

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



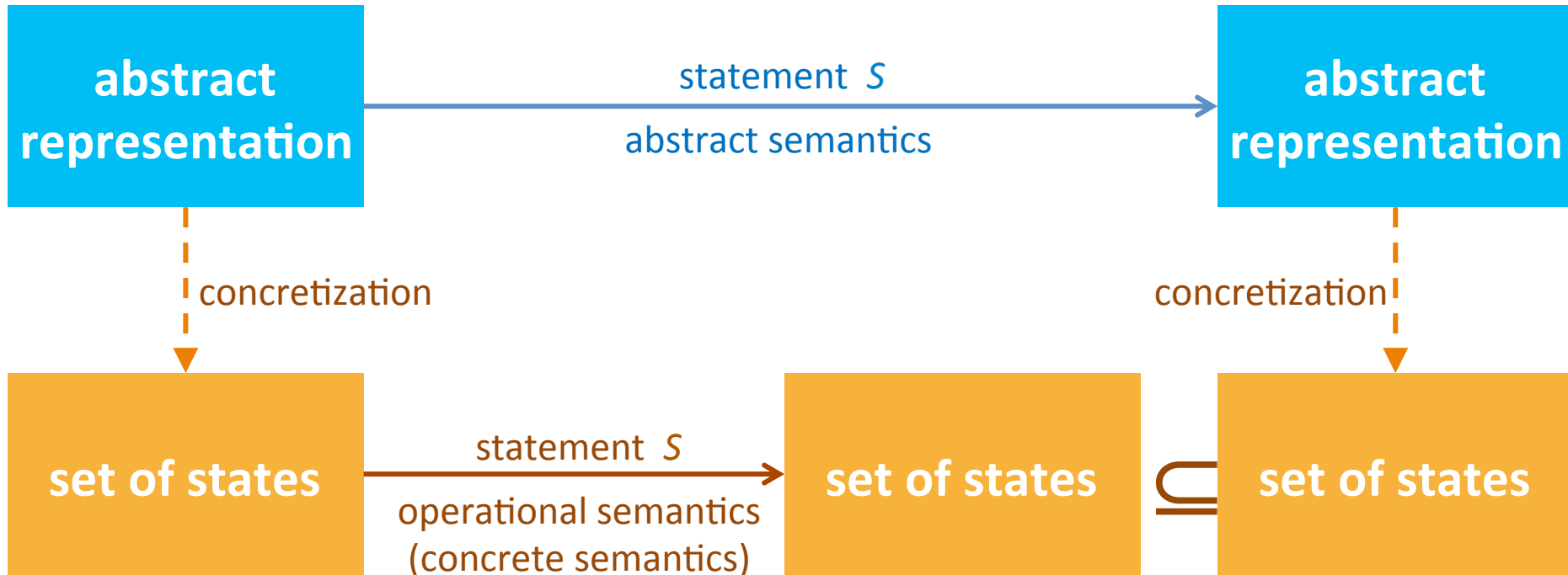
Order-Related Terminology

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

Complete lattices

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

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:

$$x_0 \sqsubseteq x_1 \sqsubseteq \dots \Rightarrow f(x_0) \sqsubseteq f(x_1) \sqsubseteq \dots$$

- It is always true that:

$$\sqcup_i \langle f(x_i) \rangle \sqsubseteq f(\sqcup_i \langle x_i \rangle)$$

- But

$$f(\sqcup_i \langle x_i \rangle) \sqsubseteq \sqcup_i \langle f(x_i) \rangle$$

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

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

- 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
 $W \in [\Sigma_{\perp} \rightarrow \Sigma_{\perp}]$
- 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 $F^k = \lambda x. F(F(\dots F(x)\dots))$ (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 $k \in \mathbb{N}$
 $F^k(\perp) \sqsubseteq x$
 - The least of all fixed points is
 $\sqcup_k F^k(\perp)$
- Proof:
 - i. By induction on k .
 - Base: $F^0(\perp) = \perp \sqsubseteq x$
 - Induction step: $F^{k+1}(\perp) = F(F^k(\perp)) \sqsubseteq F(x) = x$
 - ii. It suffices to show that $\sqcup_k F^k(\perp)$ is a fixed-point
 - $F(\sqcup_k F^k(\perp)) = \sqcup_k F^{k+1}(\perp) = \sqcup_k F^k(\perp)$

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

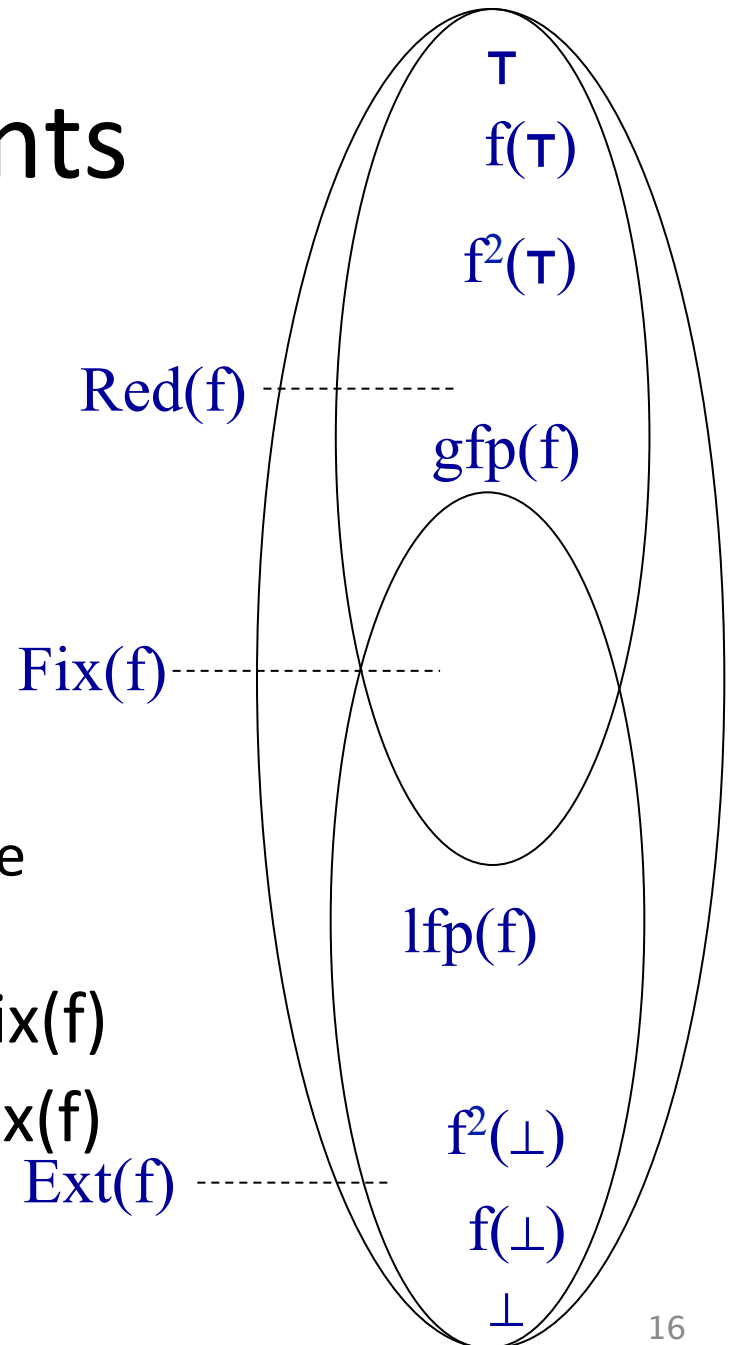
- Let (D, \sqsubseteq) be a partial order
- D is a **complete lattice** if every subset has both greatest lower bounds and least upper bounds

Knaster-Tarski Theorem

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

Fixed Points

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



Collecting semantics

○ label10:

● if $x \leq 0$ goto label11

● $x := x - 1$ $\dots [x \mapsto 3] [x \mapsto 2] [x \mapsto 1]$

● goto label10

● label11:

$\dots [x \mapsto 3] [x \mapsto 2] [x \mapsto 2] [x \mapsto -1] [x \mapsto 0] [x \mapsto 1]$ ●

$\dots [x \mapsto -2] [x \mapsto -1]$

exit

$[x \mapsto 0] [x \mapsto 1]$ ●

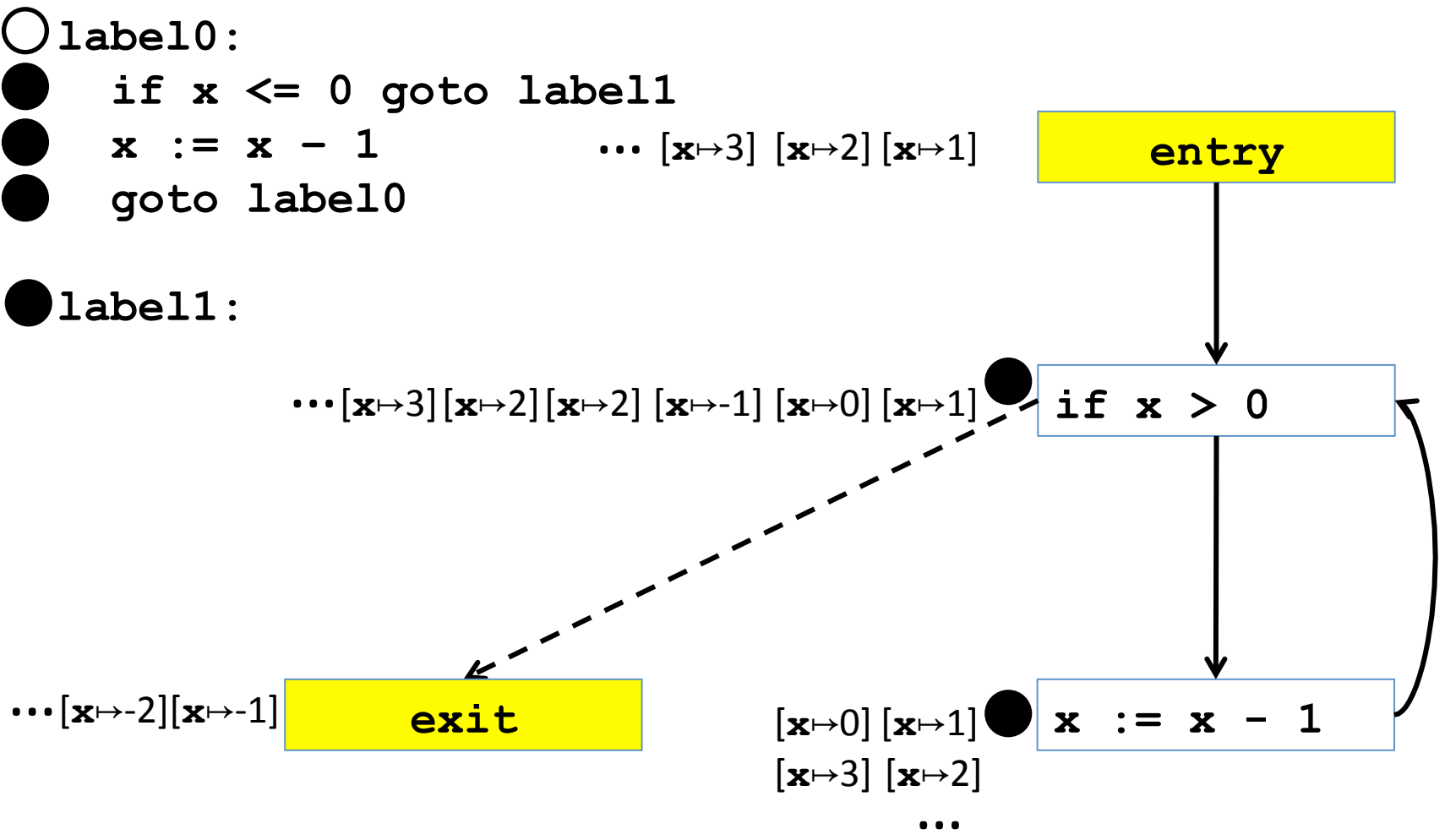
$[x \mapsto 3] [x \mapsto 2]$

\dots

entry

if $x > 0$

$x := x - 1$



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

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 :

$$L_v = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$$

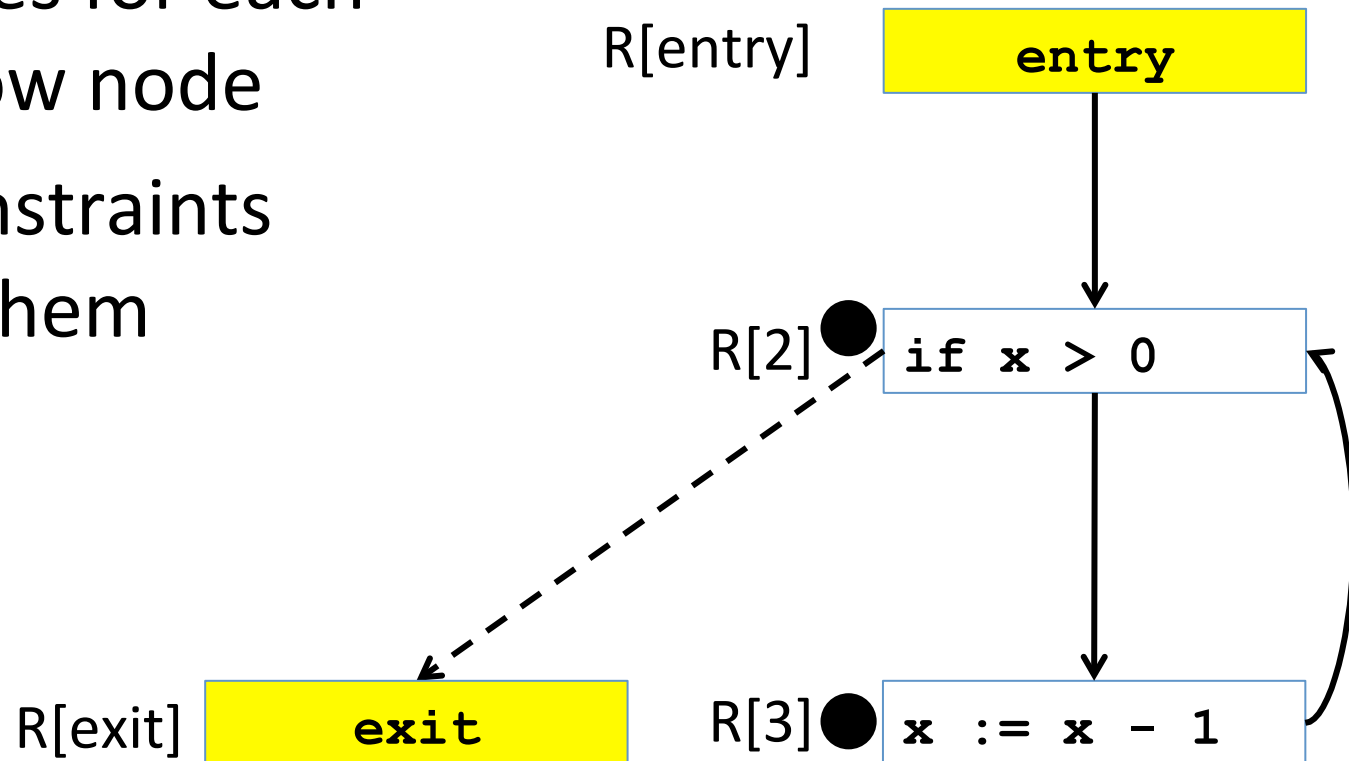
- Lattice for entire control-flow graph with nodes V :

$$L_{\text{CFG}} = \text{Map}(V, L_v)$$

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

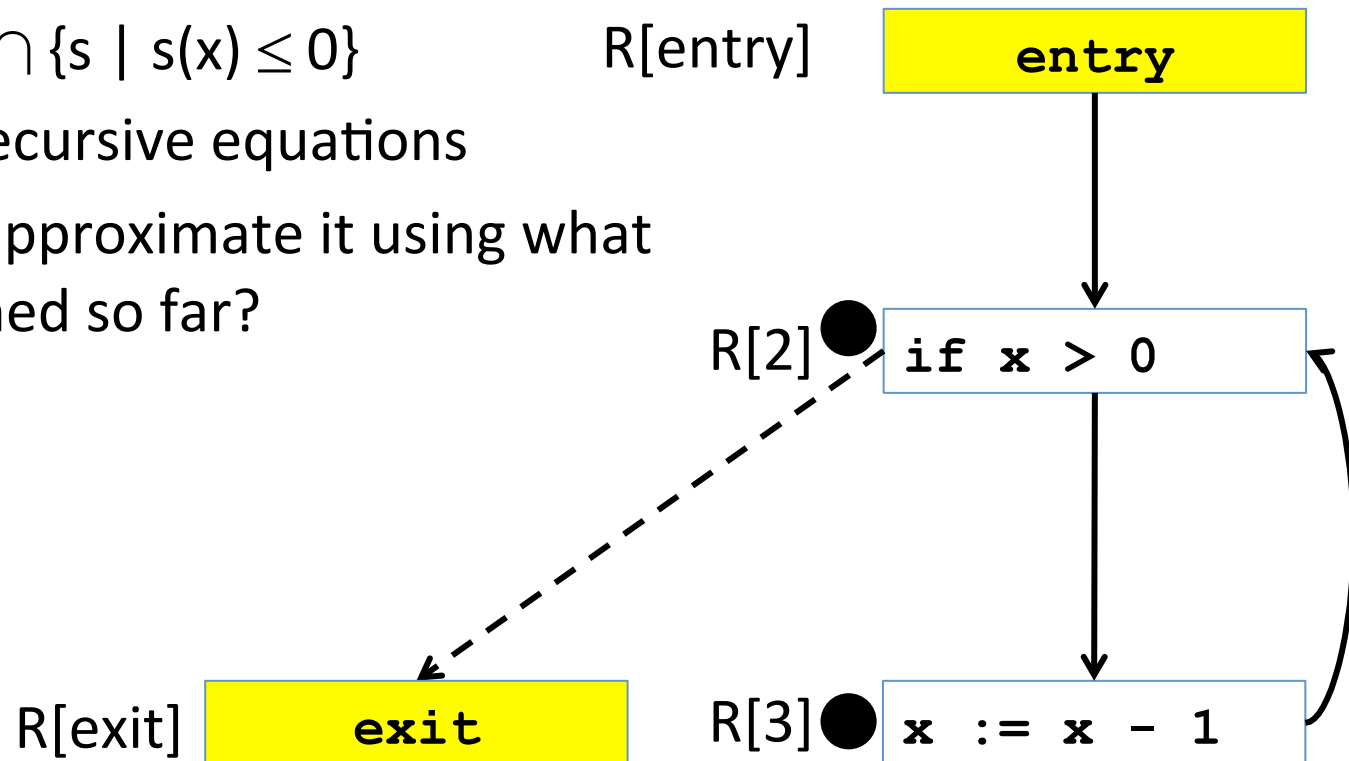
Equational definition of the semantics

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



Equational definition of the semantics

- $R[2] = R[\text{entry}] \cup \llbracket \mathbf{x} := \mathbf{x} - 1 \rrbracket R[3]$
- $R[3] = R[2] \cap \{s \mid s(x) > 0\}$
- $R[\text{exit}] = R[2] \cap \{s \mid s(x) \leq 0\}$
- A system of recursive equations
- How can we approximate it using what we have learned so far?

