of symbolic domains

- 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
  $$\beta(s) = \alpha(\{s\}) = \sqcap^A\{\varphi \mid s \models \varphi\}$$
  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
  $$\gamma(\varphi) = \{ s \mid s \models \varphi \} = \text{models}(\varphi)$$

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

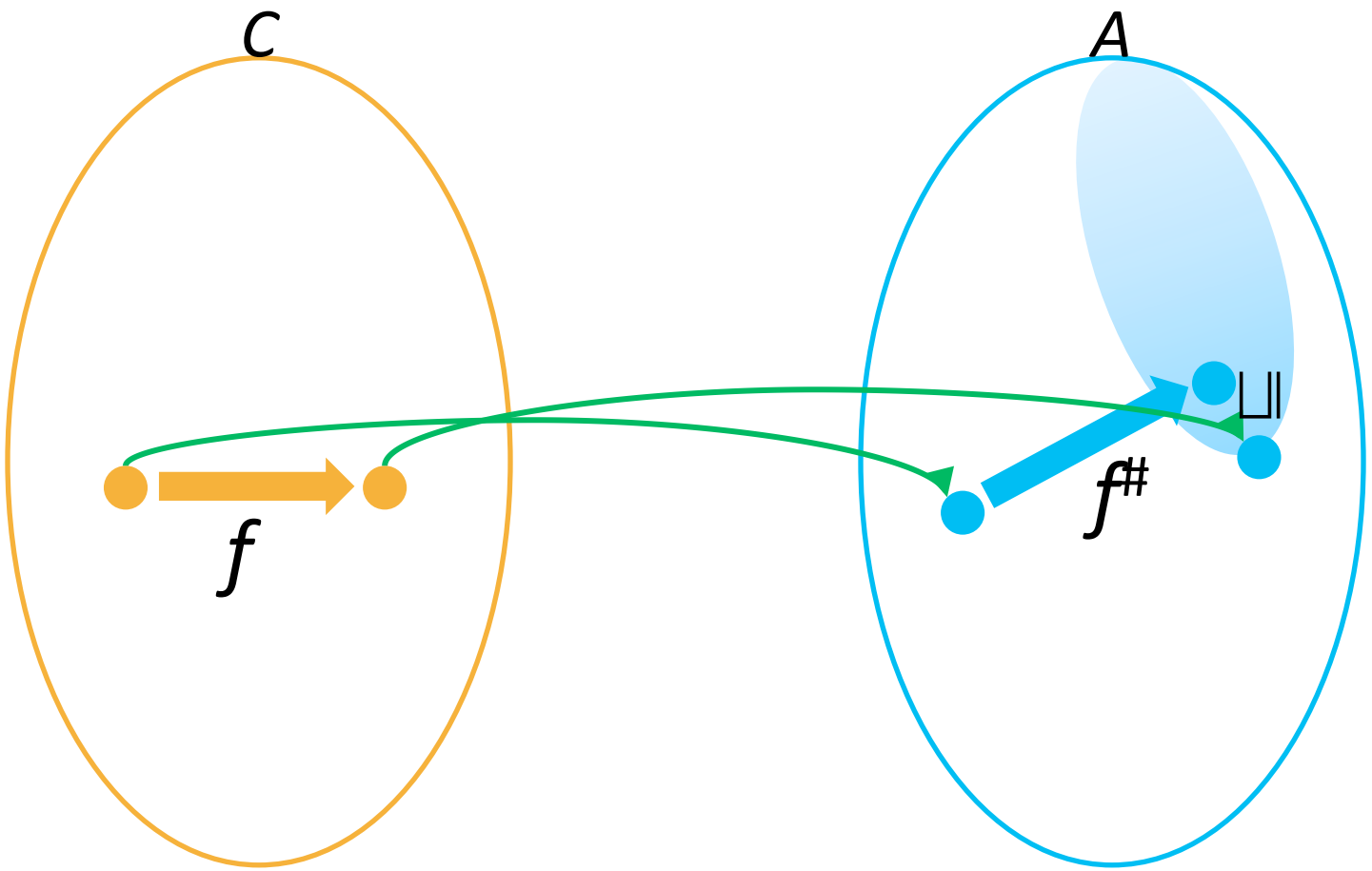


Sound abstract transformer

- 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 $f : D^C \longrightarrow D^C$
  an abstract transformer $f^\# : D^A \longrightarrow D^A$
- 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
    $\alpha(f(\gamma(a))) \sqsubseteq^A f^\#(a)$