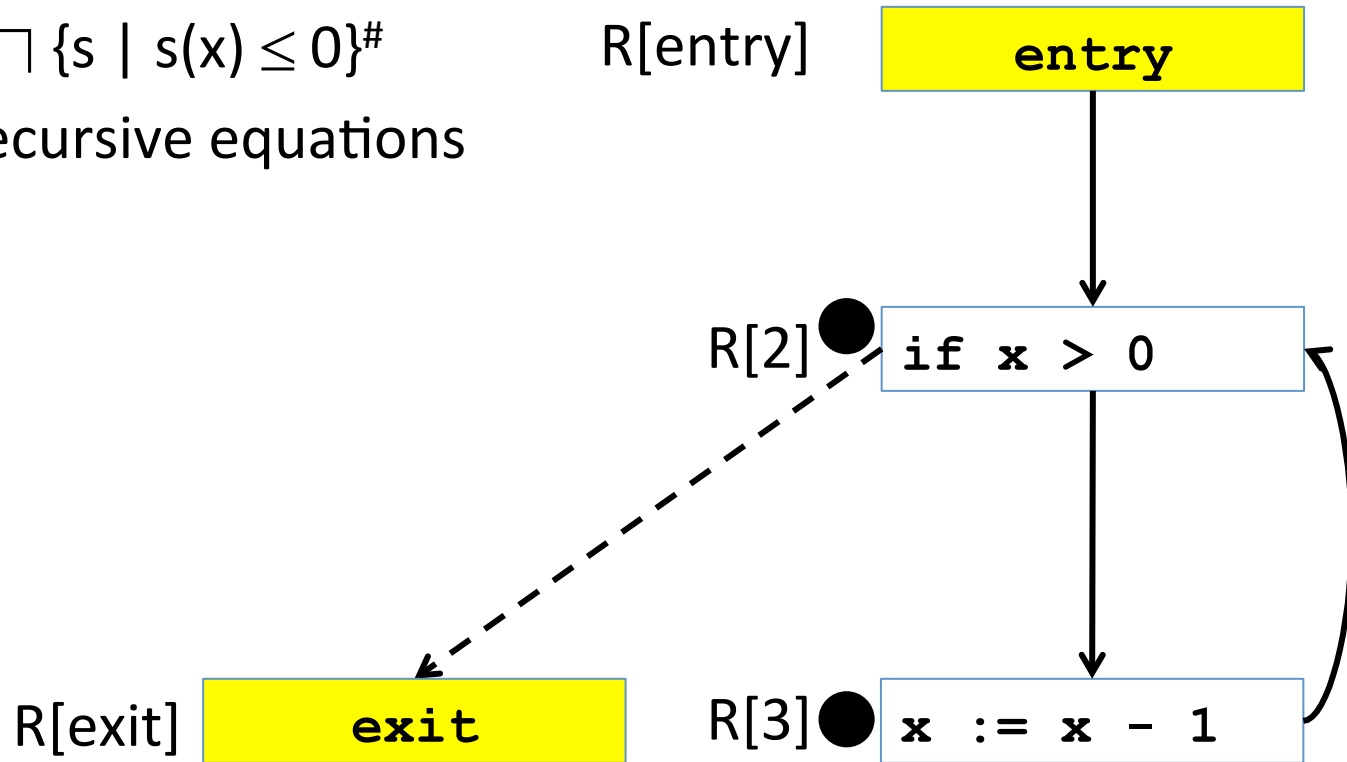


An abstract semantics

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

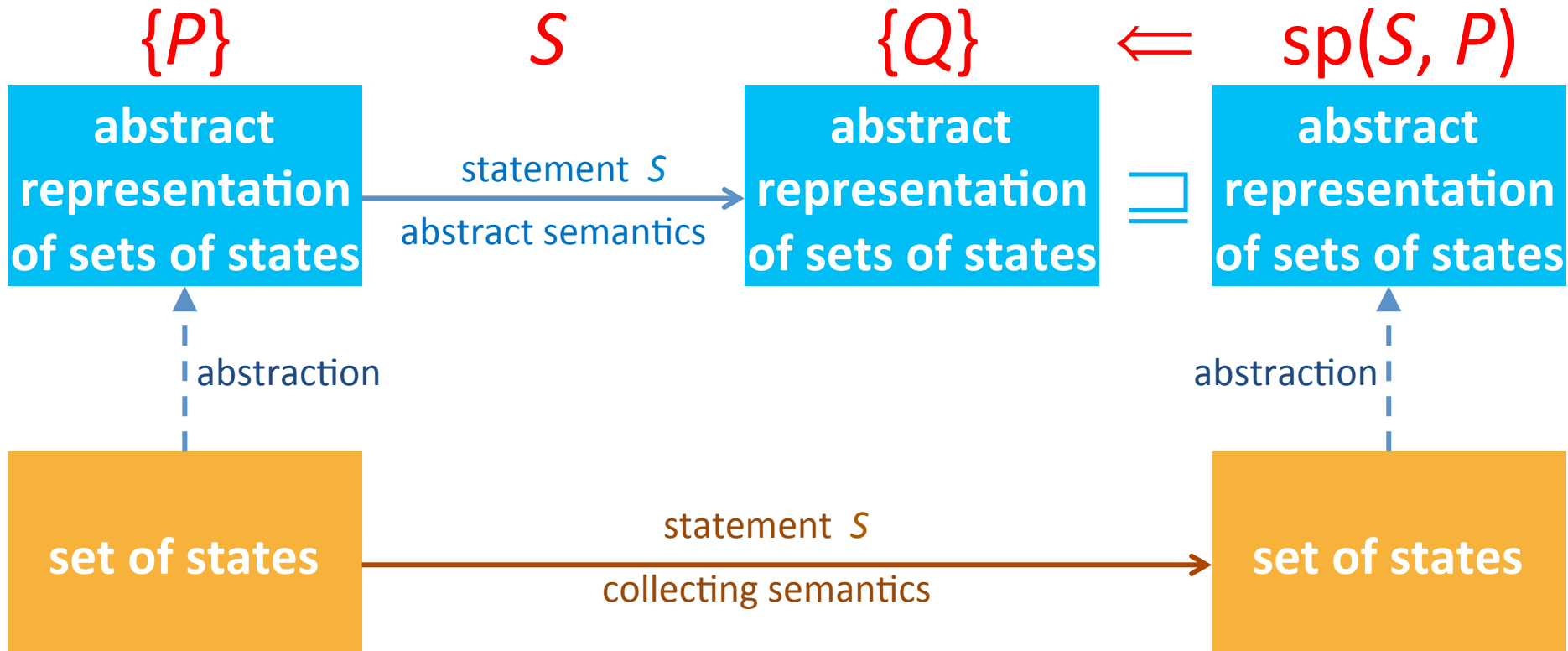
Abstract transformer for $\mathbf{x} := \mathbf{x} - 1$

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

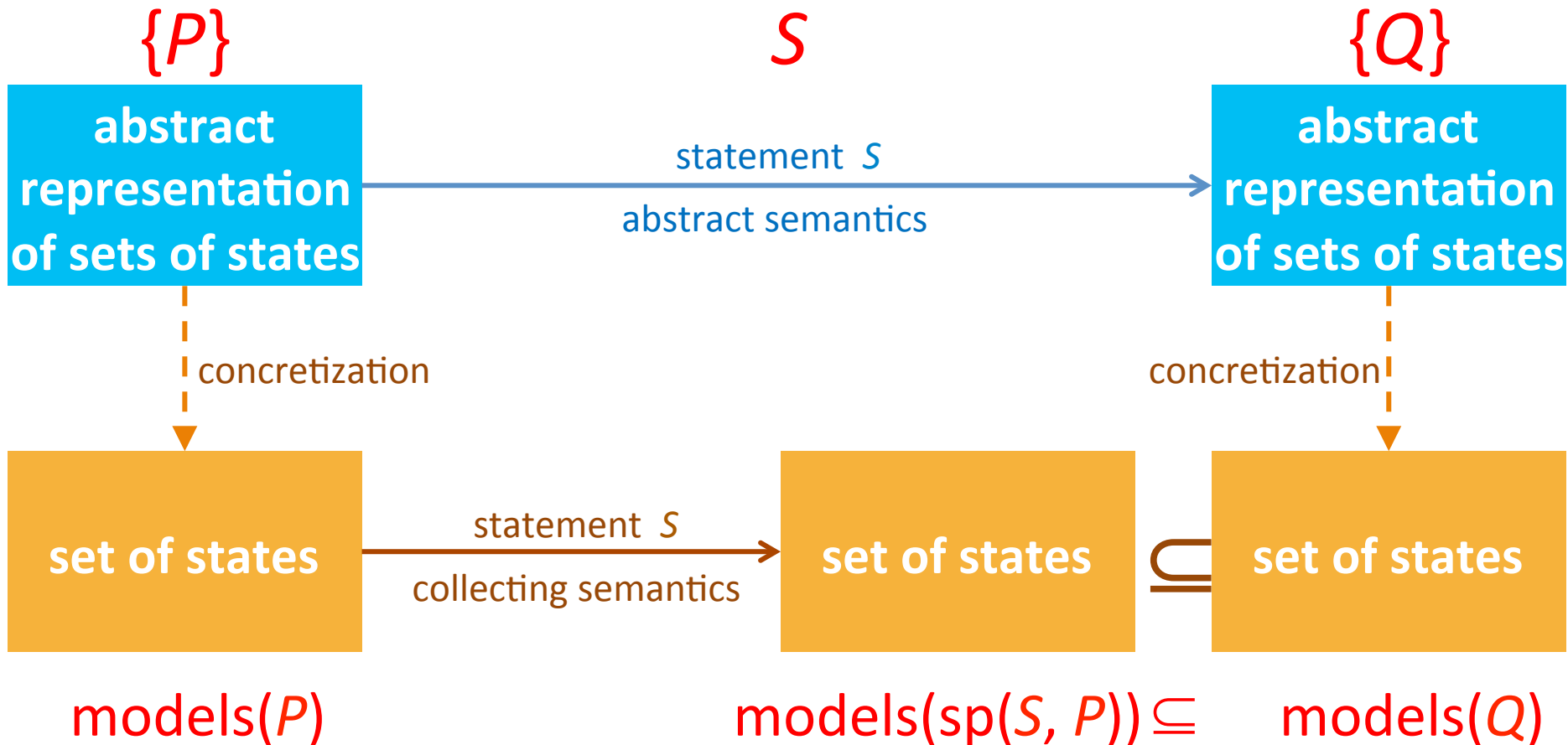


Abstract interpretation via abstraction

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 :

$$L_v = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$$

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

$$L_{\text{CFG}} = \text{Map}(V, L_v)$$

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

Equation systems in general

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

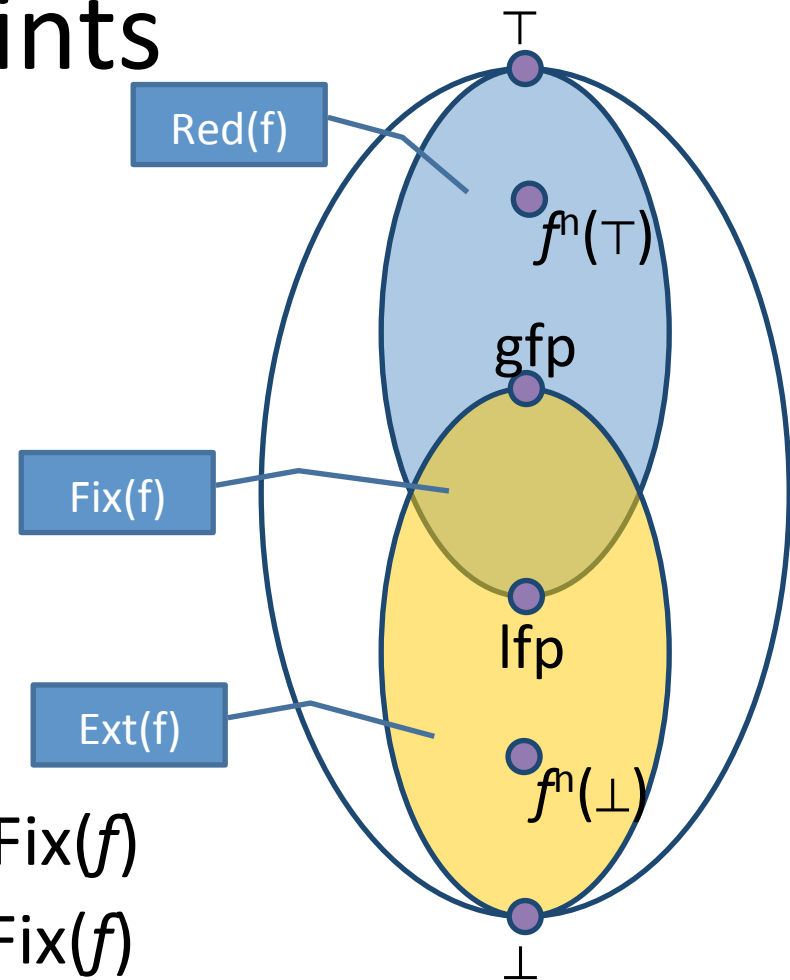
- Join: $f(X, Y) = X \sqcup Y$
Prove it!
- For a set X and any function g
 $F(X) = \{ g(x) \mid x \in X \}$
Prove it!
- Notice that the collecting semantics function is defined in terms of
 - Join (set union)
 - Semantic function for atomic statements lifted to sets of states

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 $x \sqsupseteq f(x)$

Fixed-points

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$
- $f : D \rightarrow D$ **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]**
 - $\text{lfp}(f) = \sqcap \text{Fix}(f) = \sqcap \text{Red}(f) \in \text{Fix}(f)$
 - $\text{gfp}(f) = \sqcup \text{Fix}(f) = \sqcup \text{Ext}(f) \in \text{Fix}(f)$

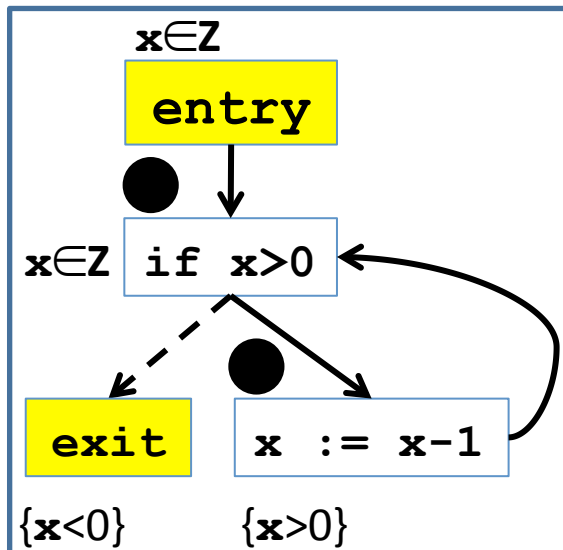


1. Does a solution always exist? Yes
2. If so, is it unique? No, but it has least/greatest solutions
3. If so, is it computable? Under some conditions...

Fixed point example for program

- $R[0] = \{x \in \mathbb{Z}\}$
 $R[1] = R[0] \cup R[4]$
 $R[2] = R[1] \cap \{s \mid s(x) > 0\}$
 $R[3] = R[1] \cap \{s \mid s(x) \leq 0\}$
 $R[4] = \llbracket x := x - 1 \rrbracket R[2]$

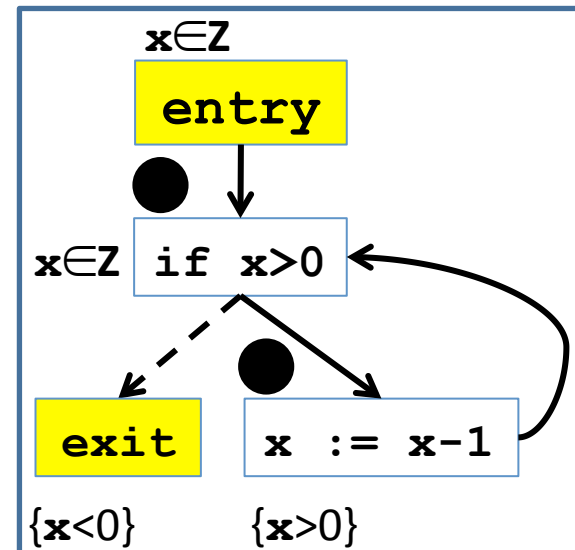
d



=



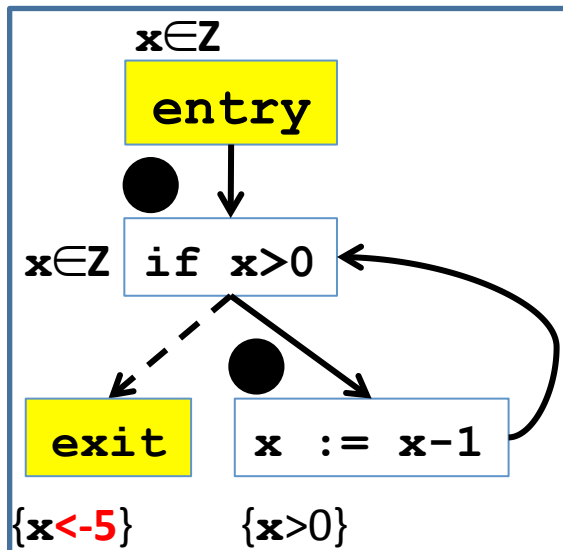
F(d) : Fixed-point



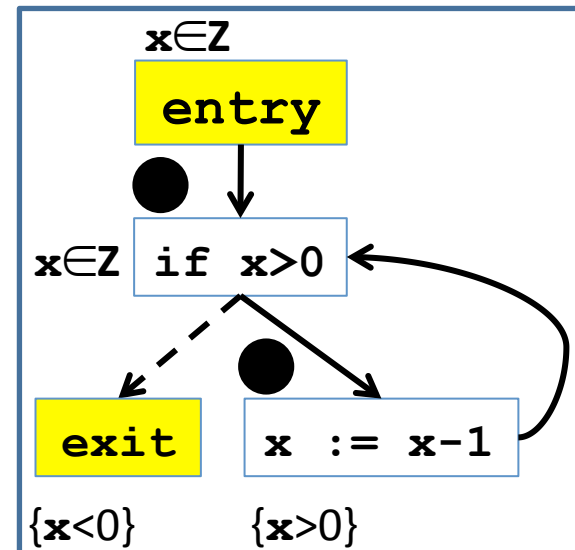
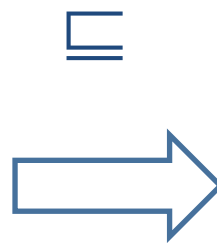
Fixed point example for program

- $R[0] = \{\mathbf{x} \in \mathbb{Z}\}$
- $R[1] = R[0] \cup R[4]$
- $R[2] = R[1] \cap \{s \mid s(x) > 0\}$
- $R[3] = R[1] \cap \{s \mid s(x) \leq 0\}$
- $R[4] = \llbracket \mathbf{x} := \mathbf{x} - 1 \rrbracket R[2]$

d



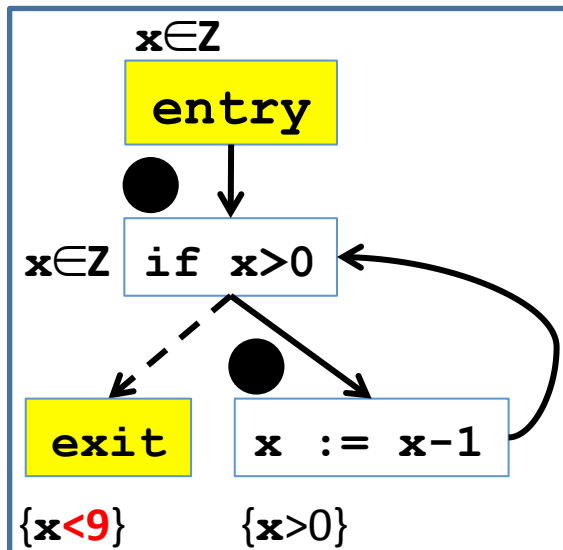
$F(d)$: pre Fixed-point



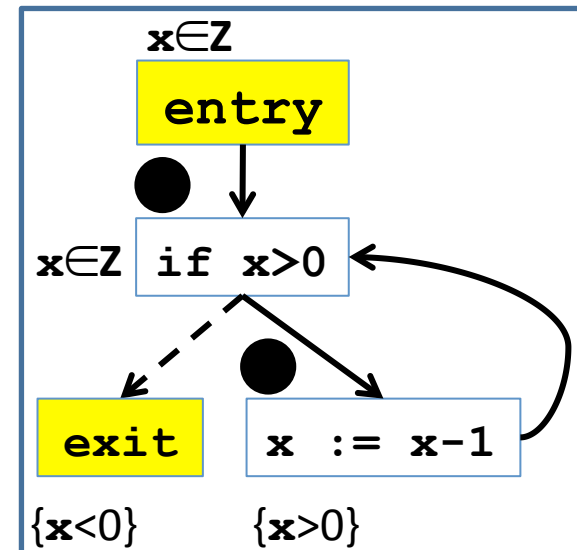
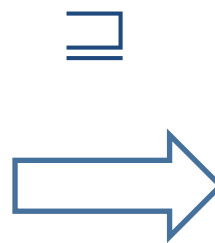
Fixed point example for program

- $R[0] = \{\mathbf{x} \in \mathbb{Z}\}$
- $R[1] = R[0] \cup R[4]$
- $R[2] = R[1] \cap \{s \mid s(x) > 0\}$
- $R[3] = R[1] \cap \{s \mid s(x) \leq 0\}$
- $R[4] = \llbracket \mathbf{x} := \mathbf{x} - 1 \rrbracket R[2]$

d



$F(d)$: post Fixed-point



Continuity and ACC condition

- Let $L = (D, \sqsubseteq, \sqcup, \perp)$ 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:

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

Fixed-point theorem [Kleene]

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

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

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

Proof:

Resulting algorithm

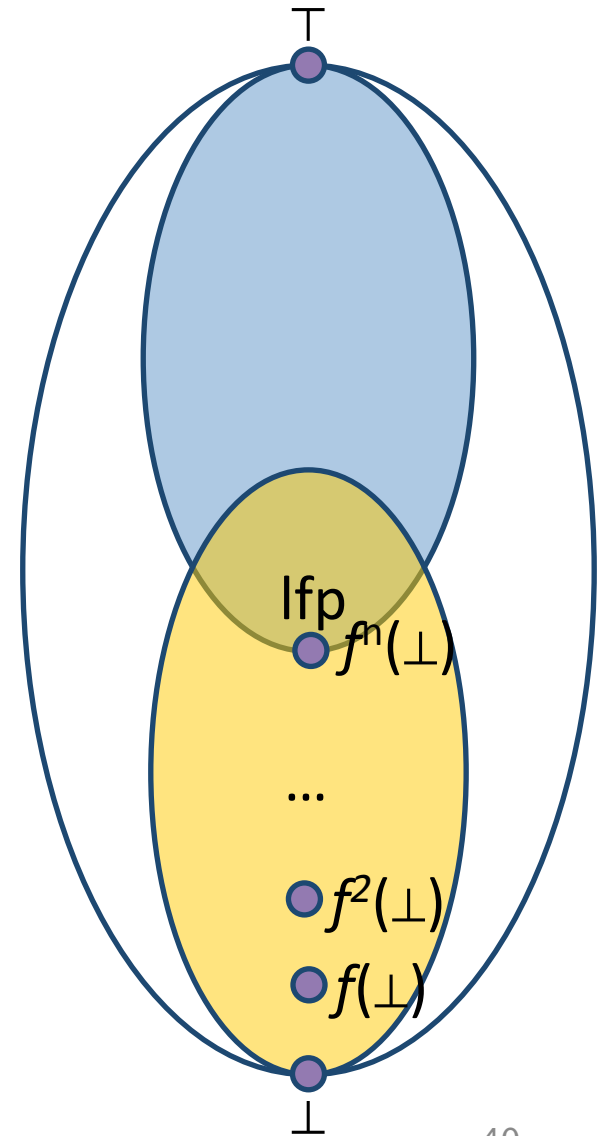
- Kleene's fixed point theorem gives a constructive method for computing the lfp

Mathematical definition

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

Algorithm

```
 $d := \perp$   
while  $f(d) \neq d$  do  
     $d := d \sqcup f(d)$   
return  $d$ 
```



Chaotic iteration

- Input:
 - A cpo $L = (D, \sqsubseteq, \sqcup, \perp)$ satisfying ACC
 - $L^n = L \times L \times \dots \times L$
 - 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: $\text{lfp}(f)$
- A worklist-based algorithm

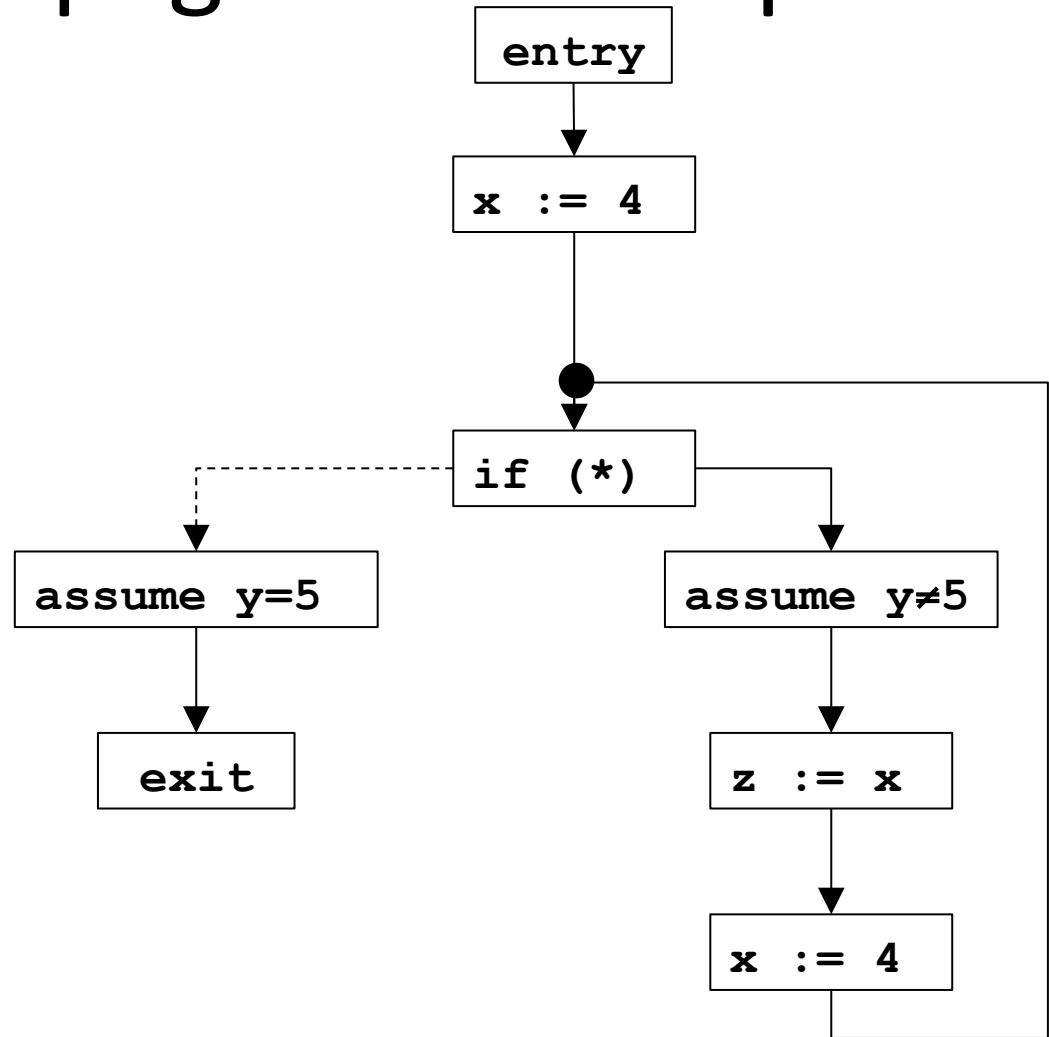
```
for i:=1 to n do
  X[i] :=  $\perp$ 
WL = {1,...,n}
while WL  $\neq \emptyset$  do
  i := pop WL // choose index non-deterministically
  N := F[i](X)
  if N  $\neq$  X[i] then
    X[i] := N
    add all the indexes that directly depend on i to WL
    (X[j] depends on X[i] if F[j] contains X[i])
return 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: $L = (D, \sqsubseteq, \sqcup, \perp)$
- Define variables $R[0, \dots, n]$ 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:
$$v_{\text{out}} = F[v](\sqcup u \mid (u, v) \text{ is a CFG edge})$$
 - Make sure each $F[v]$ is monotone
- Variable dependence determined by outgoing edges in CFG

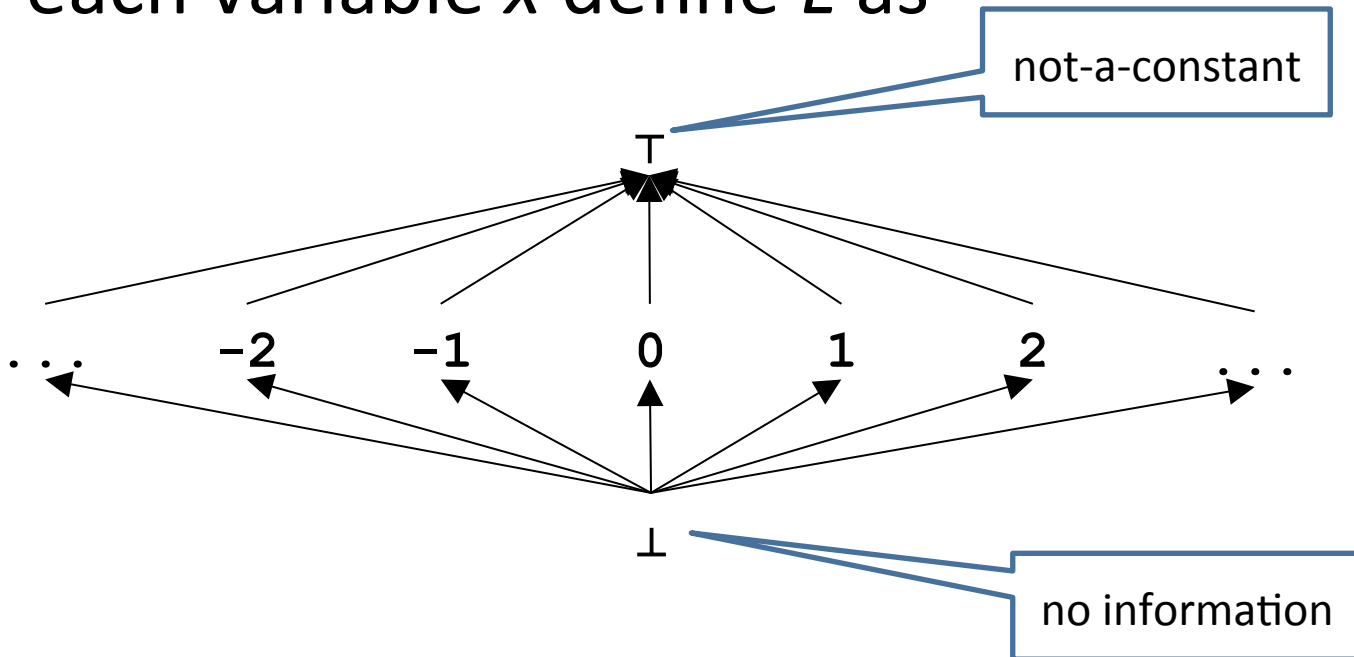
Constant propagation example

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



Constant propagation lattice

- For each variable x define L as

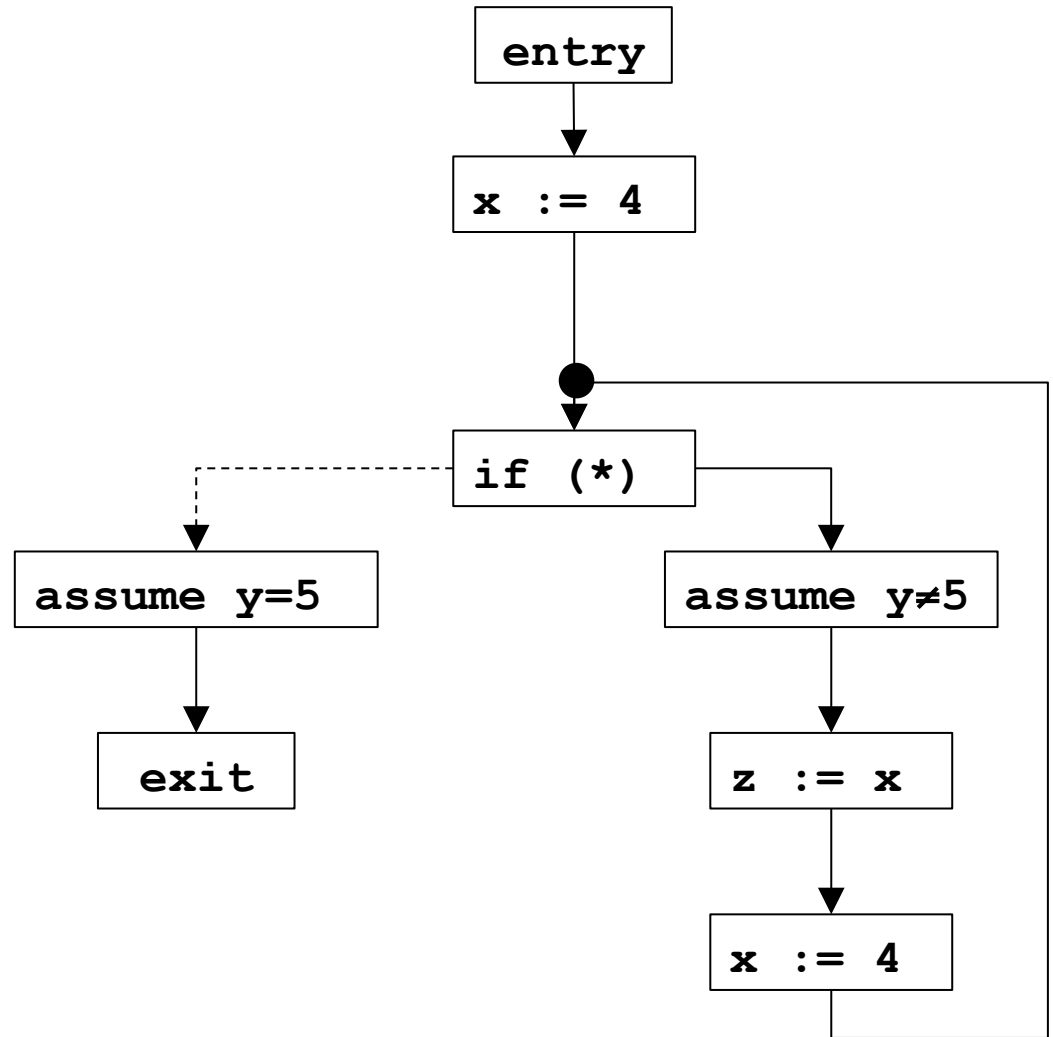


- For a set of program variables $\text{Var} = x_1, \dots, x_n$

$$L^n = L \times L \times \dots \times L$$

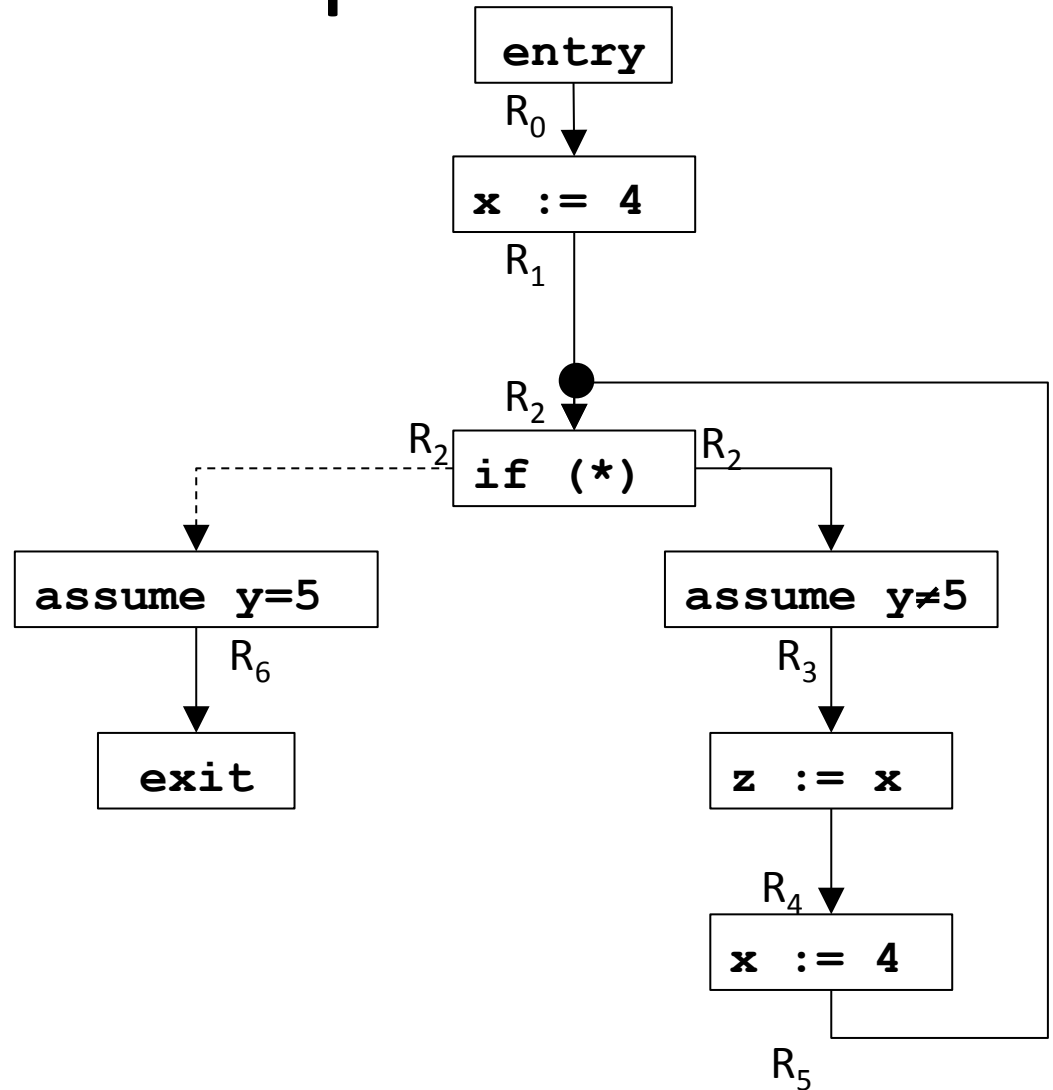
Write down variables

```
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

$$R_0 = \text{State}$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket R_0$$

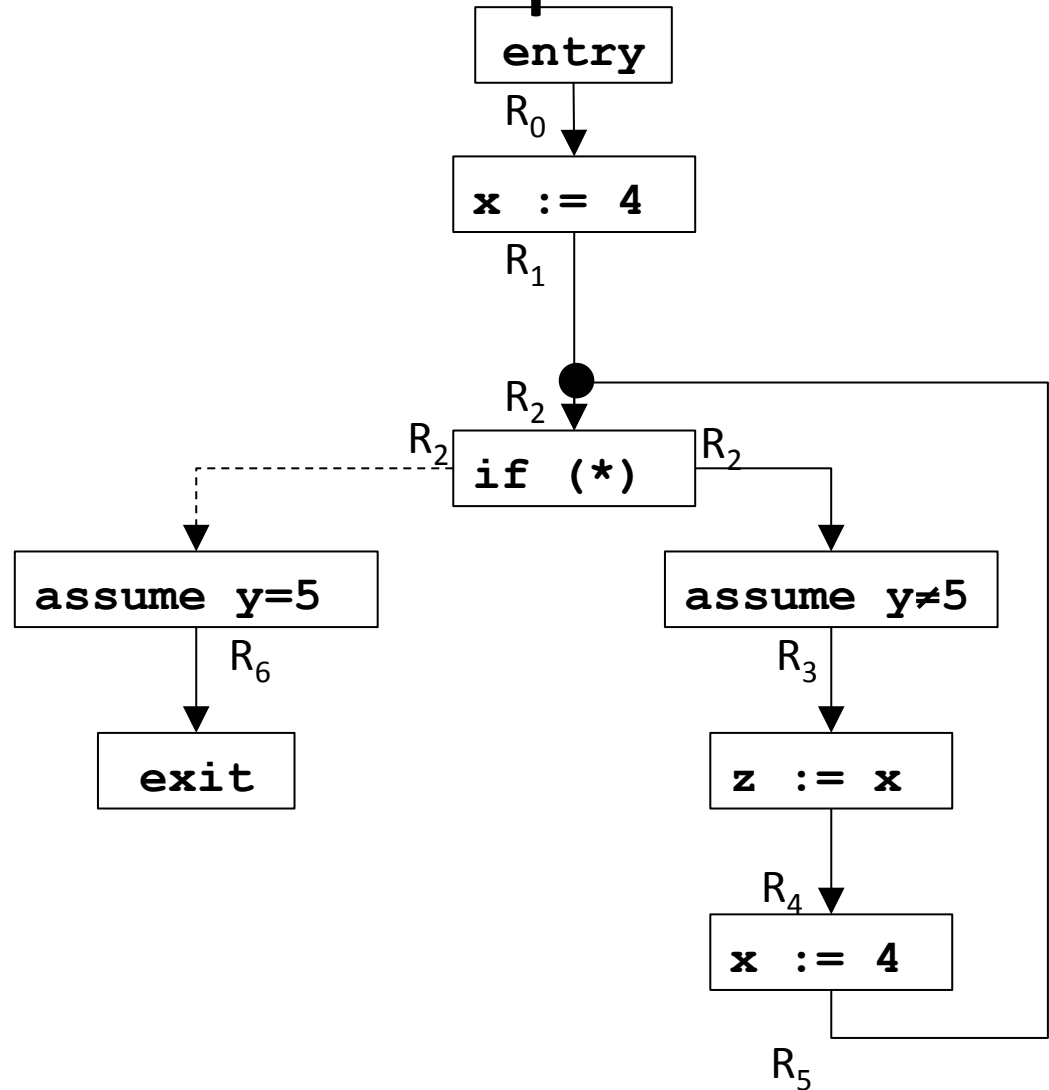
$$R_2 = R_1 \cup R_5$$

$$R_3 = \llbracket \text{assume } \mathbf{y} \neq 5 \rrbracket R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket R_4$$

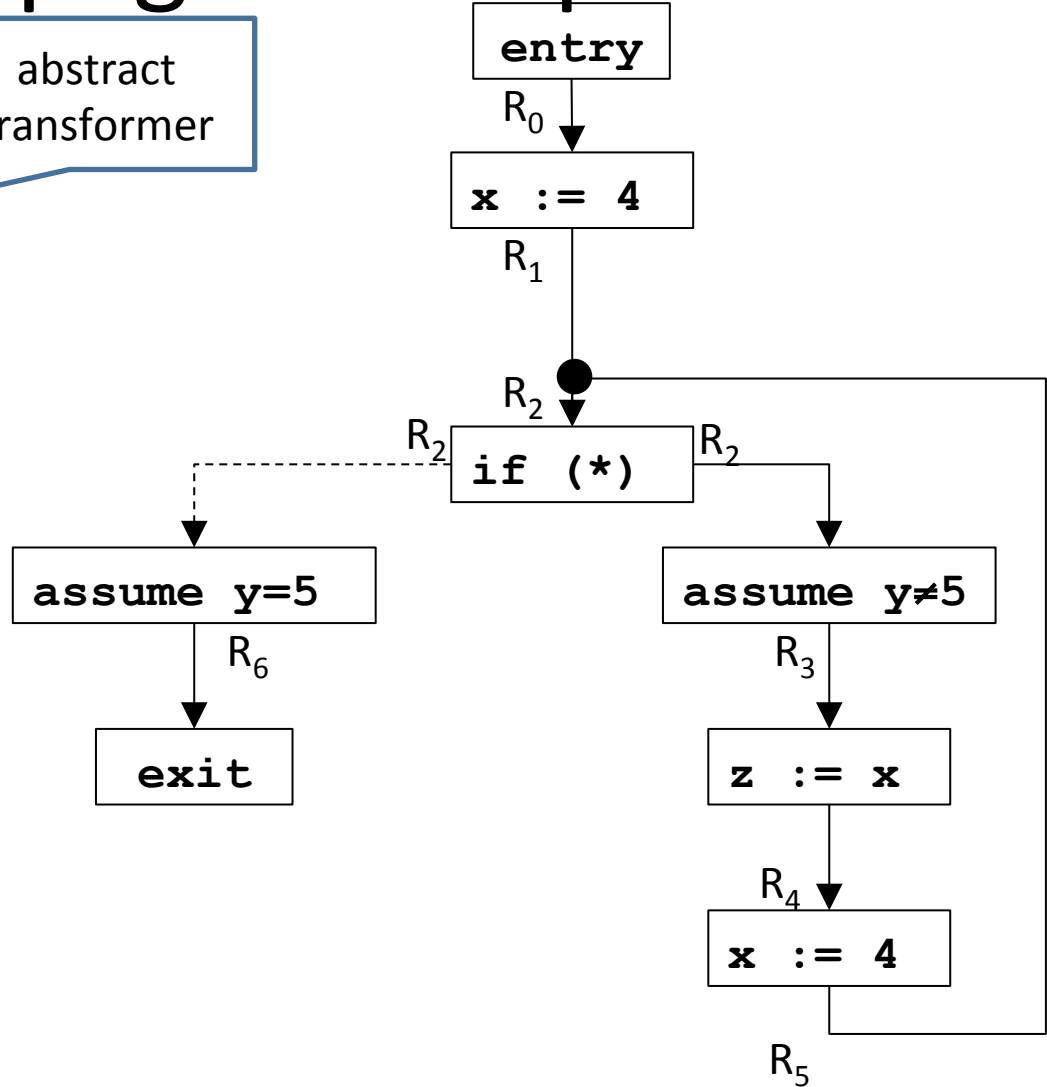
$$R_6 = \llbracket \text{assume } \mathbf{y} = 5 \rrbracket R_2$$