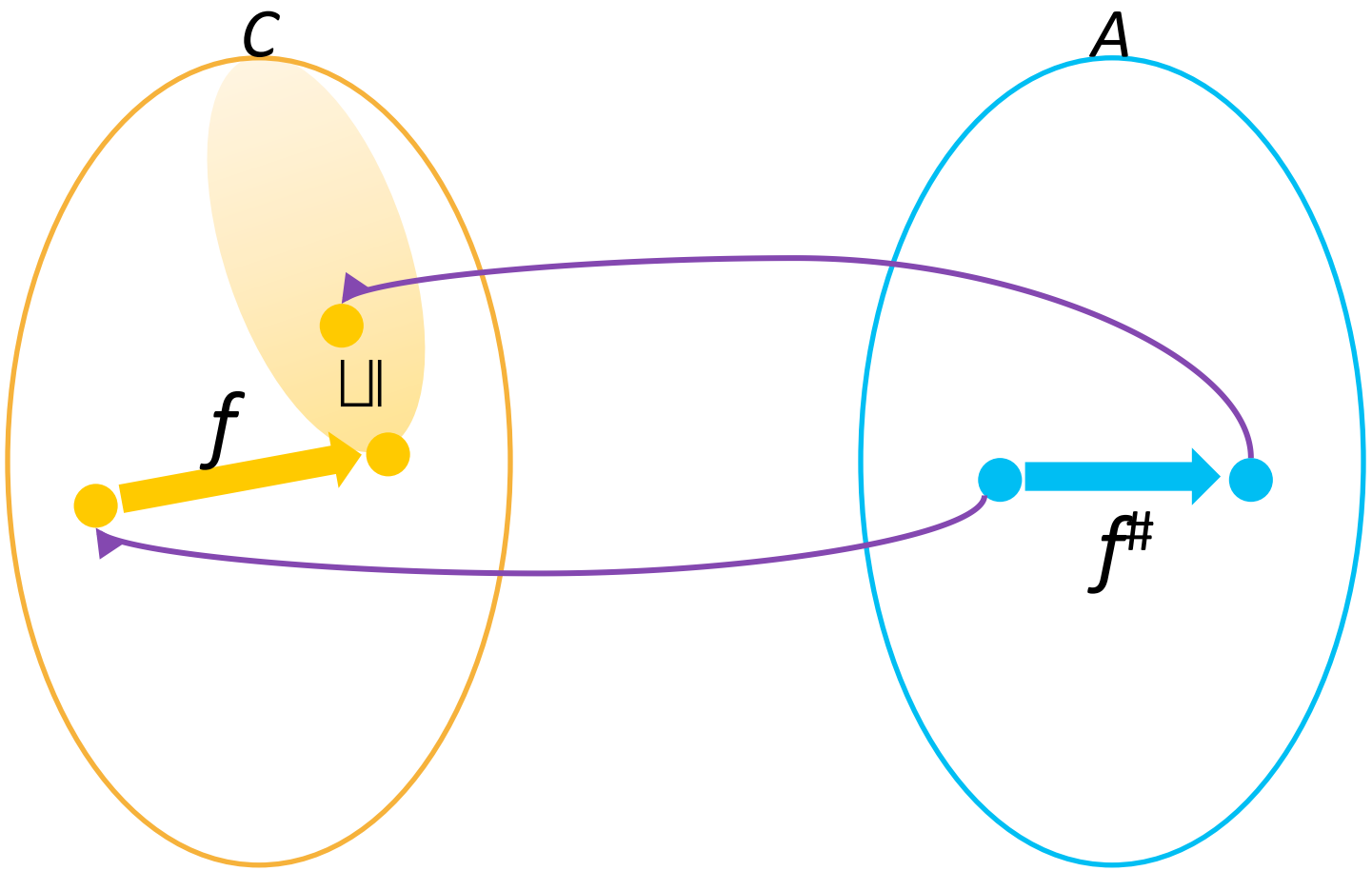
Transformer soundness condition 1

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



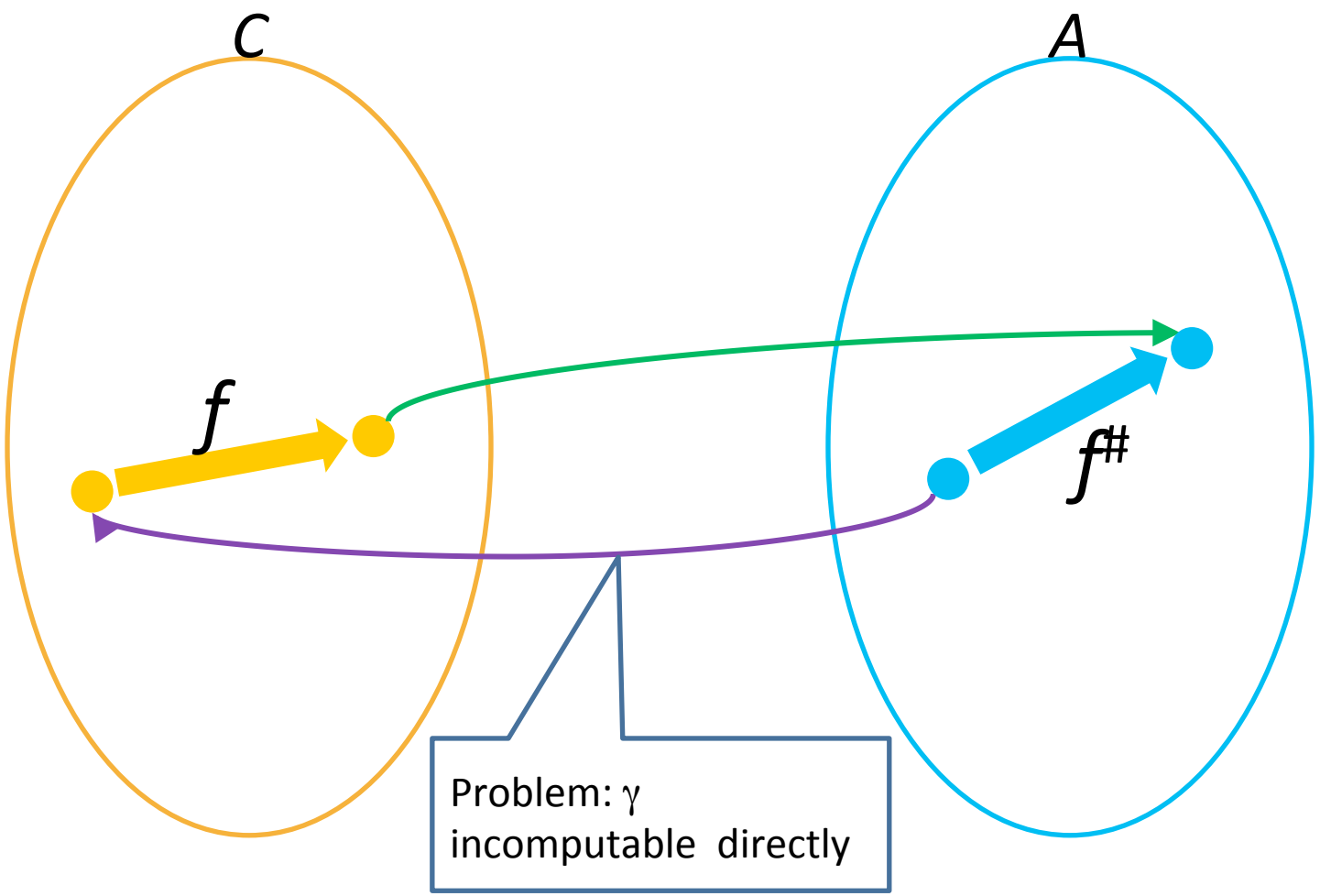
Transformer soundness condition 2

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



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

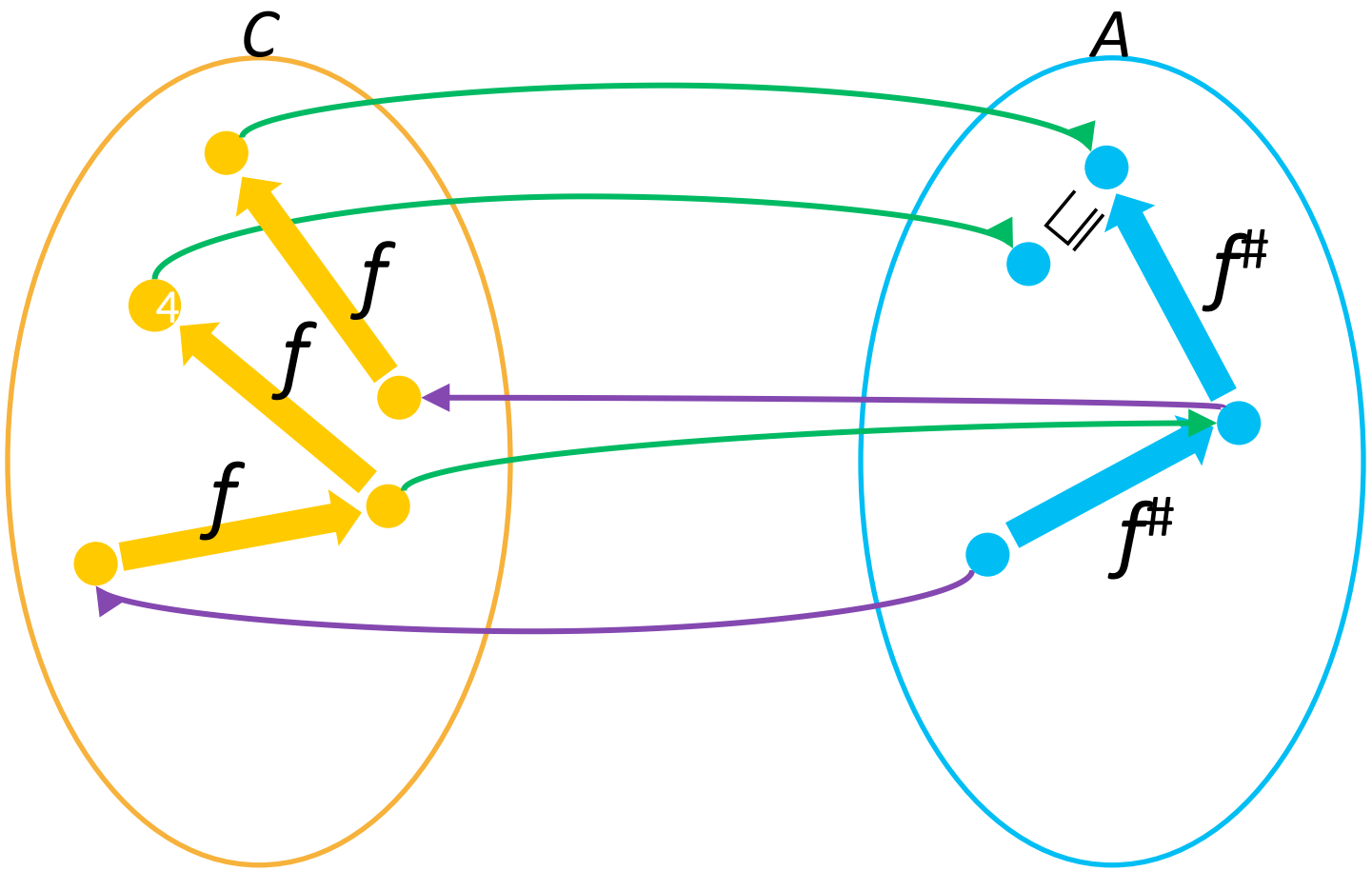
- $[\![x{:=}y]\!]^{\#2}\ )\wedge\{y{=}z\}) = \{x{=}y,\ y{=}z\} \sqsupseteq \{x{=}y,\ y{=}z,\ x{=}z\}$
- Idea: apply reduction operator again after the vanilla transformer
- $[\![x{:=}y]\!]^{\#3}$ = Explicate $\circ$ $[\![x{:=}y]\!]^{\#1}$ $\circ$ Explicate
- Observation : after the first time we apply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