Constant propagation equations

$$\begin{aligned}
 R_0 &= \top \\
 R_1 &= \llbracket \mathbf{x} := 4 \rrbracket^\# R_0 \\
 R_2 &= R_1 \sqcup R_5 \\
 R_3 &= \llbracket \mathbf{assume\ } y \neq 5 \rrbracket^\# R_2 \\
 R_4 &= \llbracket \mathbf{z} := \mathbf{x} \rrbracket^\# R_3 \\
 R_5 &= \llbracket \mathbf{x} := 4 \rrbracket^\# R_4 \\
 R_6 &= \llbracket \mathbf{assume\ } y = 5 \rrbracket^\# R_2
 \end{aligned}$$

abstract transformer



Abstract operations for CP

CP lattice for a single variable

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket^\# R_0$$

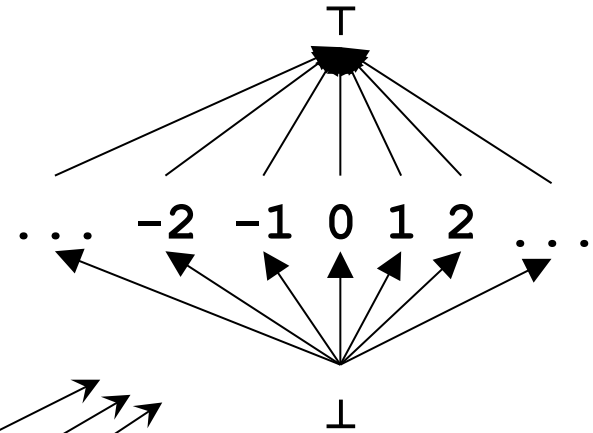
$$R_2 = R_1 \sqcup R_5$$

$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket^\# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket^\# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket^\# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket^\# R_2$$



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

$$\llbracket \mathbf{x} := 4 \rrbracket^\# (v_x, v_y, v_z) = (4, v_y, v_z)$$

$$\llbracket \mathbf{z} := \mathbf{x} \rrbracket^\# (v_x, v_y, v_z) = (v_x, v_y, v_x)$$

$$\llbracket \mathbf{assume} \ y \neq 5 \rrbracket^\# (v_x, v_y, v_z) = (v_x, v_y, v_x)$$

$$\llbracket \mathbf{assume} \ y = 5 \rrbracket^\# (v_x, v_y, v_z) = \text{if } v_y = k \neq 5 \text{ then } (\perp, \perp, \perp) \text{ else } (v_x, 5, v_z)$$

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

Chaotic iteration for CP: initialization

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

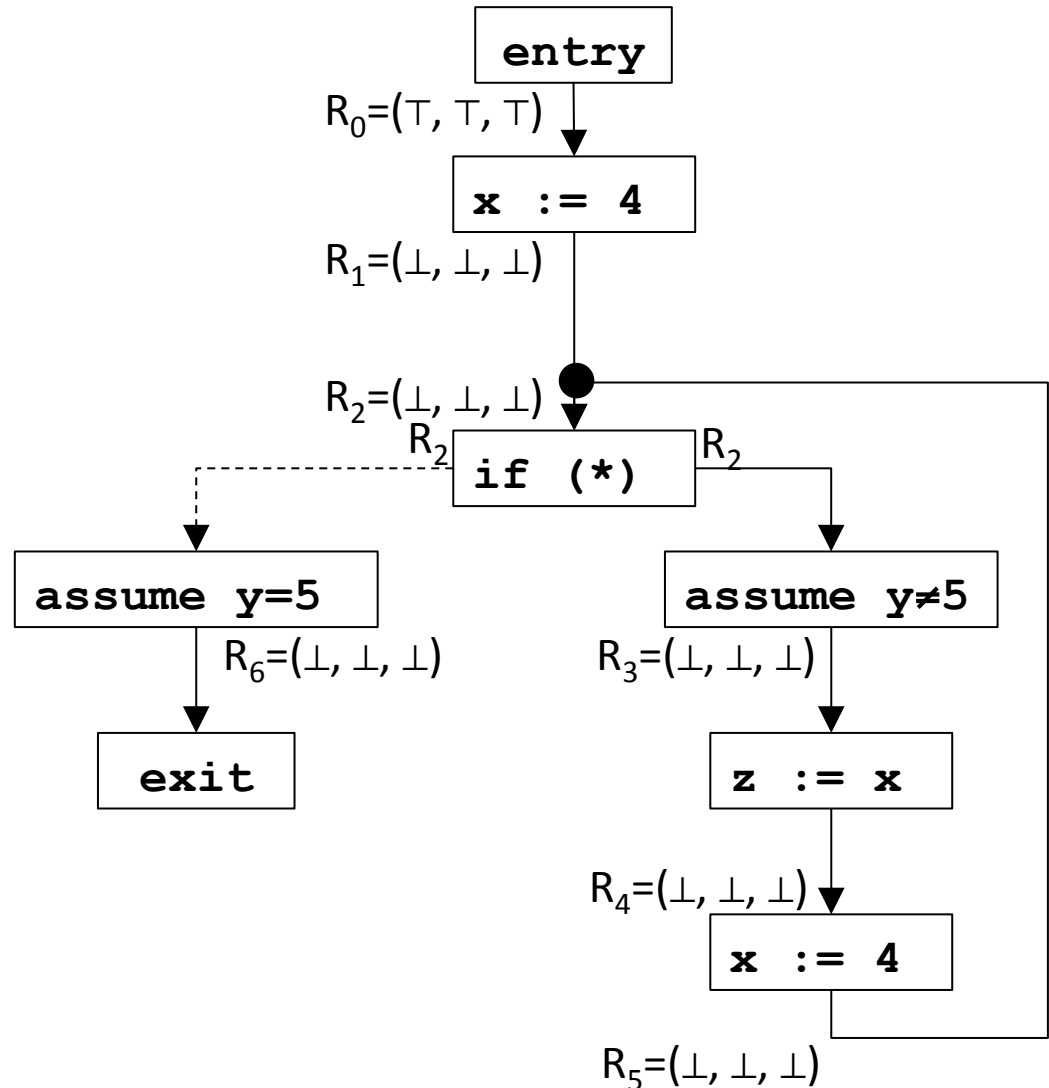
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

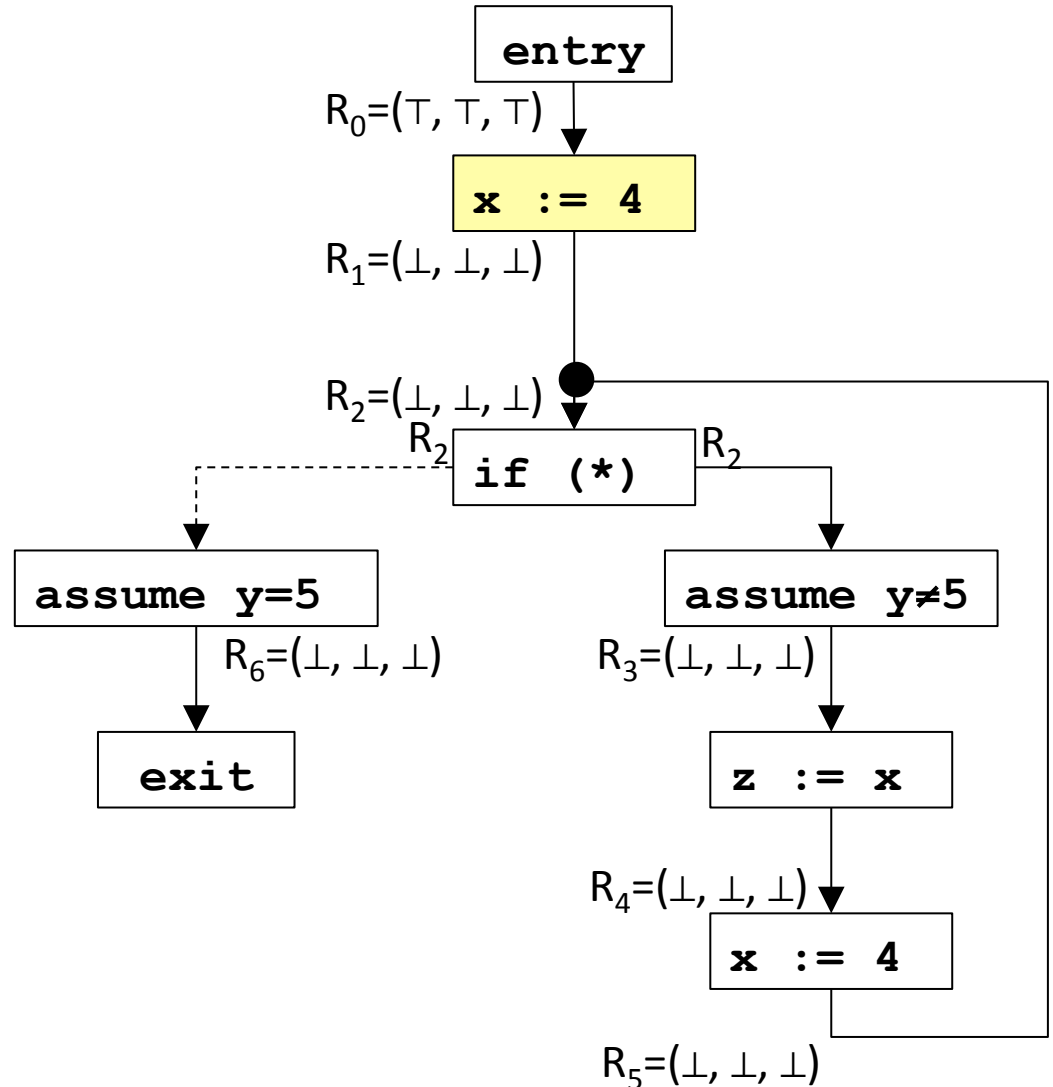
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

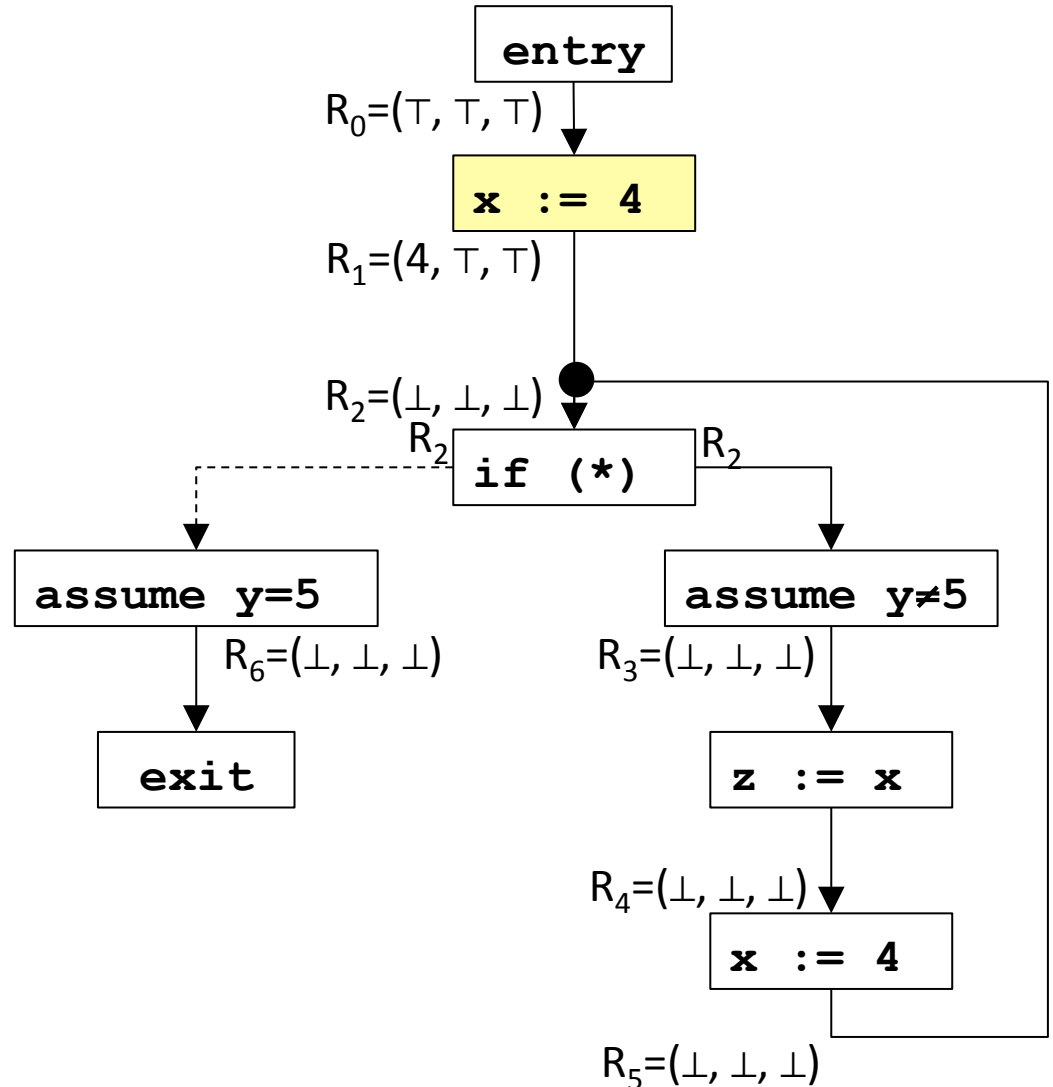
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

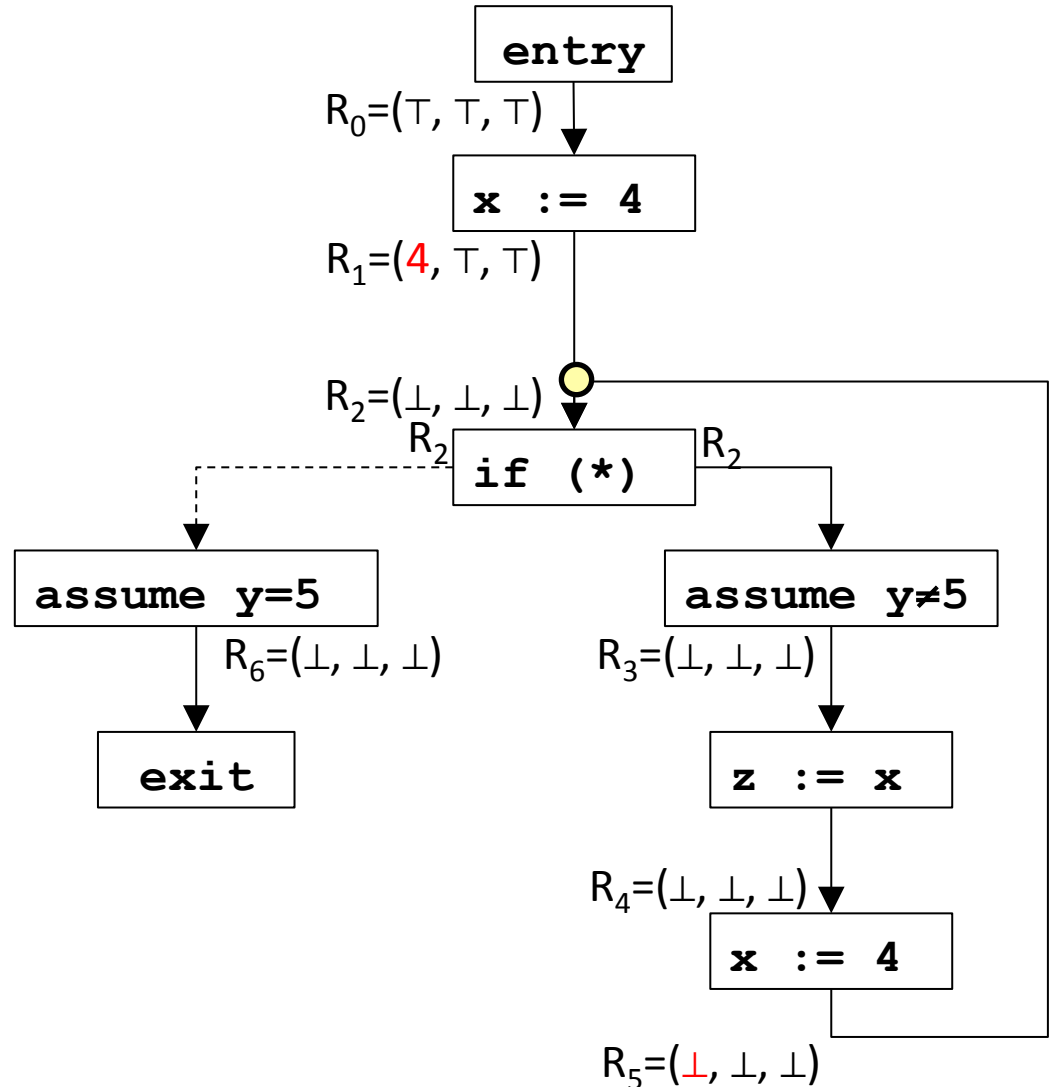
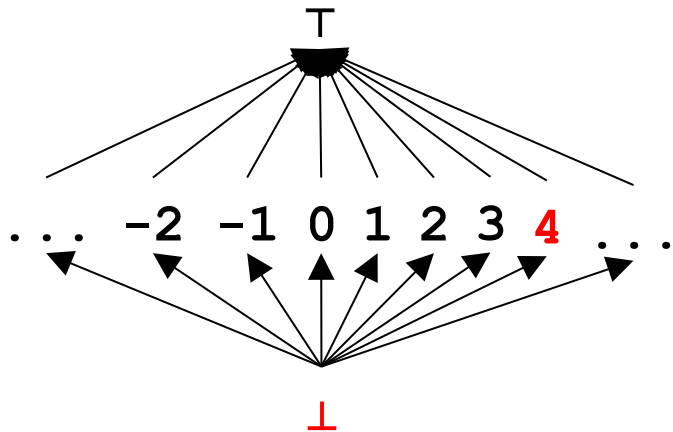
$$R_3 = \llbracket \text{assume } \mathbf{y} \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \text{assume } \mathbf{y} = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

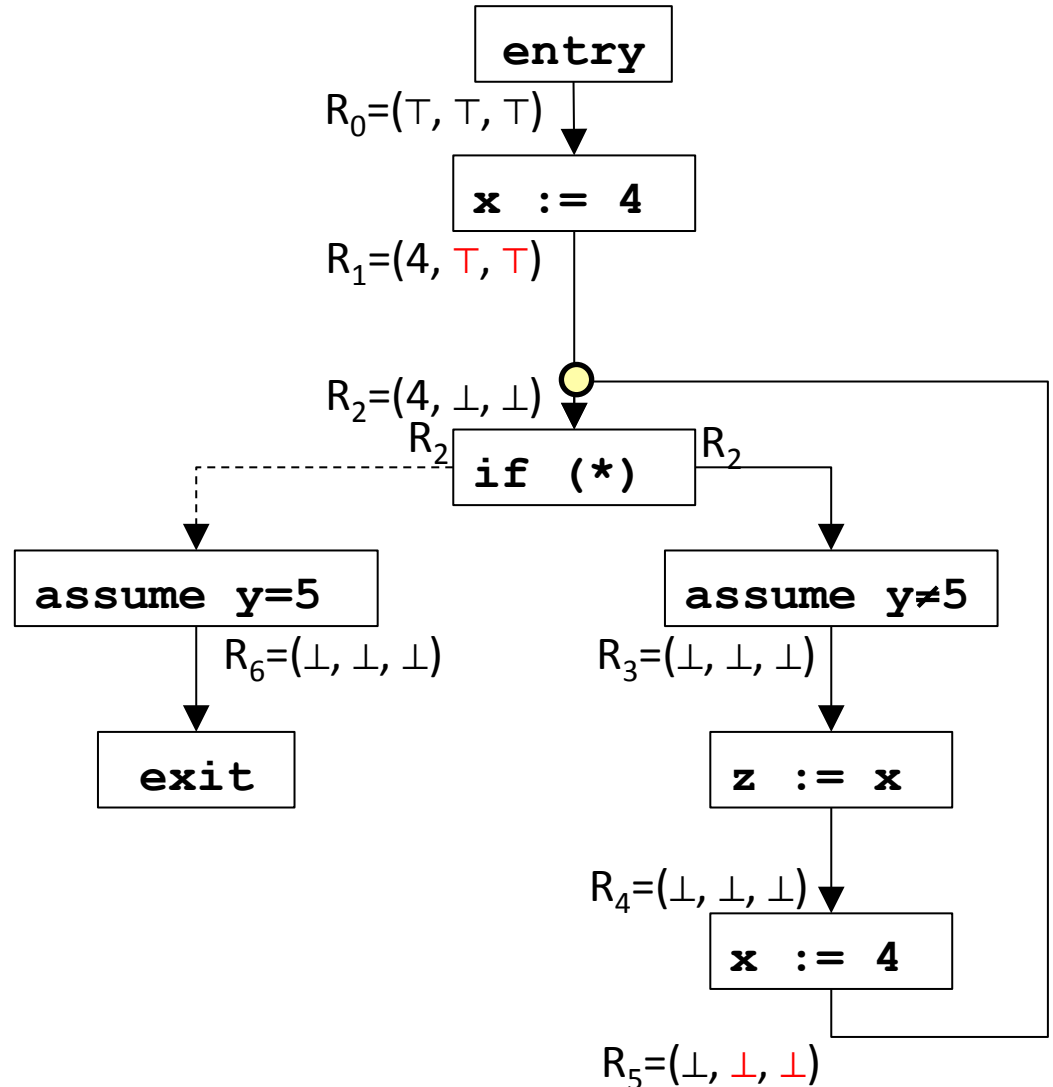
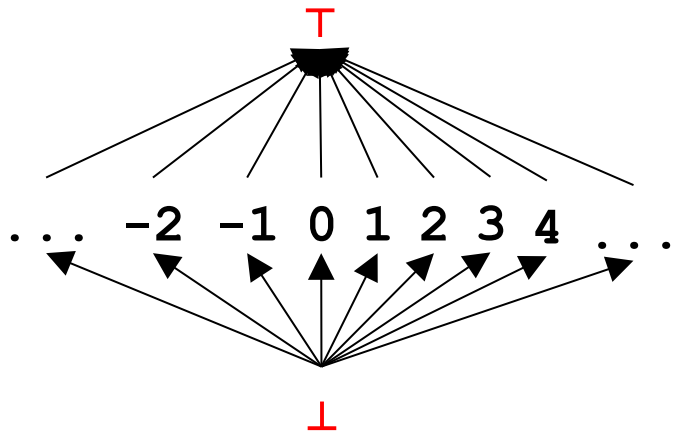
$$R_3 = \llbracket \text{assume } \mathbf{y} \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \text{assume } \mathbf{y} = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

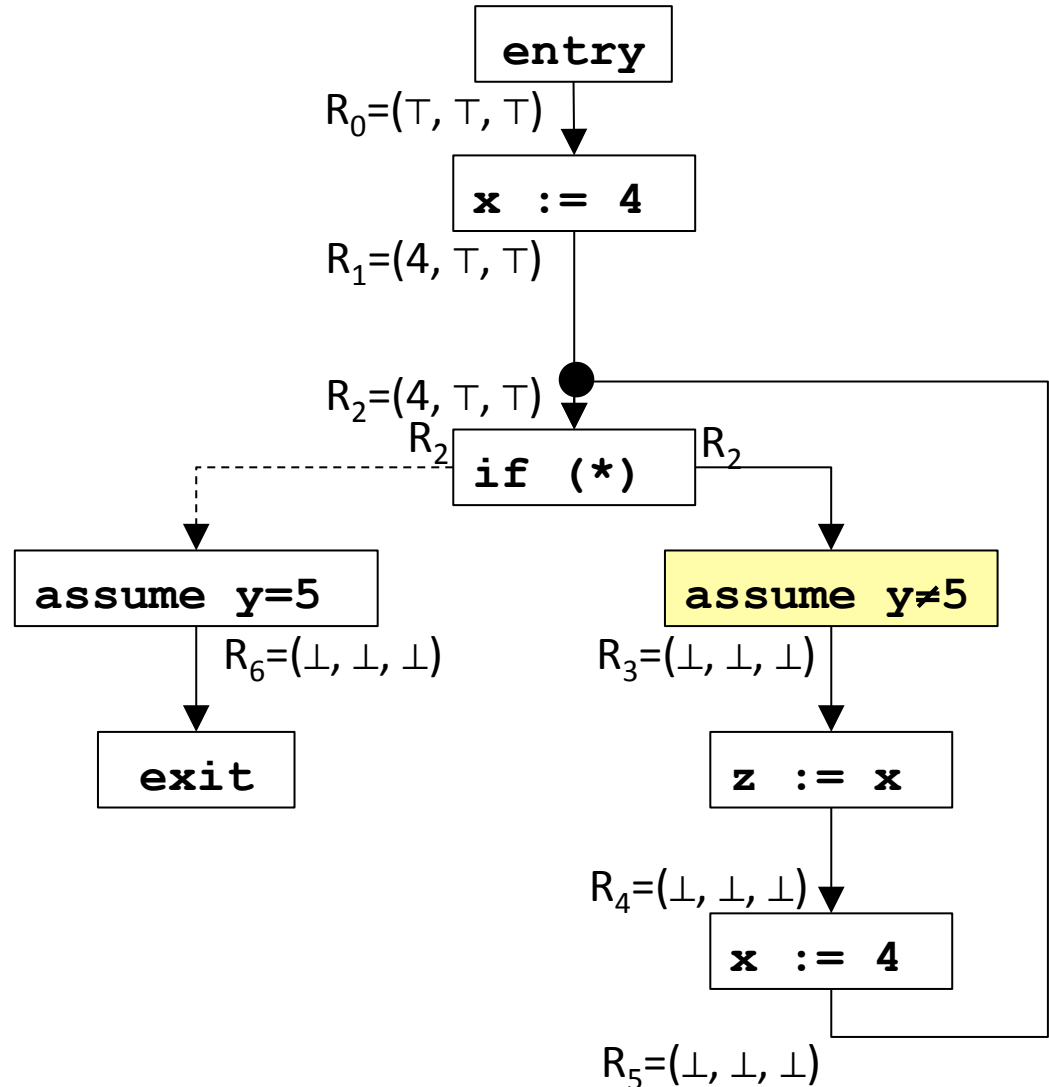
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

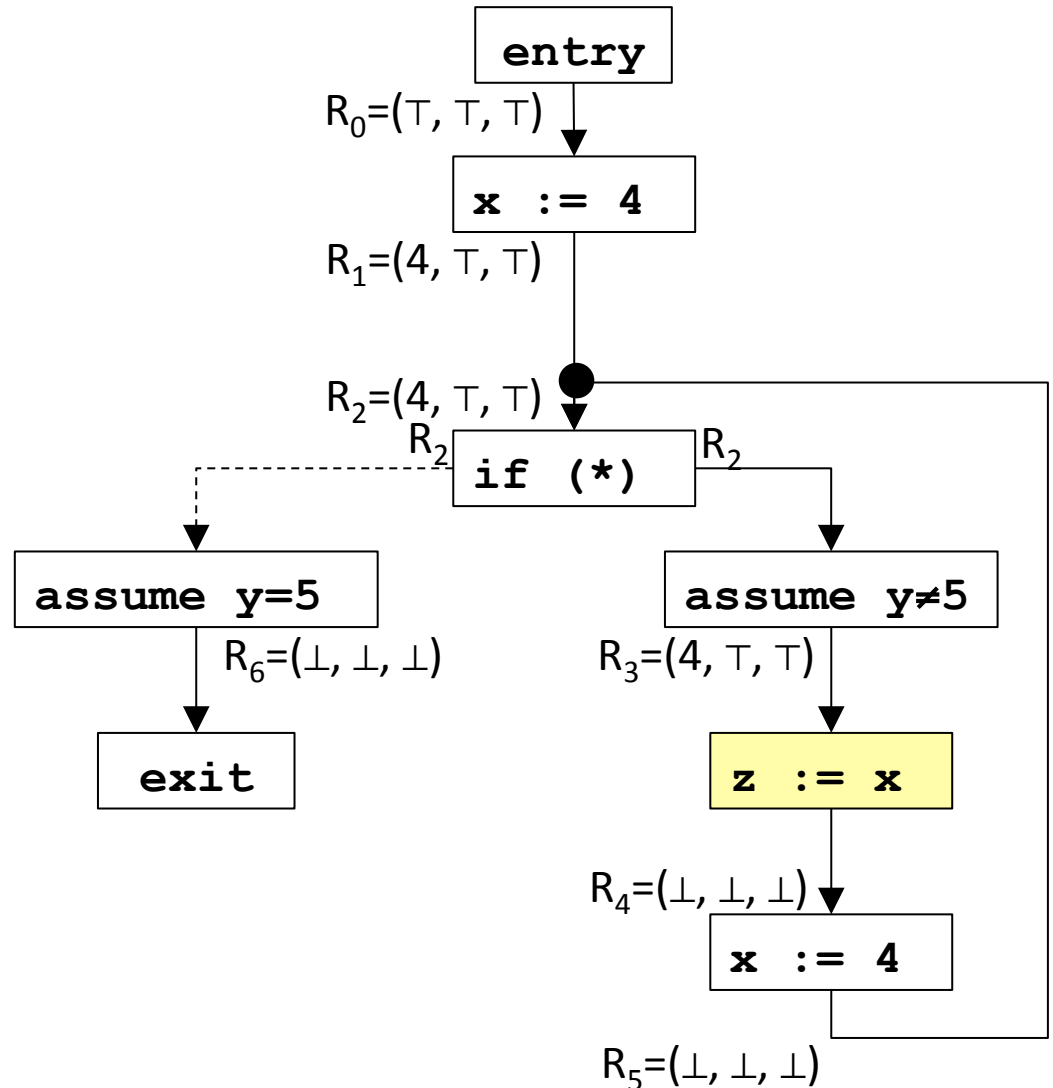
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

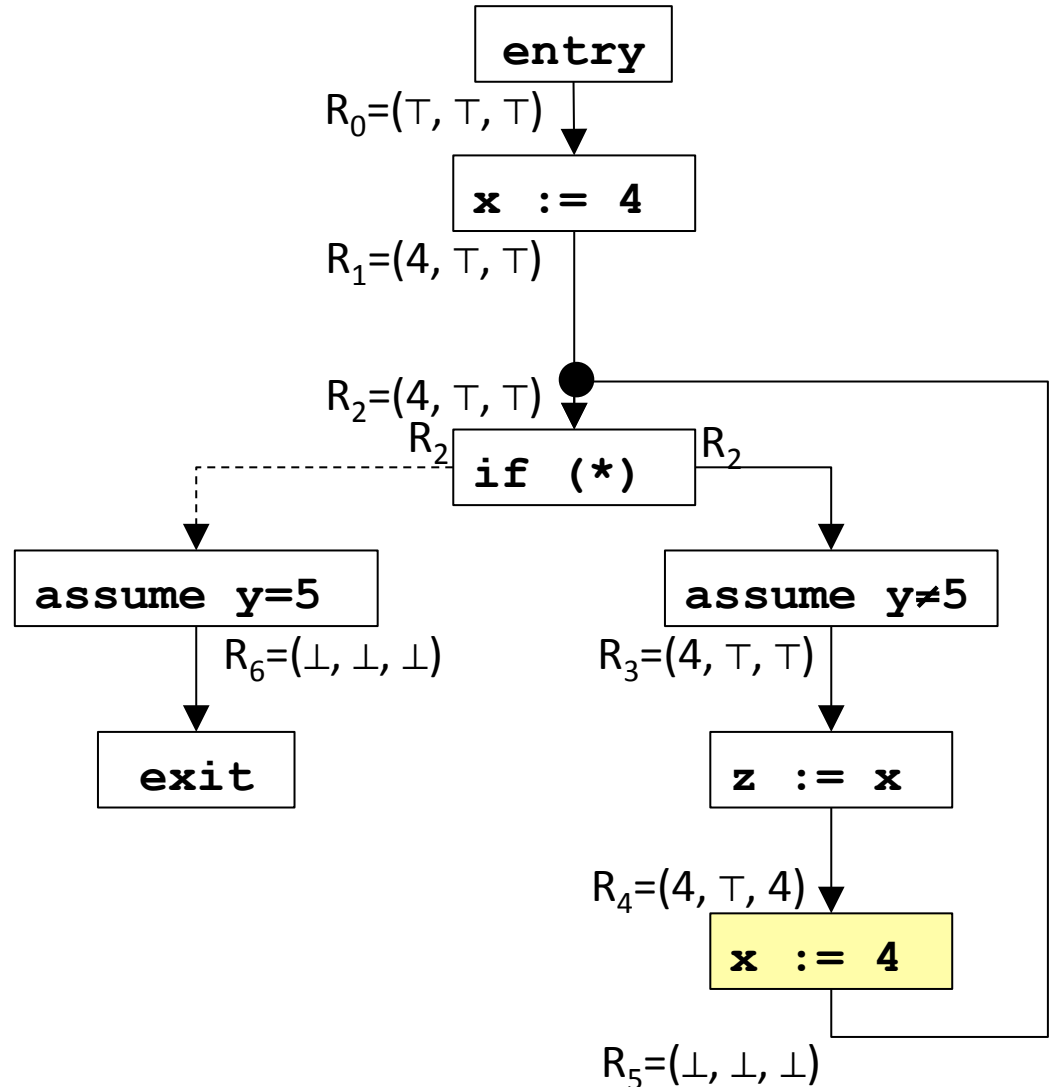
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

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



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

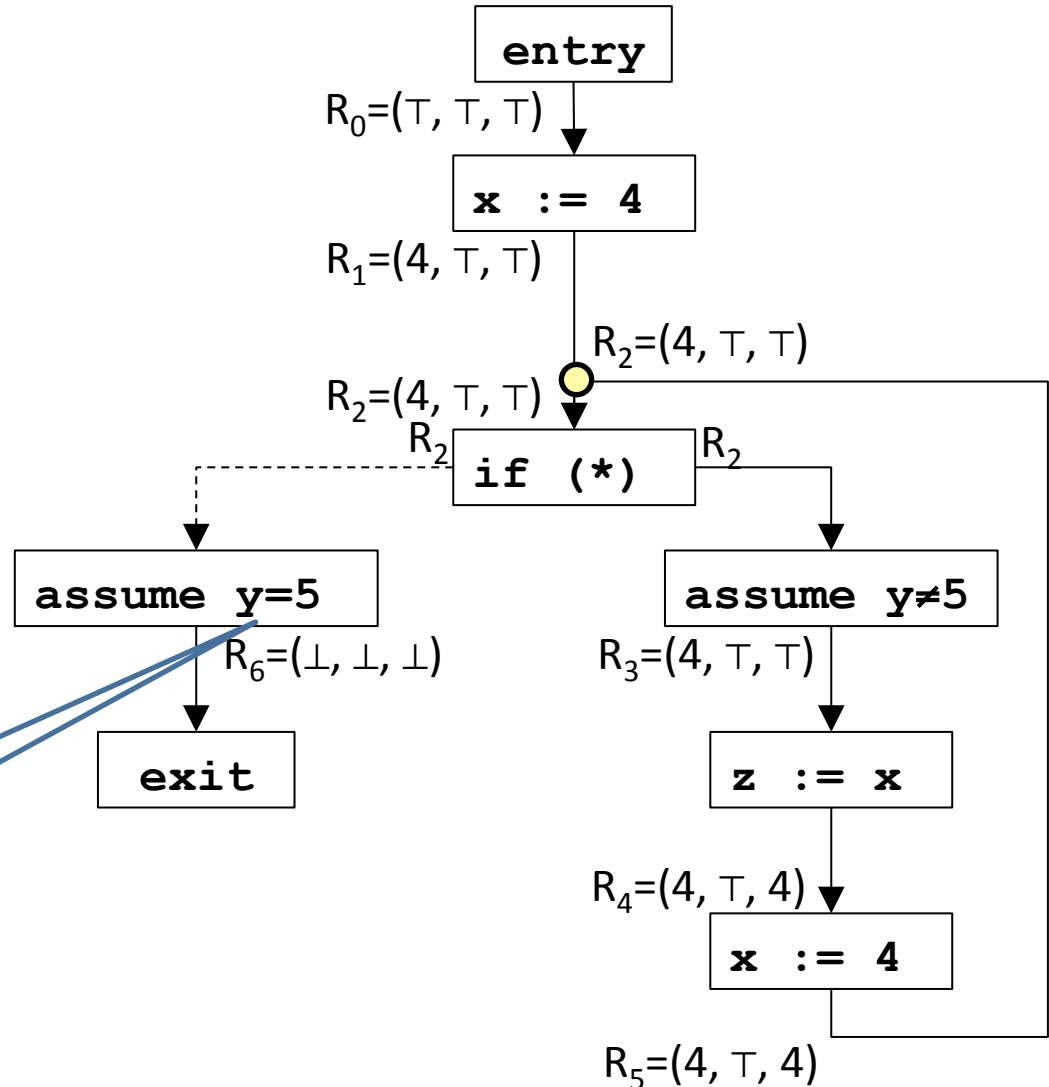
$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

$$WL = \{R_2, R_6\}$$

added R_2 back to worklist since it depends on R_5



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

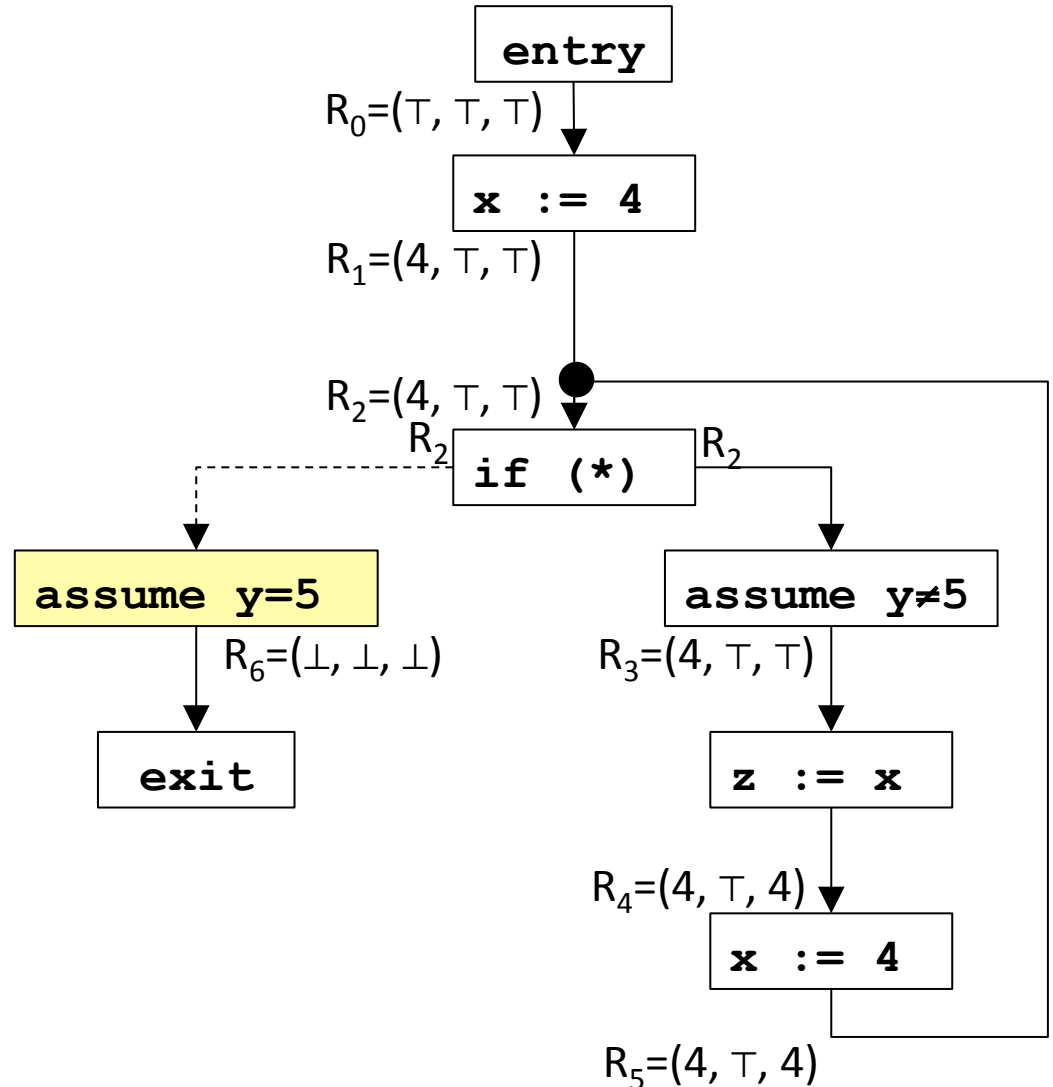
$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

$$WL = \{R_6\}$$



Chaotic iteration for CP

$$R_0 = \top$$

$$R_1 = \llbracket \mathbf{x} := 4 \rrbracket \# R_0$$

$$R_2 = R_1 \sqcup R_5$$

$$R_3 = \llbracket \mathbf{assume} \ y \neq 5 \rrbracket \# R_2$$

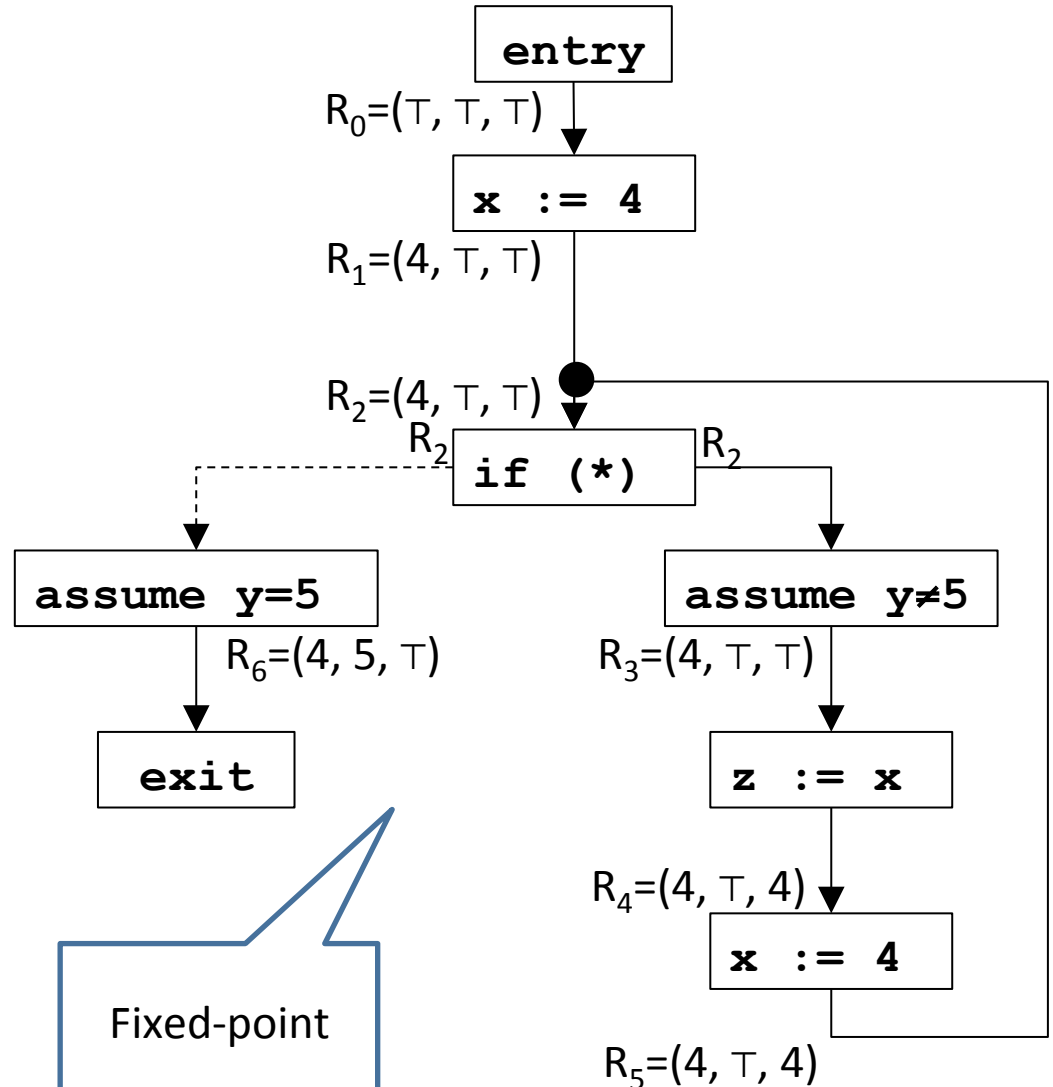
$$R_4 = \llbracket \mathbf{z} := \mathbf{x} \rrbracket \# R_3$$

$$R_5 = \llbracket \mathbf{x} := 4 \rrbracket \# R_4$$

$$R_6 = \llbracket \mathbf{assume} \ y = 5 \rrbracket \# R_2$$

WL = {}

In practice maintain
a worklist of nodes



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: $L = (D, \sqsubseteq, \sqcup, \perp)$
- Define variables $R[0, \dots, n]$ 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:
$$v_{\text{out}} = F[v](\sqcup u \mid (u, v) \text{ is a CFG edge})$$
 - 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