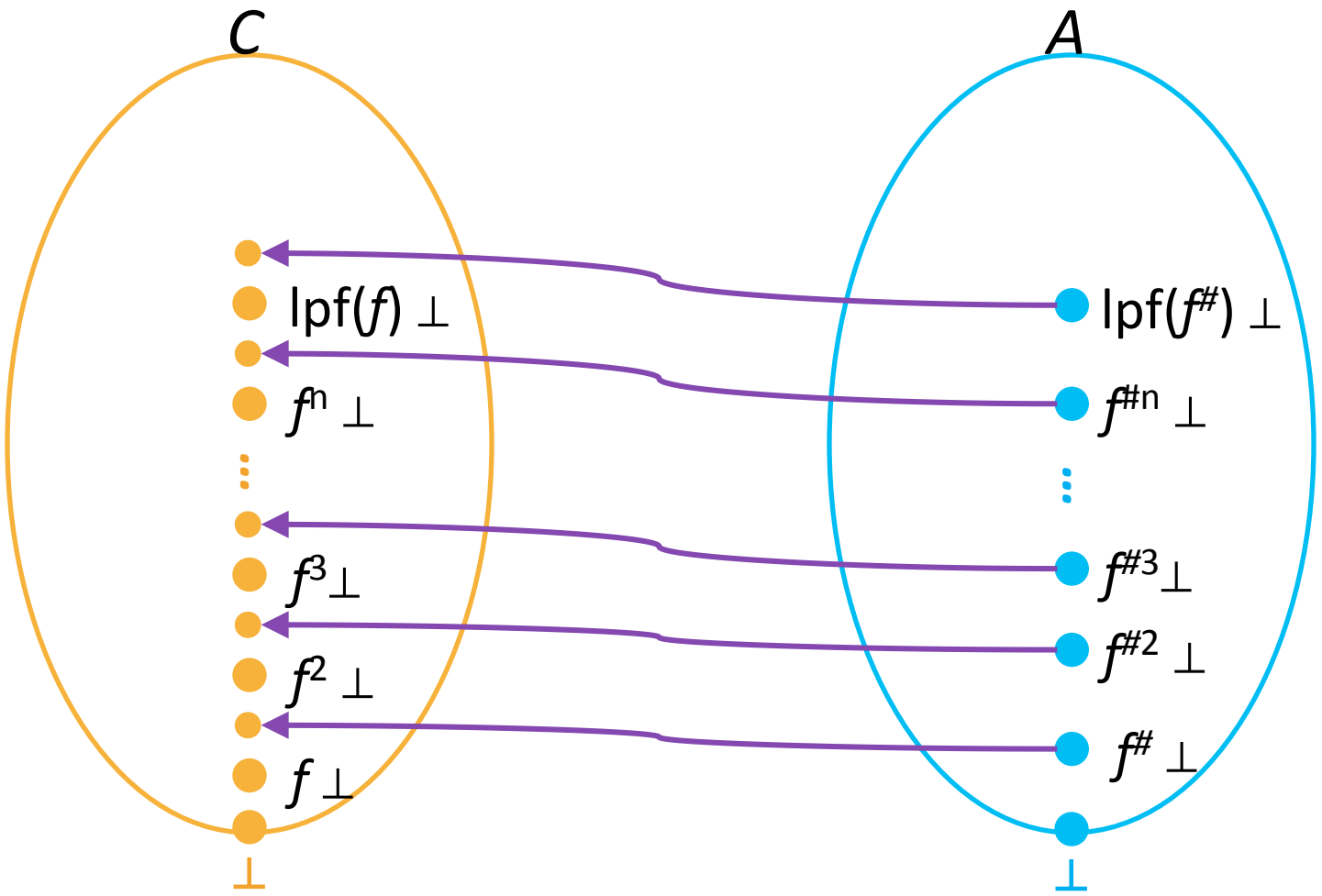
1. Given two complete lattices
   $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
   $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
   and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \rightarrow D^C$
3. Monotone abstract transformer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

Then

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

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

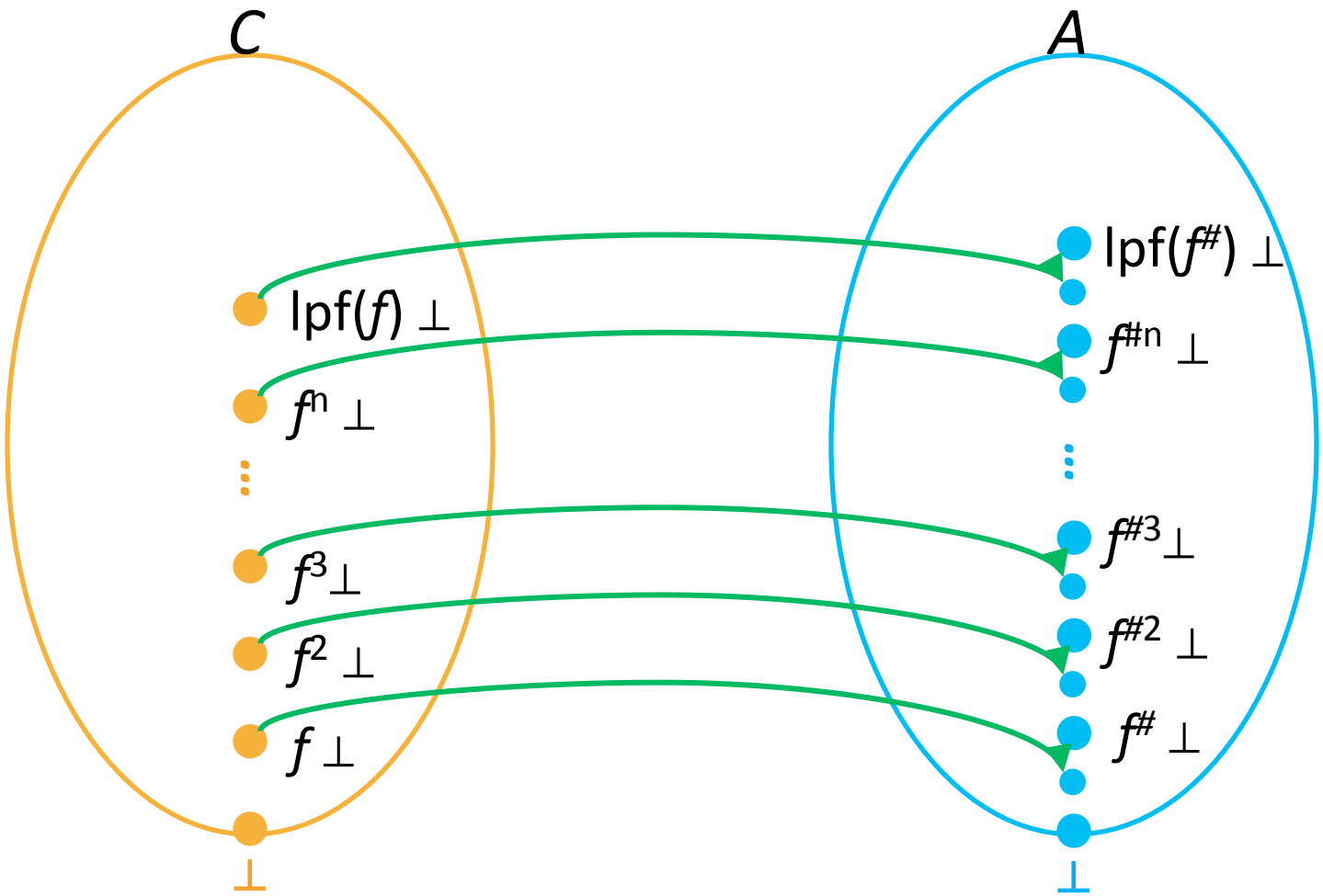
1. Given two complete lattices
   $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \bot^C, \top^C)$
   $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \bot^A, \top^A)$
   and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \longrightarrow D^C$
3. Monotone abstract transformer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

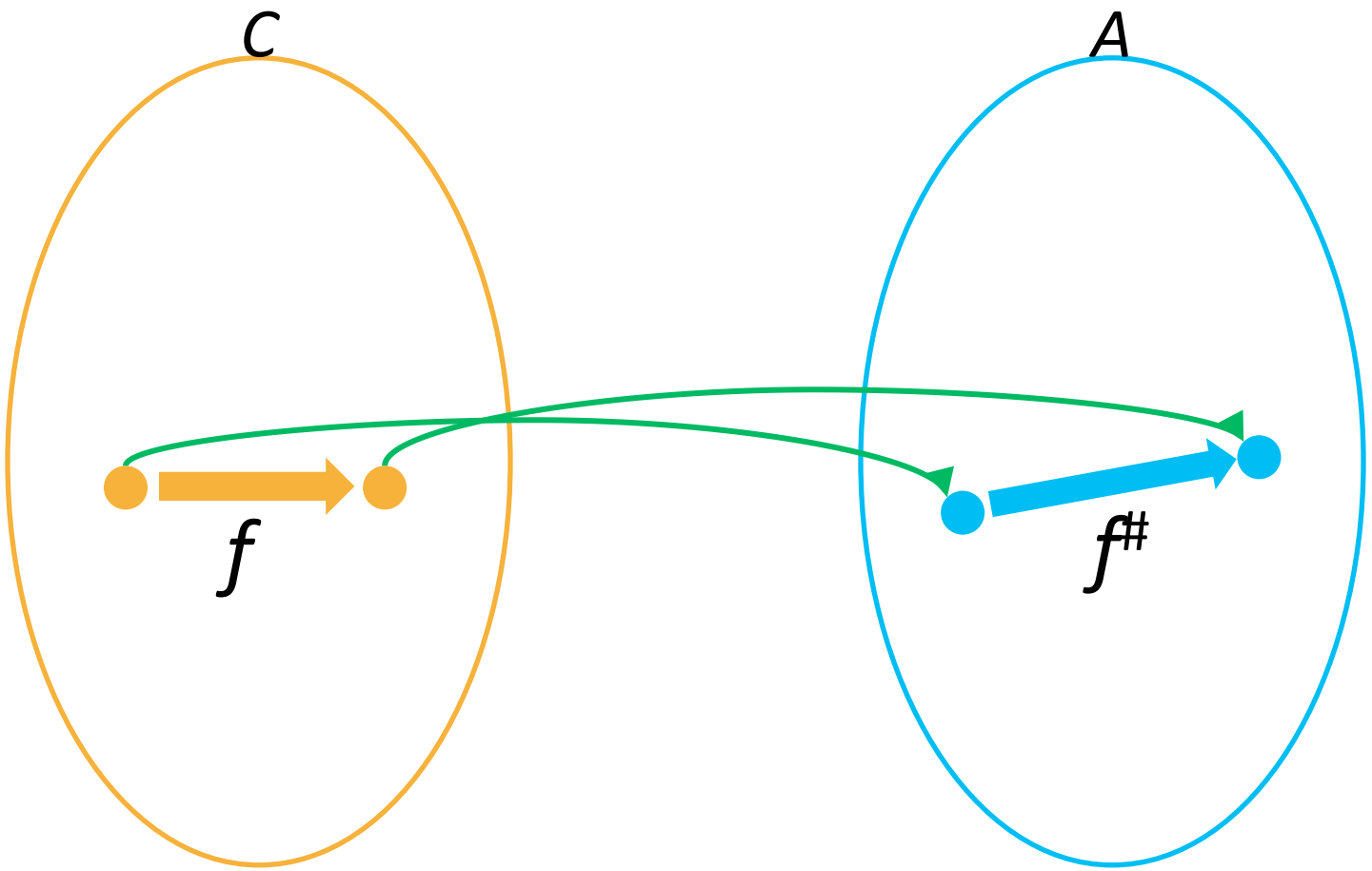
- Define an “appropriate” operational semantics
- Define “collecting” structural operational semantics
- Establish a Galois connection between collecting states and abstract states
- **Local correctness:** show that the abstract interpretation of every atomic statement is sound
  w.r.t. the collecting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