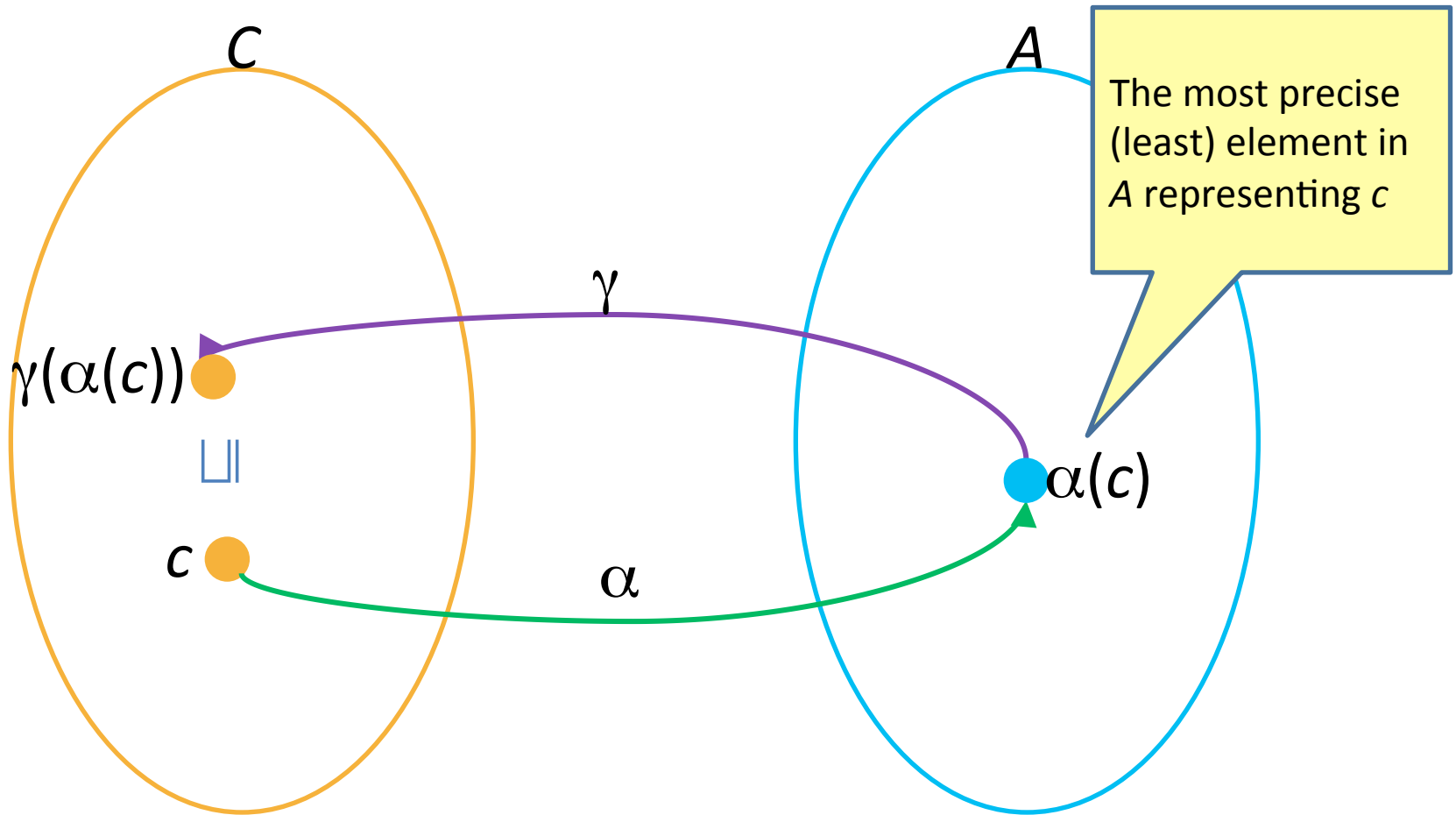
Recap

- 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?

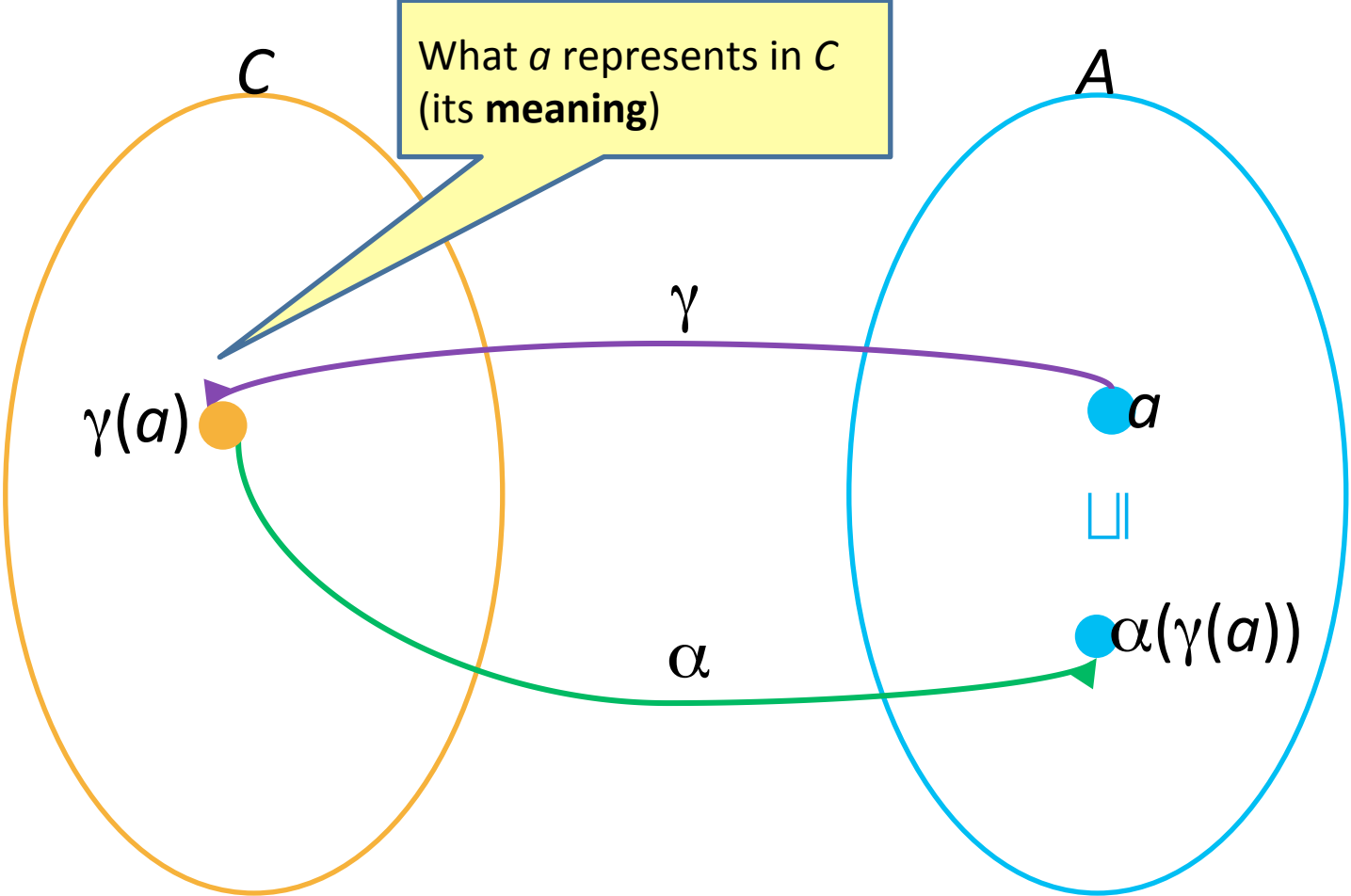
Galois Connection

- Given two complete lattices
 $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$ – concrete domain
 $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$ – 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:

$$C = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$$

- Abstract lattice:

$$EQ = \{ x=y \mid x, y \in \text{Var} \}$$

$$A = (2^{EQ}, \supseteq, \cap, \cup, EQ, \emptyset)$$

- 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:

$$C = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$$

- Abstract lattice:

$$EQ = \{ x=y \mid x, y \in \text{Var} \}$$

$$A = (2^{EQ}, \supseteq, \cap, \cup, EQ, \emptyset)$$

- Treat elements of A as both formulas and sets of constraints

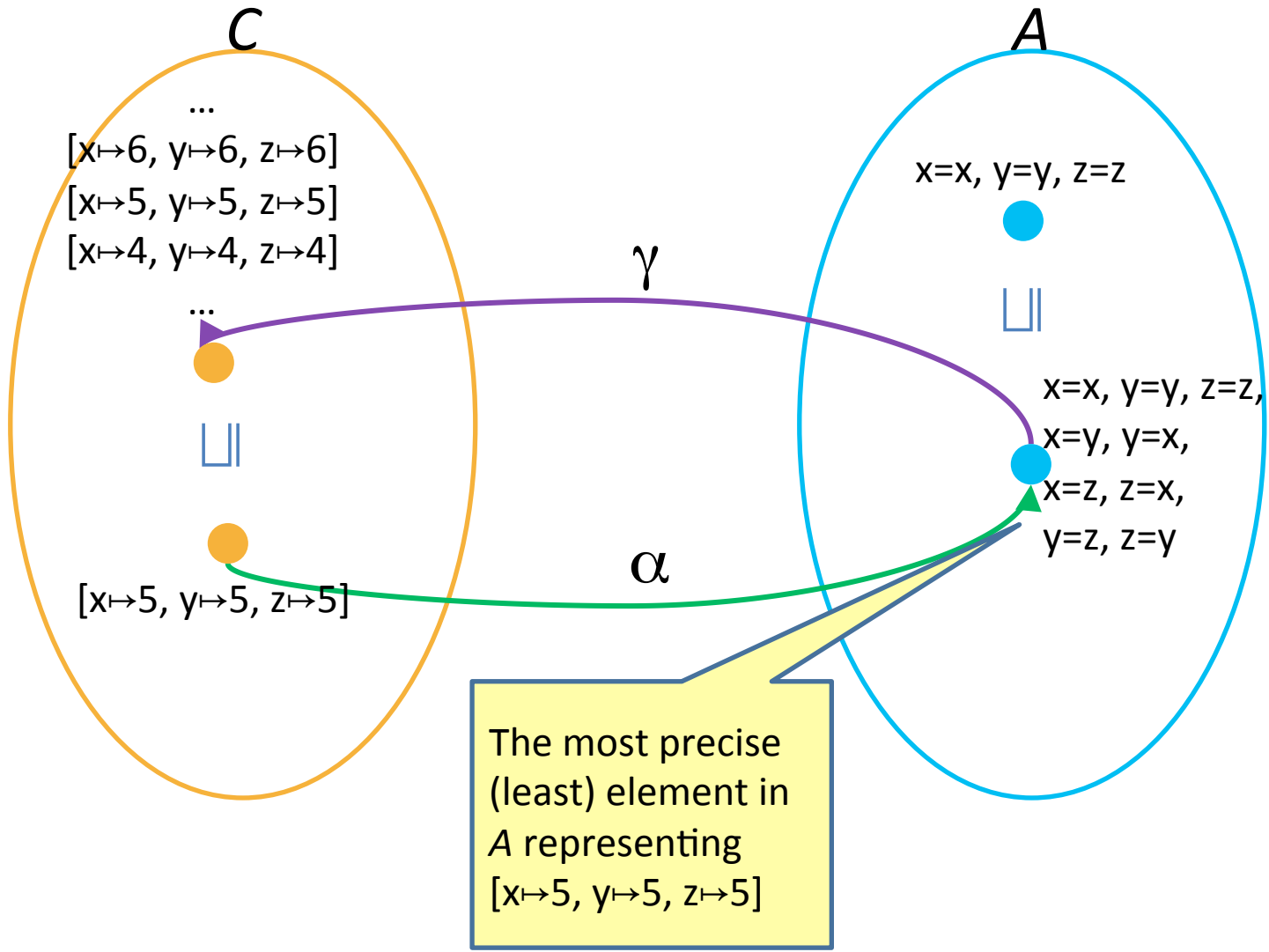
- Useful for copy propagation – a compiler optimization

- $\beta(s) = \alpha(\{s\}) = \{ x=y \mid s \models x=y \}$ that is $s \models x=y$

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

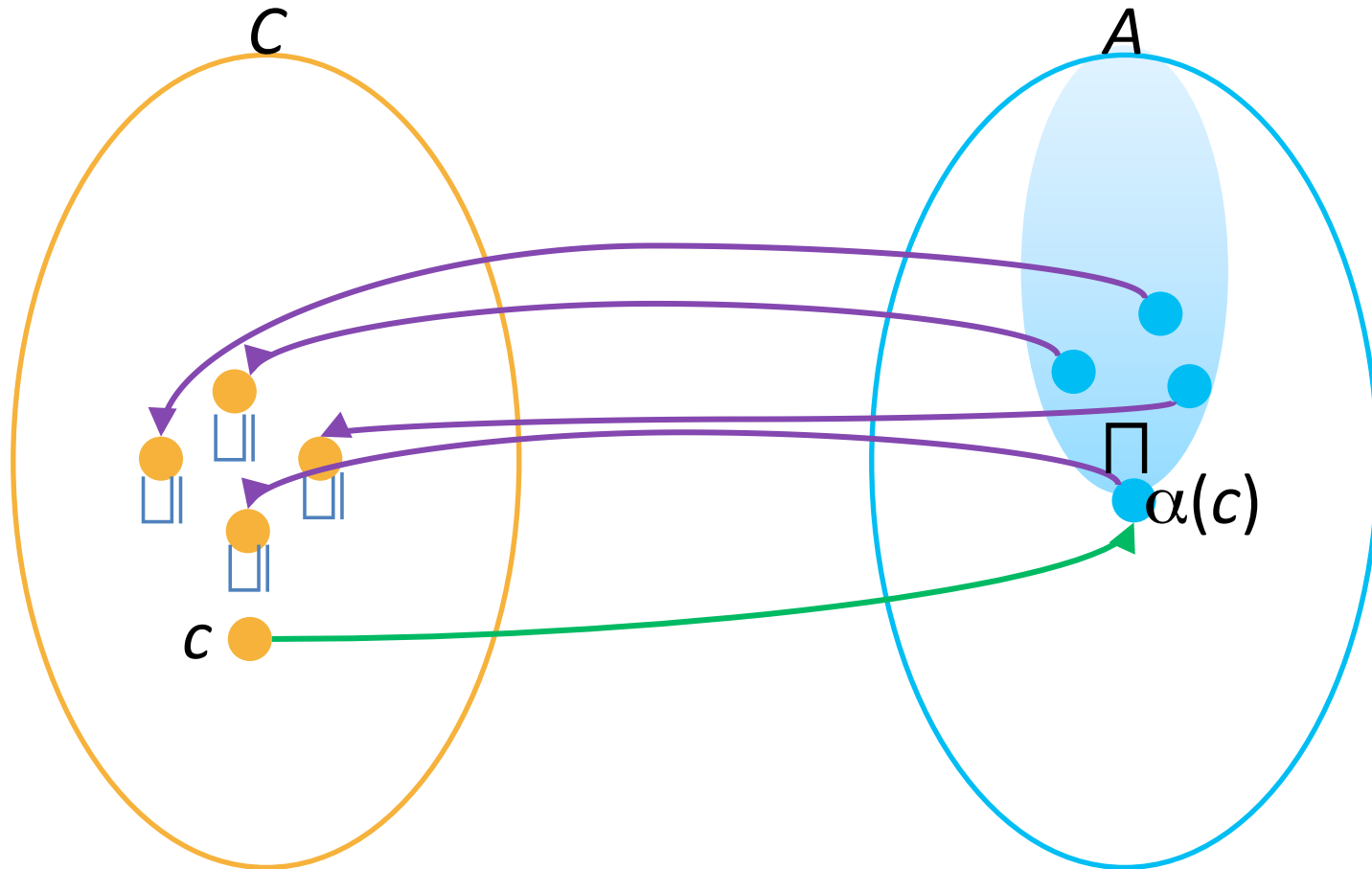
$$\gamma(Y) = \{ s \mid s \models \bigwedge Y \} = \text{models}(\bigwedge Y)$$