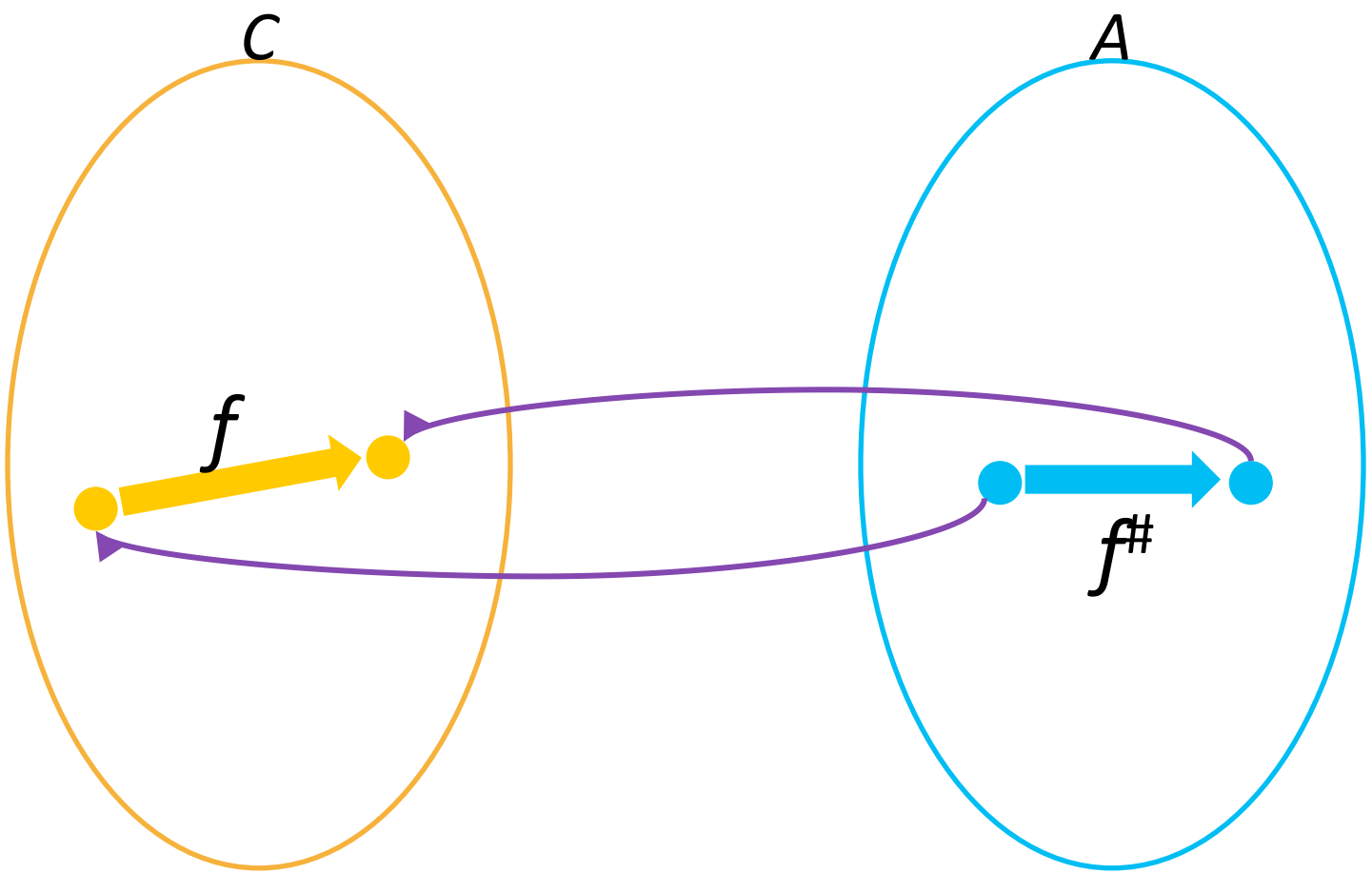
Forward complete transformer

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



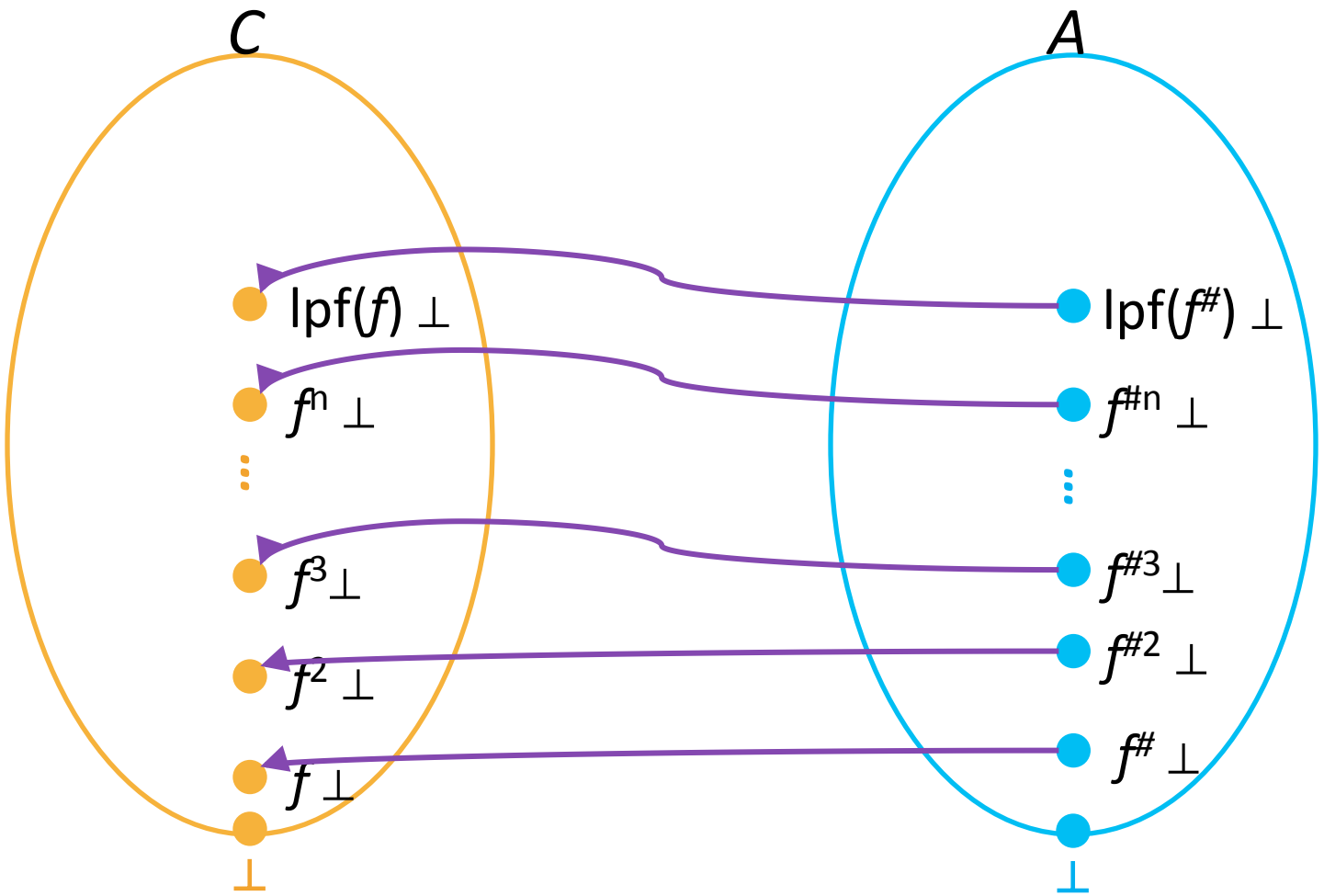
Backward complete transformer

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



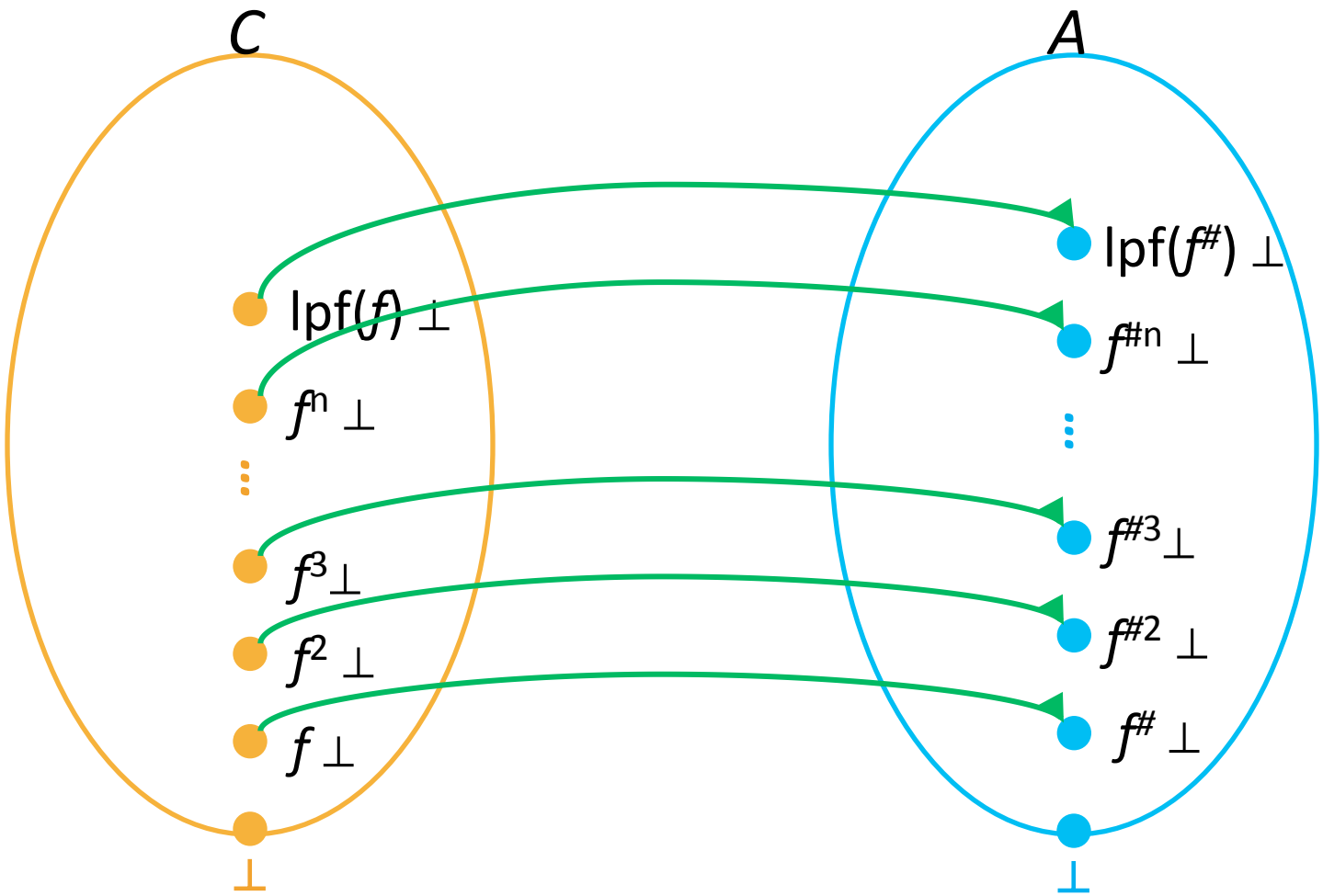
Global (backward) completeness

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

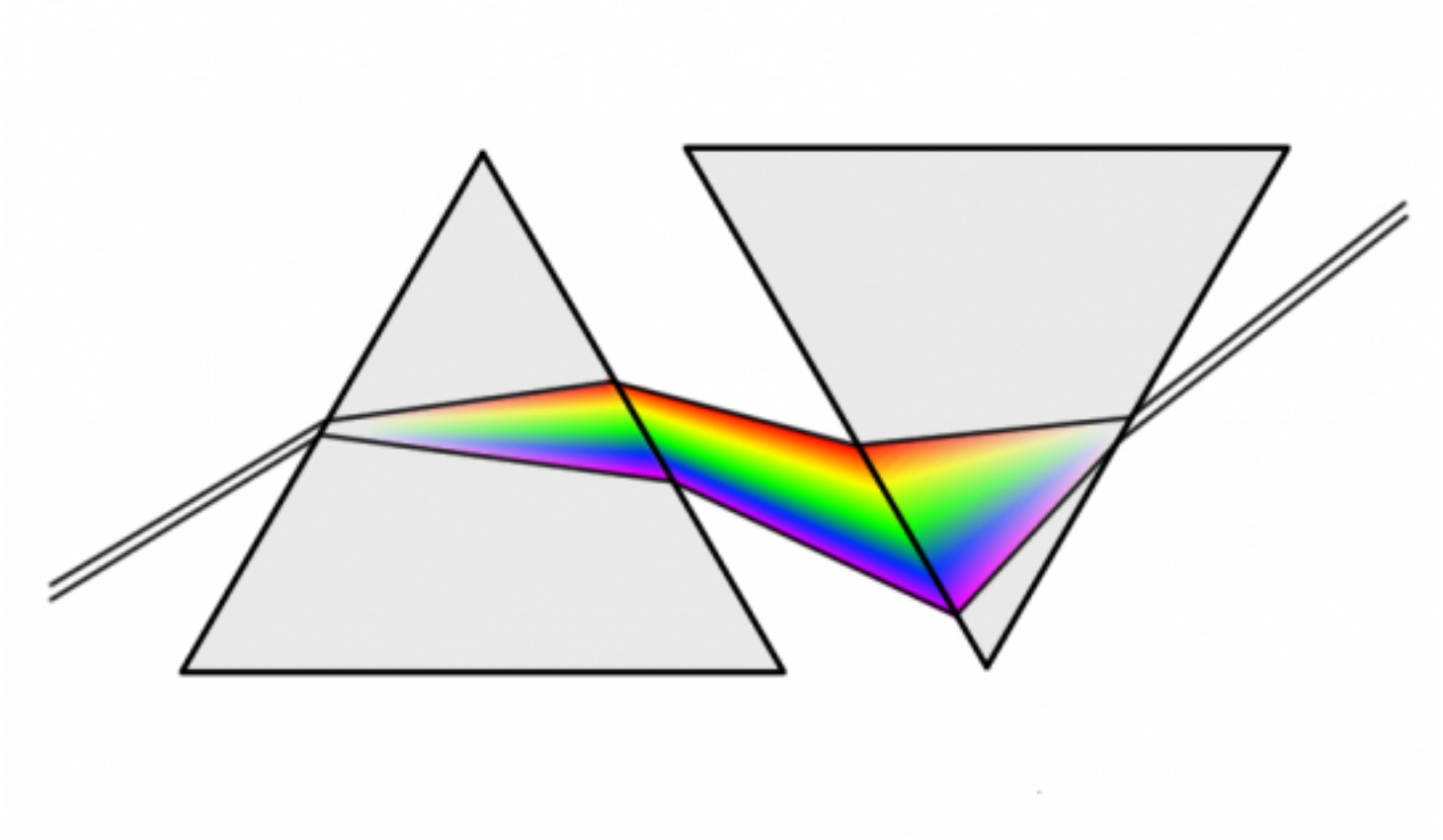


Global (forward) completeness

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



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!");
}
```

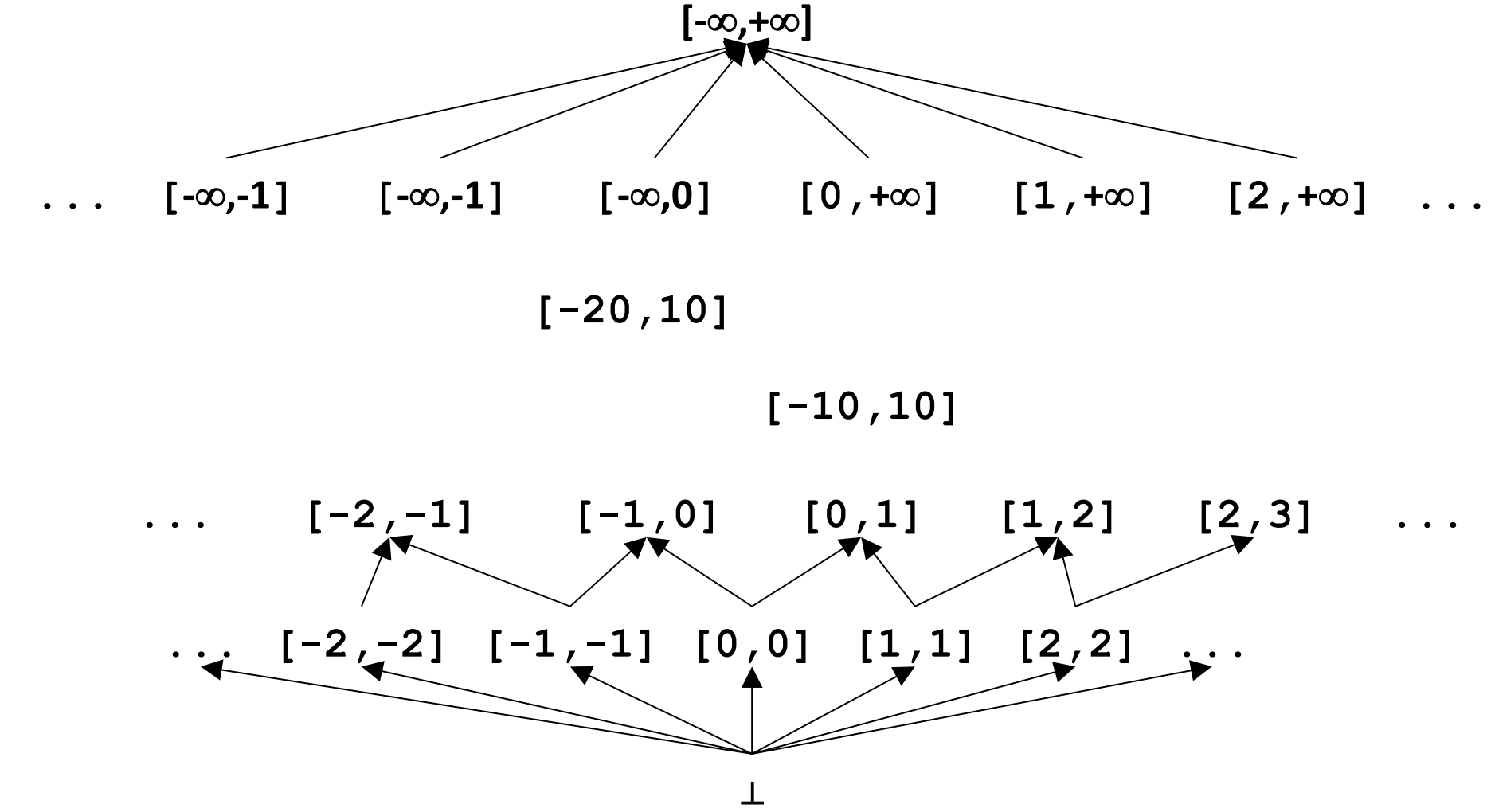
RelProd(CP, VE)

```
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

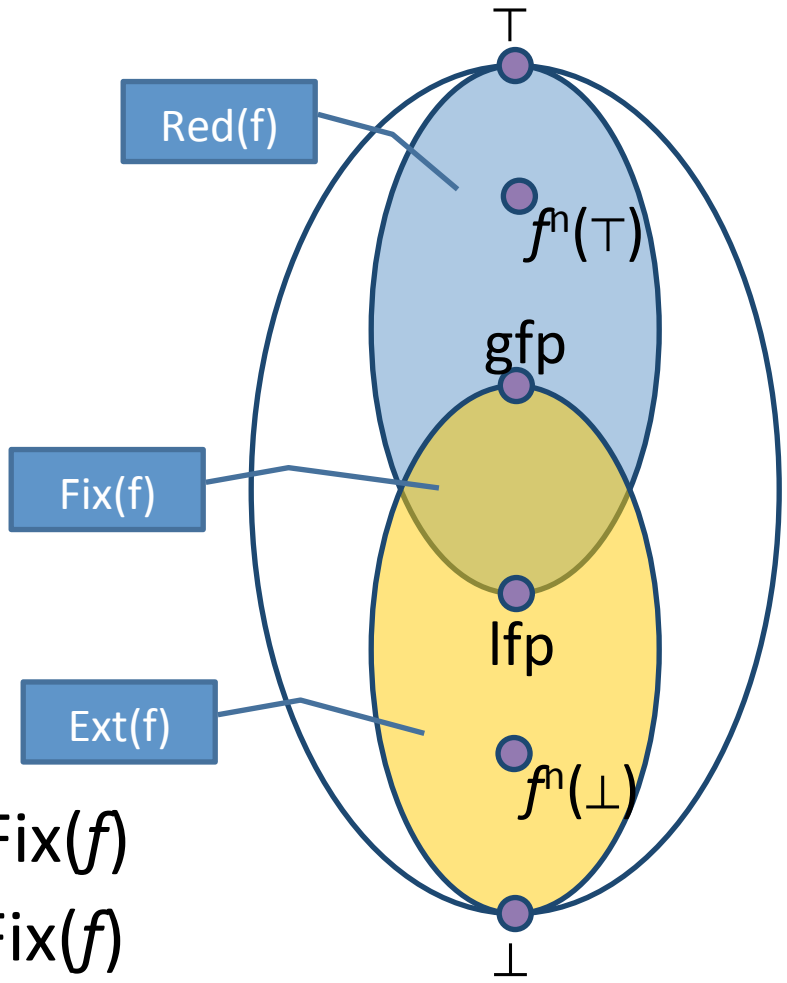
- $[\![\textbf{assume}\ x{=}c]\!]^\#\ S = S \sqcap \{x \geq c, x \leq c\}$
- $[\![\textbf{assume}\ x{<}c]\!]^\#\ S = S \sqcap \{x \leq c\text{-}1\}$
- $[\![\textbf{assume}\ x{=}y]\!]^\#\ S = S \sqcap \{x \geq \mathrm{lb}(S,y), x \leq \mathrm{ub}(S,y)\}$
- $[\![\textbf{assume}\ x{\neq}c]\!]^\#\ S = ?$