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



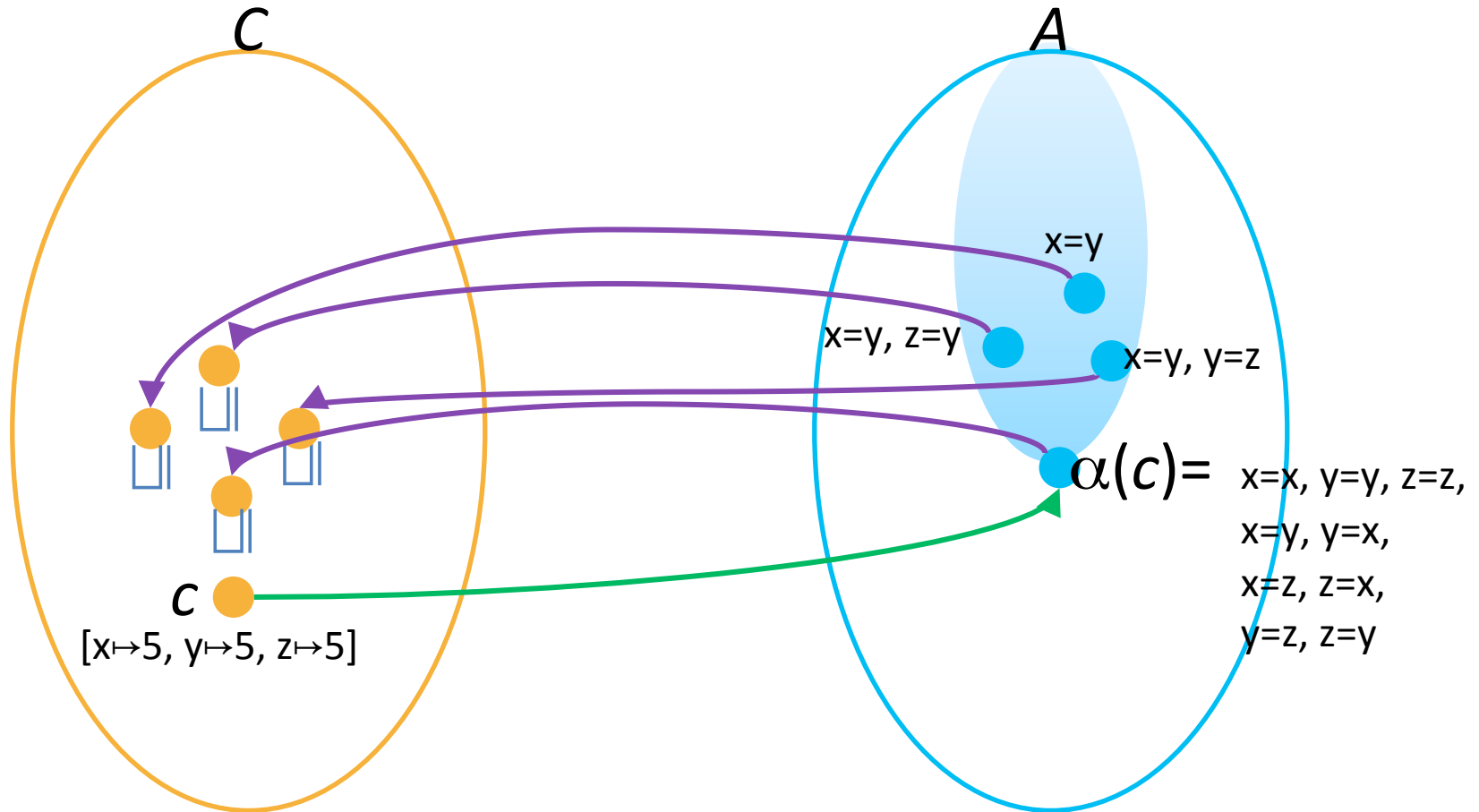
Most precise abstract representation

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

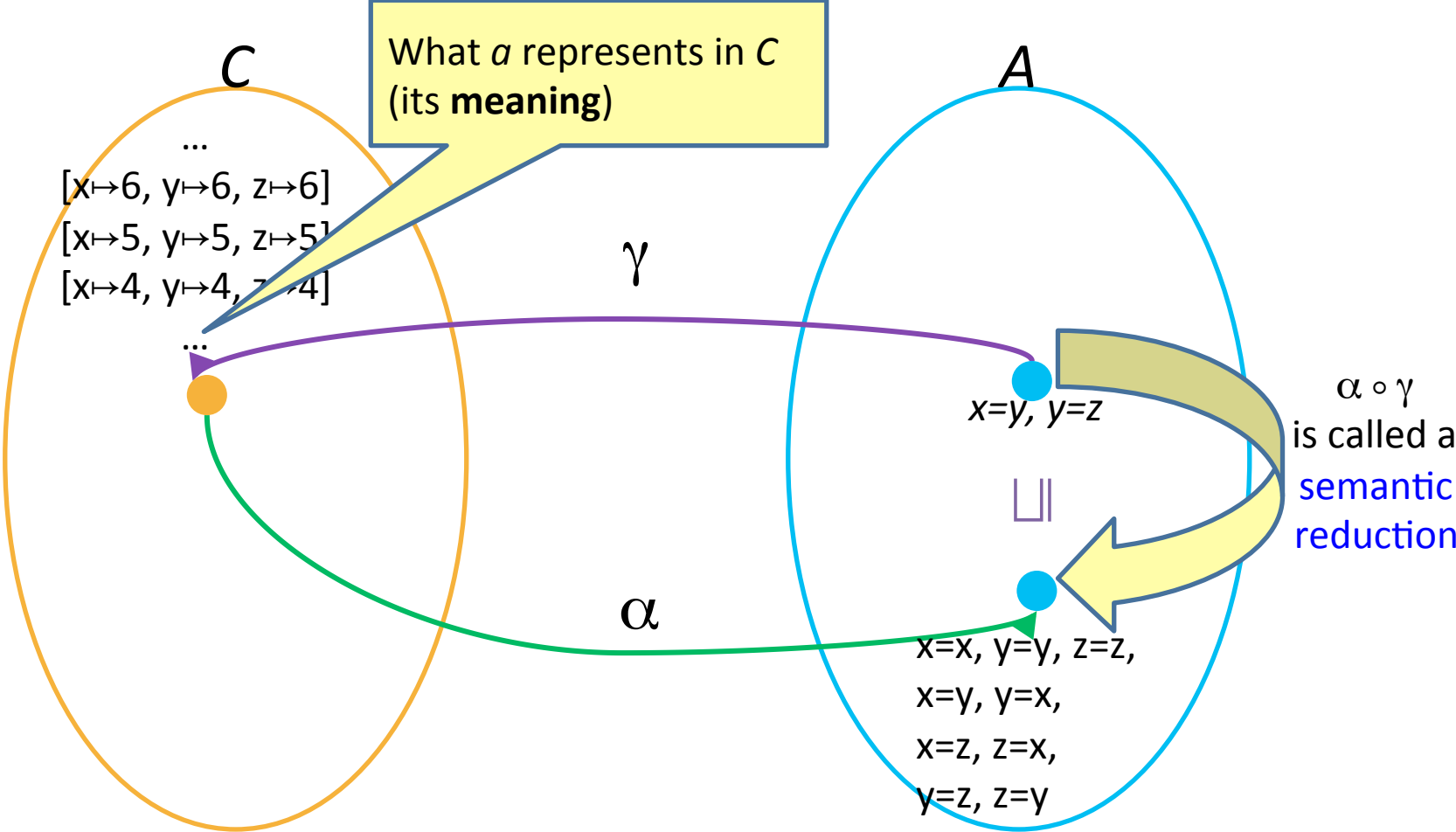


Most precise abstract representation

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

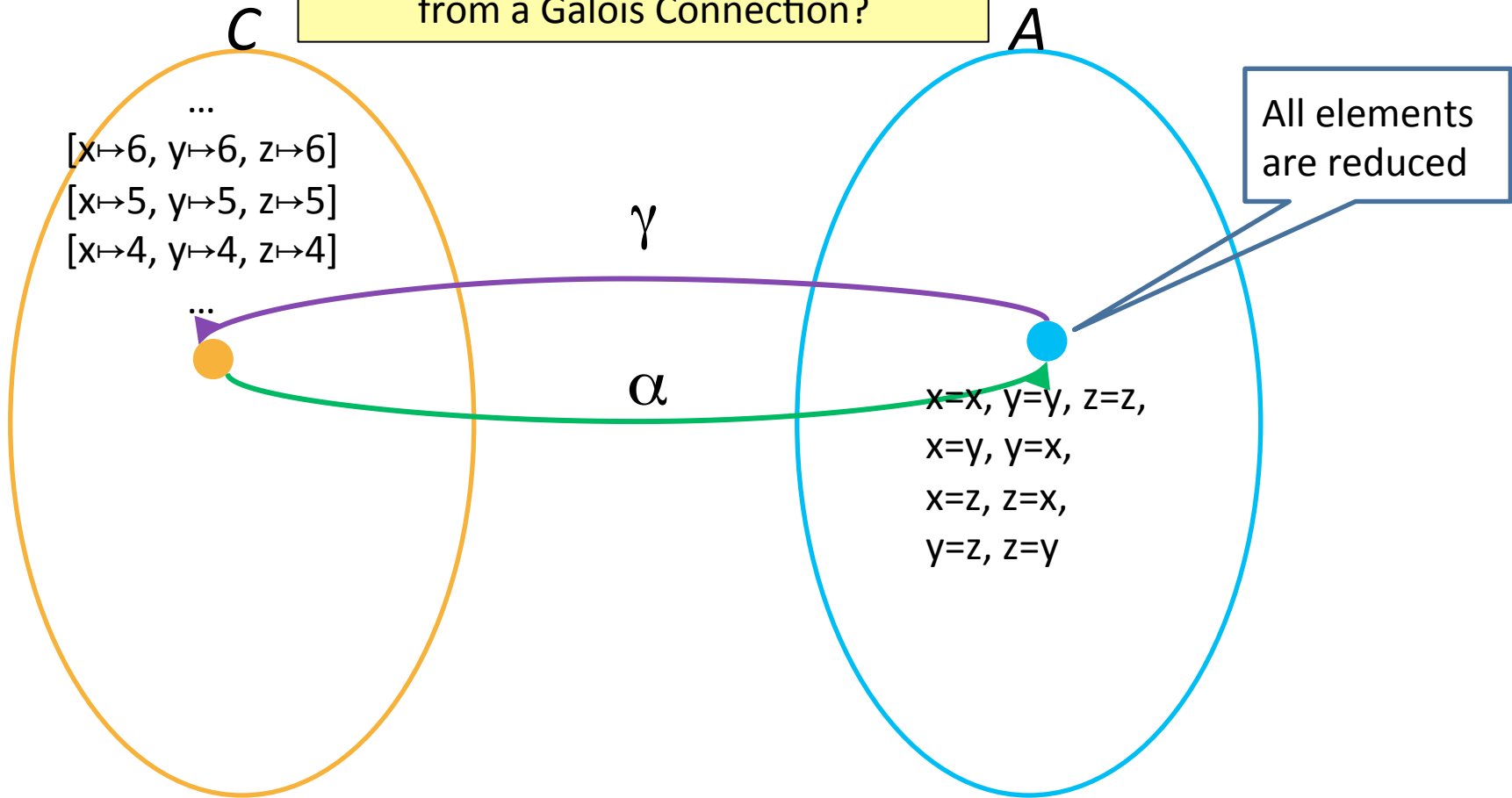


Galois Connection: $\alpha(\gamma(a)) \sqsubseteq a$



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:

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

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

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

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

The case of symbolic domains

- An important class of abstract domains are **symbolic domains** – domains of formulas
- $C = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$
 $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$
- 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

$$C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$$

$$A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$$

$$M = (D^M, \sqsubseteq^M, \sqcup^M, \sqcap^M, \perp^M, \top^M)$$

and

Galois connections

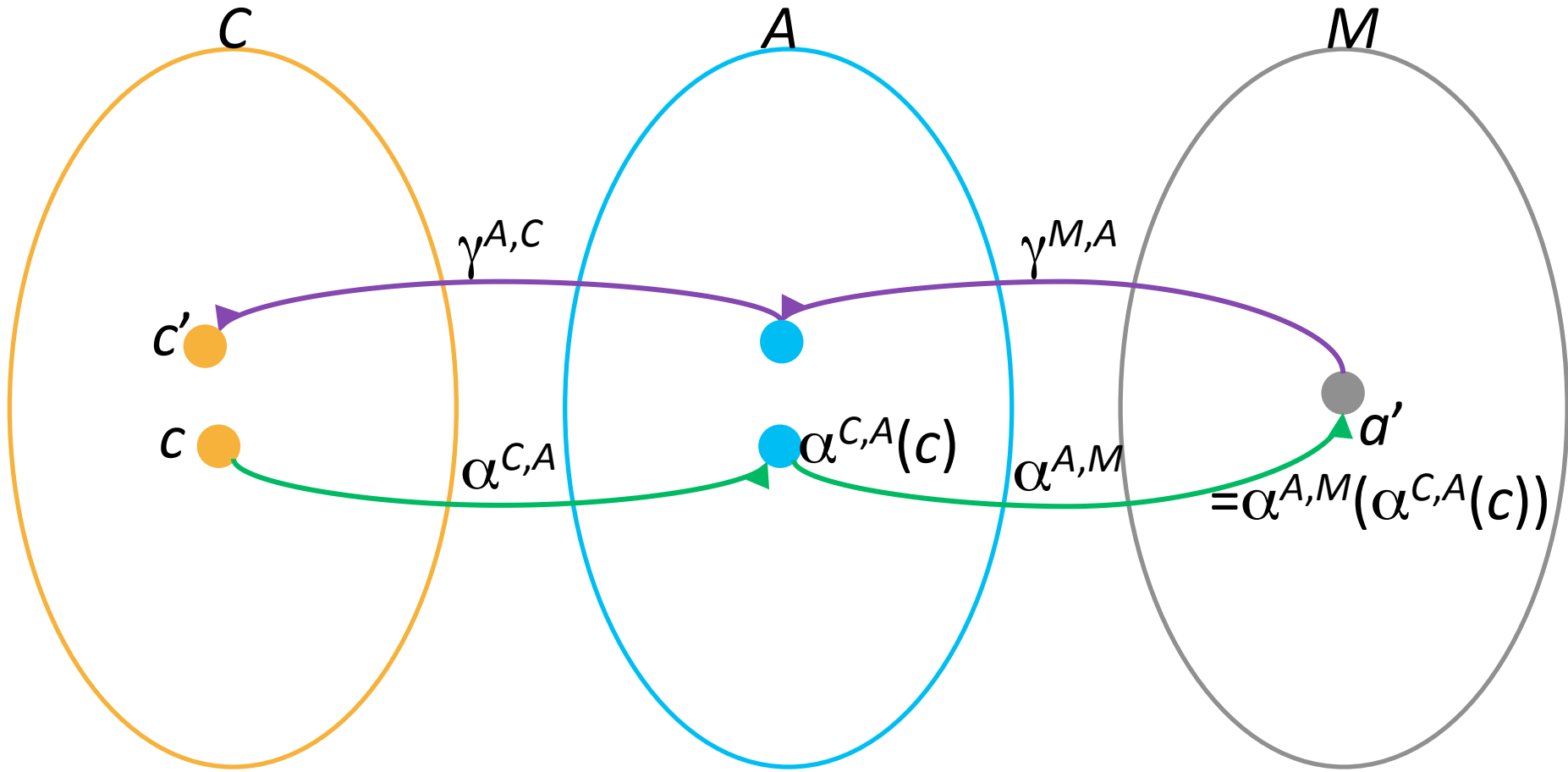
$$GC^{C,A} = (C, \alpha^{C,A}, \gamma^{A,C}, A) \text{ and } GC^{A,M} = (A, \alpha^{A,M}, \gamma^{M,A}, M)$$

- **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

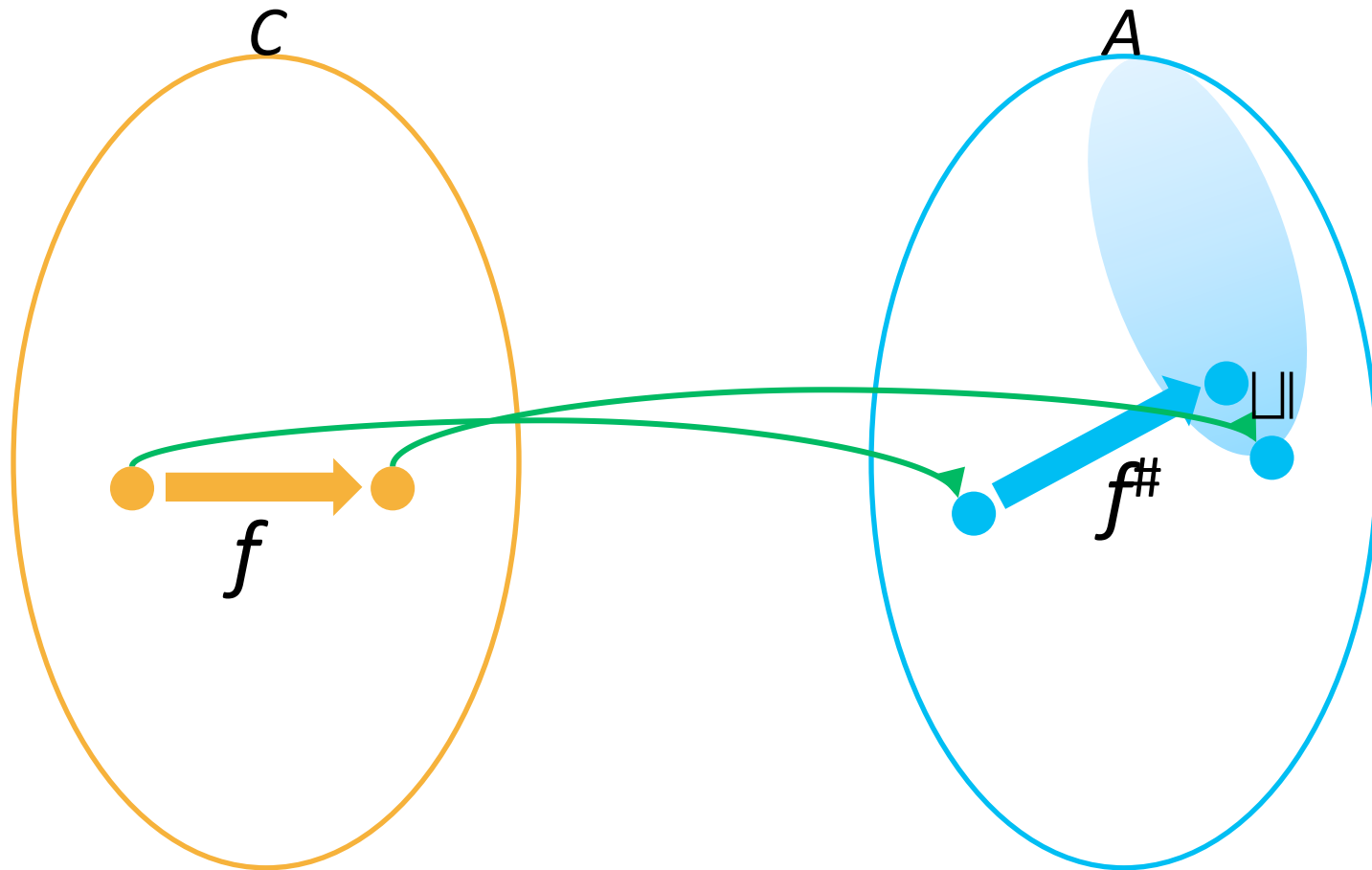


Sound abstract transformer

- Given two lattices
 $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$
 $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$
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
 $\alpha(f(\gamma(a))) \sqsubseteq^A f^\#(a)$

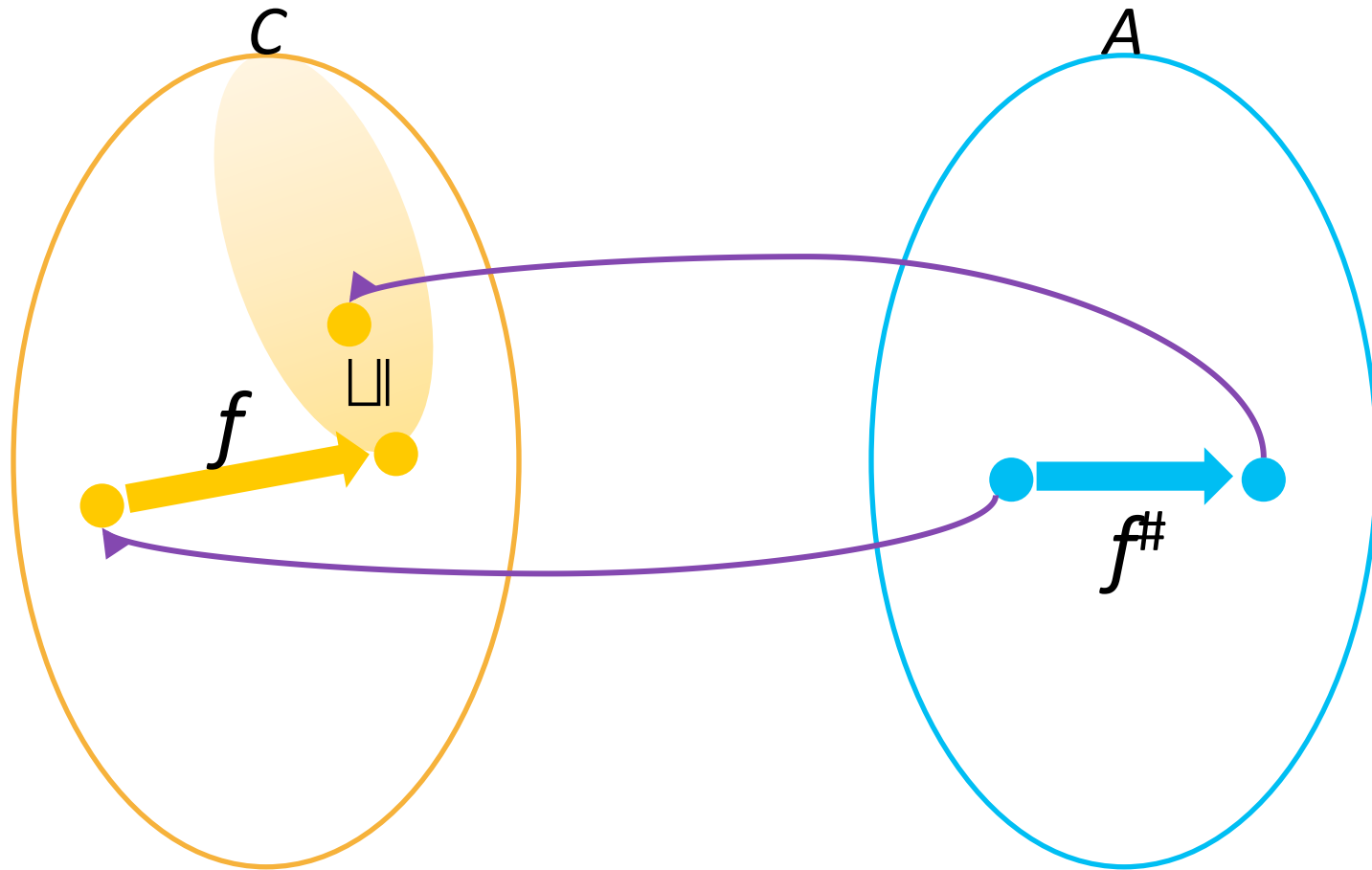
Transformer soundness condition 1

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



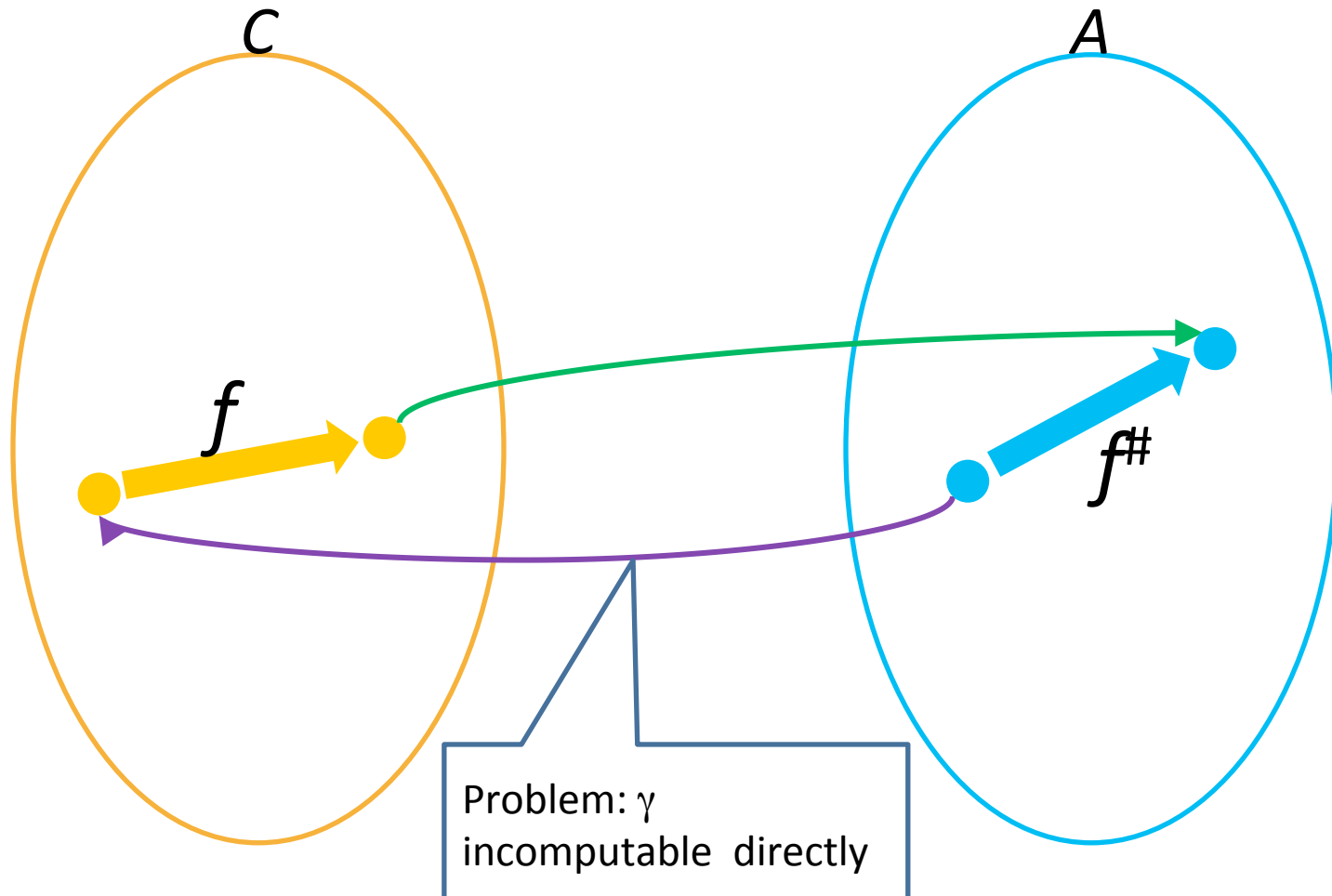
Transformer soundness condition 2

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



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

- $C = (2^{\text{State}}, \subseteq, \cup, \cap, \emptyset, \mathbf{State})$
- $EQ = \{x=y \mid x, y \in \text{Var}\}$
 $A = (2^{EQ}, \supseteq, \cap, \cup, EQ, \emptyset)$
- $\beta(s) = \alpha(\{s\}) = \{x=y \mid s \models x=y\}$ that is $s \models x=y$
 $\alpha(X) = \cap\{\beta(s) \mid s \in X\} = \sqcap^A \{\beta(s) \mid s \in X\}$
 $\gamma(\varphi) = \{s \mid s \models \varphi\} = \text{models}(\varphi)$
- Concrete: $\llbracket x:=y \rrbracket X = \{s[x \mapsto s.y] \mid s \in X\}$
- Abstract: $\llbracket x:=y \rrbracket^\# X = ?$

Developing a transformer for EQ - 1

- Input has the form $X = \bigwedge \{a=b\}$
- $sp(x:=expr, \varphi) = \exists v. x=expr[v/x] \wedge \varphi[v/x]$
- $sp(x:=y, X) = \exists v. x=y[v/x] \wedge \bigwedge \{a=b\}[v/x] = \dots$
- 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

- $sp(x:=y, X) = \exists v. x=y[v/x] \wedge \bigwedge \{a=b\}[v/x] = \dots$
- Two cases
 - x is y : $sp(x:=y, X) = X$
 - x is different from y :
$$sp(x:=y, X) = \exists v. x=y \wedge EQ_{\supset} X, x)[v/x] \wedge EQ_c(X, x)[v/x]$$
$$= x=y \wedge EQ_c(X, x) \wedge \exists v. EQ_{\supset} X, x)[v/x]$$
$$\Rightarrow x=y \wedge EQ_c(X, x)$$
- Vanilla transformer: $\llbracket x:=y \rrbracket^{\#1} X = x=y \wedge EQ_c(X, x)$
- Example: $\llbracket x:=y \rrbracket^{\#1} \bigwedge \{x=p, q=x, m=n\} = \bigwedge \{x=y, m=n\}$
Is this the most precise result?