assume transformers

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

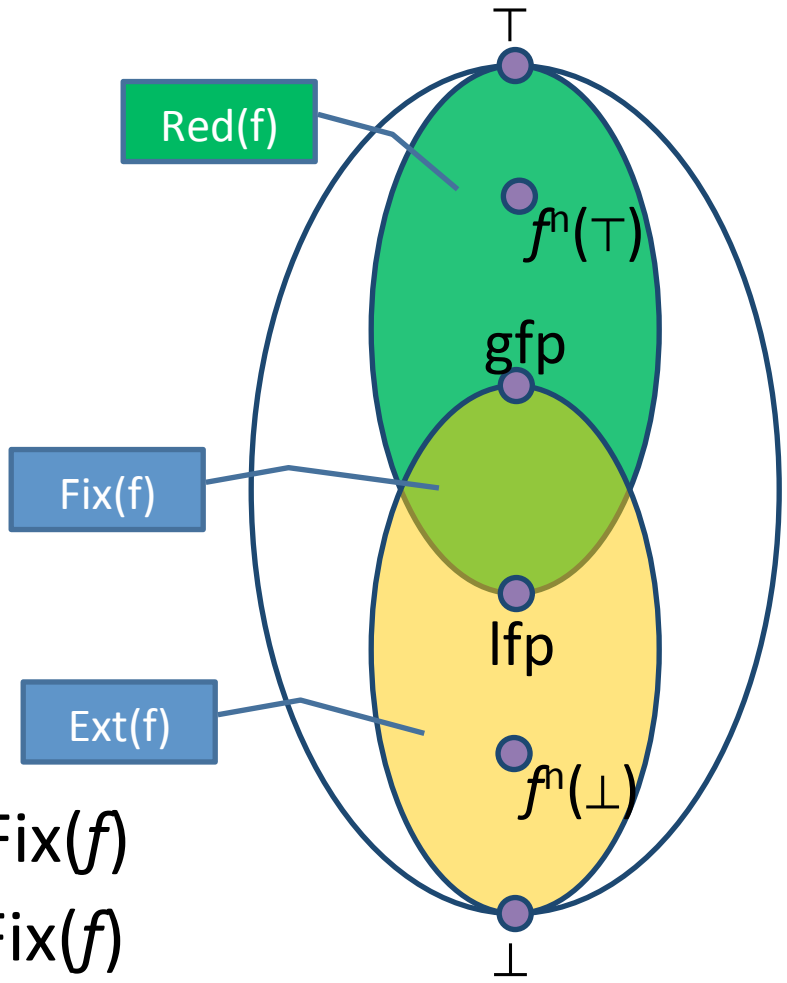
Effect of function *f* on lattice elements

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

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

Resulting algorithm

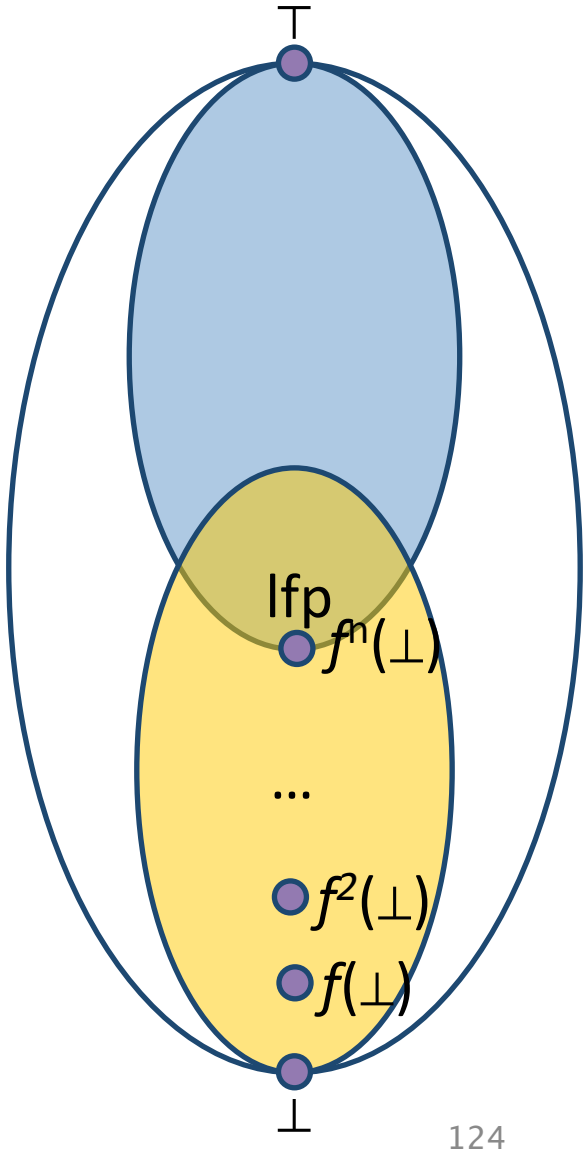
- Kleene's fixed point theorem gives a constructive method for computing the lfp

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!");
}
```

- $R[0] = \{\mathbf{x} \in \mathbf{Z}\}$
  $R[1] = [\![\mathbf{x := 7}]\!]$
  $R[2] = R[1] \cup R[4]$
  $R[3] = R[2] \cap \{s \mid s(x) < 1000\}$
  $R[4] = [\![\mathbf{x := x+1}]\!]\ R[3]$
  $R[5] = R[2] \cap \{s \mid s(x) \geq 1000\}$
  $R[6] = R[5] \cap \{s \mid s(x) \neq 1001\}$

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!");
    }
}
```

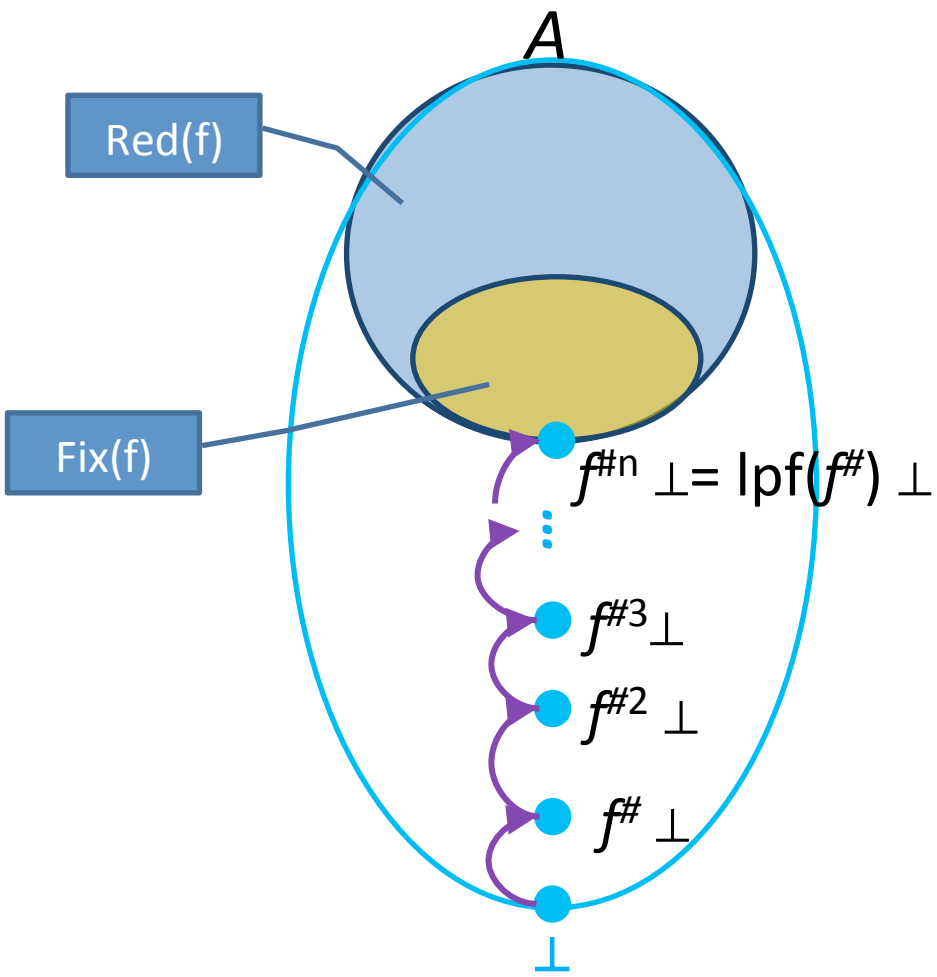
Widening

- 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)

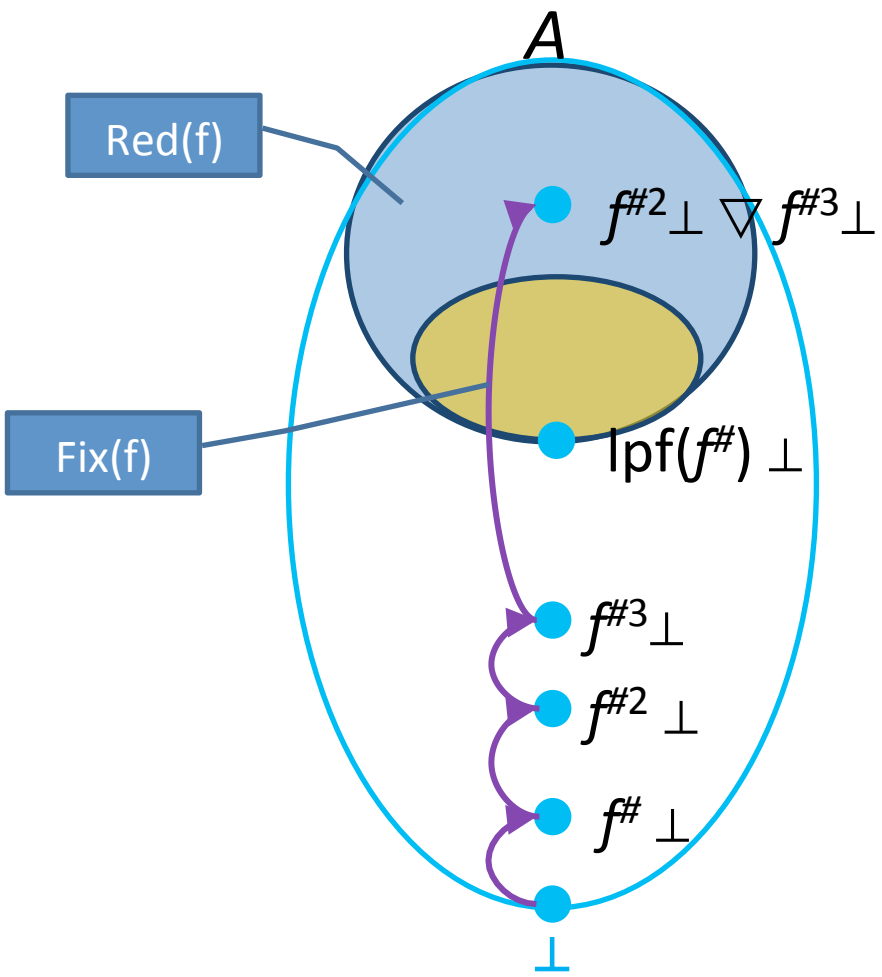
Formal definition

- 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] $\sqcap$ [-∞,999]
  R[4] = R[3] + [1,1]
  R[5] = R[2] $\sqcap$ [1001,+∞]
  R[6] = R[5] $\sqcap$ [999,+∞] $\sqcup$ R[5] $\sqcap$ [1001,+∞]

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

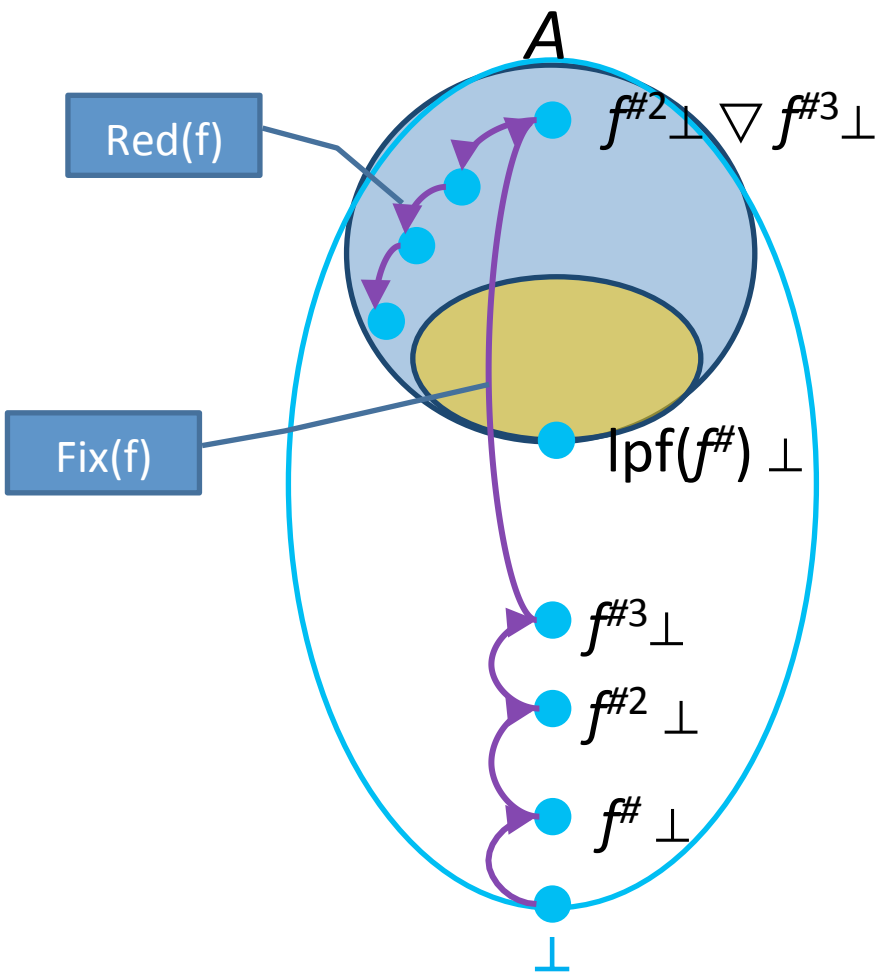
- $[0,1] \triangledown [0,2] = [0, \infty]$
- $[0,2] \triangledown [0,2] = [0,2]$

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] = ⊤
R[1] = [7,7]
R[2] = R[1] ⊔ R[4]
R[2.1] = R[2.1] ▽ R[2]
R[3] = R[2.1] ⊓ [-∞,999]
R[4] = R[3] + [1,1]
R[5] = R[2] ⊓ [1001,+∞]
R[6] = R[5] ⊓ [999,+∞] ⊔ R[5] ⊓ [1001,+∞]

Phase 2:
R[0] = ⊤
R[1] = [7,7]
R[2] = R[1] ⊔ R[4]
R[2.1] = R[2.1] △ R[2]
R[3] = R[2.1] ⊓ [-∞,999]
R[4] = R[3]+[1,1]
R[5] = $R[2]^{\#}$ ⊓ [1000,+∞]
R[6] = R[5] ⊓ [999,+∞] ⊔ R[5] ⊓ [1001,+∞]

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.
```