Developing a transformer for EQ - 3

- $\llbracket x:=y \rrbracket^{\#1} \wedge \{x=p, x=q, m=n\} = \wedge \{x=y, m=n\} \equiv \wedge \{x=y, m=n, p=q\}$
 - Where does the information $p=q$ come from?
- $sp(x:=y, X) =$
 $x=y \wedge EQc(X, x) \wedge \exists v. EQ \rangle X, x)[v/x]$
- $\exists v. EQ \rangle X, x)[v/x]$ 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 $\llbracket x:=y \rrbracket^{\#2} = \llbracket x:=y \rrbracket^{\#1} \circ \text{Explicate}$

- $\llbracket x:=y \rrbracket^{\#2} \wedge \{x=p, x=q, m=n\} = \wedge \{x=y, m=n, p=q\}$
is this the best transformer?

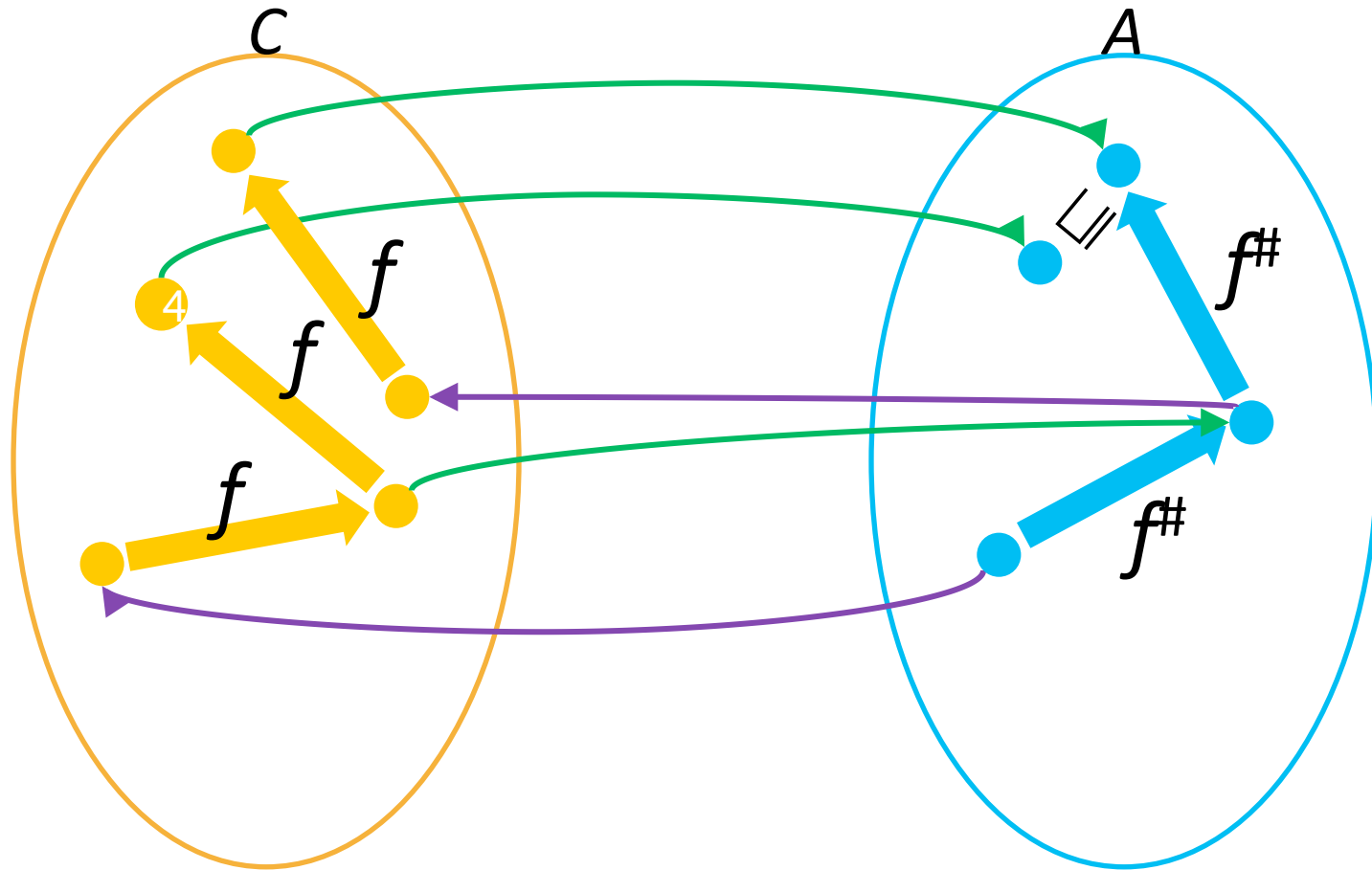
Developing a transformer for EQ - 5

- $\llbracket x:=y \rrbracket^{\#2} \wedge \{y=z\} = \{x=y, y=z\} \sqsupseteq \{x=y, y=z, x=z\}$
- Idea: apply reduction operator again after the vanilla transformer
- $\llbracket x:=y \rrbracket^{\#3} = \text{Explicate} \circ \llbracket x:=y \rrbracket^{\#1} \circ \text{Explicate}$
- 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: $\llbracket x:=y \rrbracket^{\#}(X) = \text{Explicate} \circ \llbracket x:=y \rrbracket^{\#1}$
 - 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
 $\alpha(f(f(\gamma(a)))) \not\sqsubseteq f^\#(f^\#(a))$

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



Soundness theorem 1

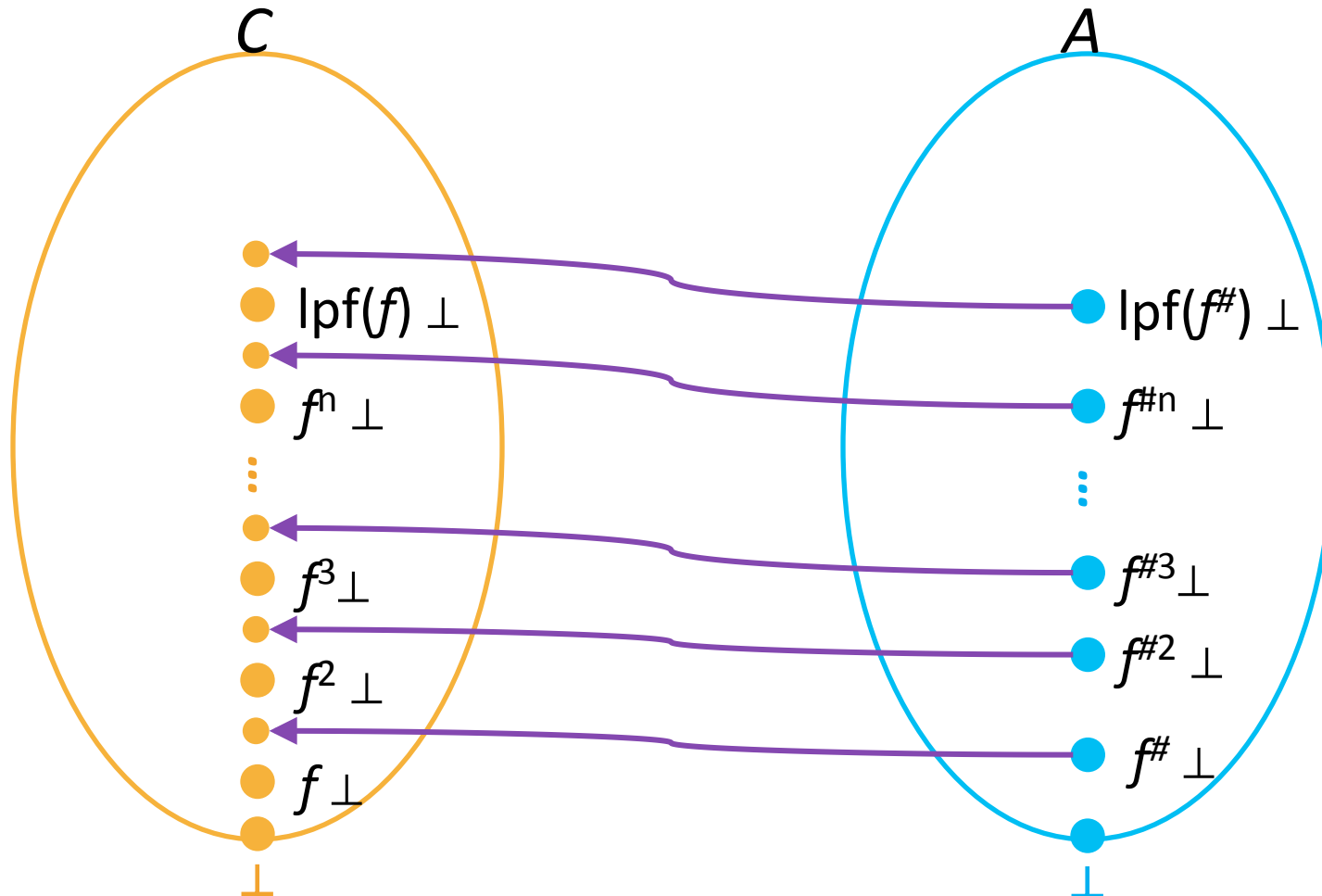
1. Given two complete lattices
 $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$
 $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$
and $GC^{C,A} = (C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \rightarrow D^C$
3. Monotone abstract transformer $f^\#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

Then

$$\begin{aligned} \text{lfp}(f) &\sqsubseteq \gamma(\text{lfp}(f^\#)) \\ \alpha(\text{lfp}(f)) &\sqsubseteq \text{lfp}(f^\#) \end{aligned}$$

Soundness theorem 1

$$\begin{aligned}
 \forall a \in D^A : f(\gamma(a)) \sqsubseteq \gamma(f^\#(a)) &\Rightarrow \forall a \in D^A : f^n(\gamma(a)) \sqsubseteq \gamma(f^{\#n}(a)) \\
 &\Rightarrow \forall a \in D^A : \text{lfp}(f^n)(\gamma(a)) \sqsubseteq \gamma(\text{lfp}(f^{\#n})(a)) \\
 &\Rightarrow \text{lfp}(f) \perp \sqsubseteq \text{lfp}(f^\#) \perp
 \end{aligned}$$



Soundness theorem 2

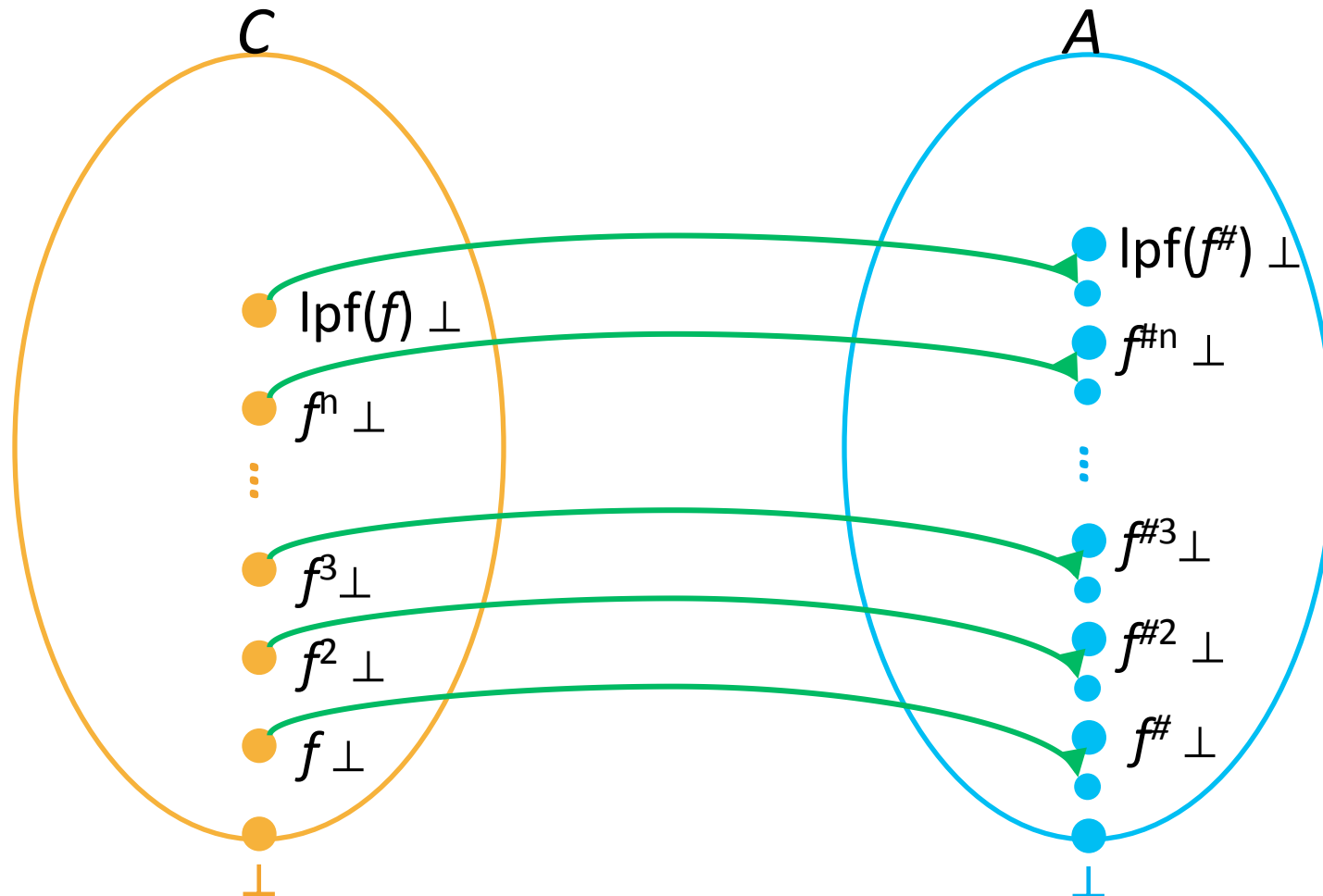
1. Given two complete lattices
 $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$
 $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$
and $GC^{C,A} = (C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \rightarrow D^C$
3. Monotone abstract transformer $f^\#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

Then

$$\begin{aligned} \alpha(\text{lfp}(f)) &\sqsubseteq \text{lfp}(f^\#) \\ \text{lfp}(f) &\sqsubseteq \gamma(\text{lfp}(f^\#)) \end{aligned}$$

Soundness theorem 2

$$\begin{aligned}
 \forall c \in D^C : \alpha(f(c)) \sqsubseteq f^\#(\alpha(c)) &\Rightarrow \forall c \in D^C : \alpha(f^n(c)) \sqsubseteq f^{\#n}(\alpha(c)) \\
 &\Rightarrow \forall c \in D^C : \alpha(\text{lfp}(f)(c)) \sqsubseteq \text{lfp}(f^\#)(\alpha(c)) \\
 &\Rightarrow \text{lfp}(f) \perp \sqsubseteq \text{lfp}(f^\#) \perp
 \end{aligned}$$



A recipe for a sound static analysis

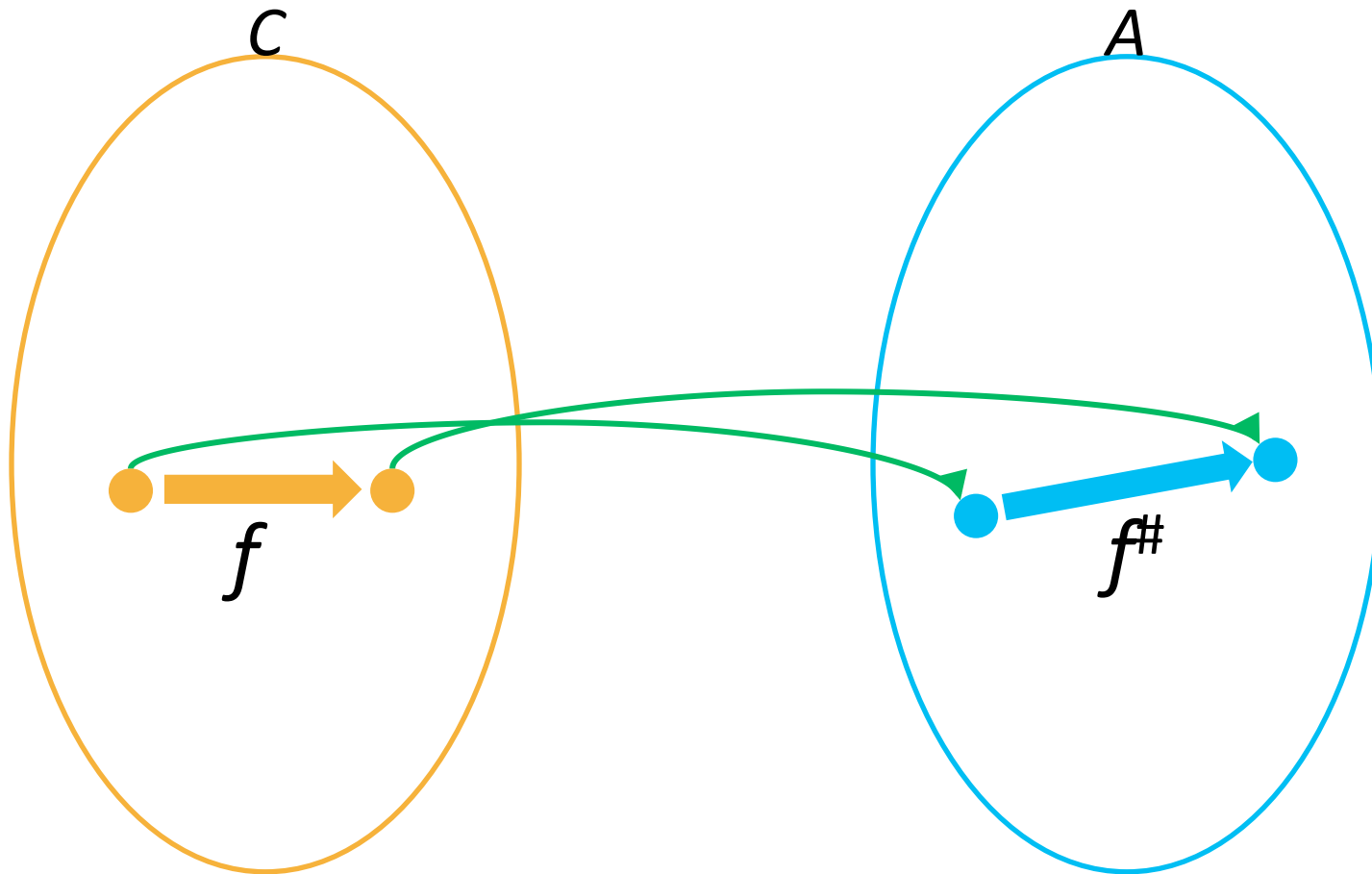
- Define an “appropriate” operational semantics
- Define “collecting” structural operational semantics
- Establish a Galois connection between collecting states and abstract states
- **Local correctness:** show that the abstract interpretation of every atomic statement is **sound** w.r.t. the collecting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

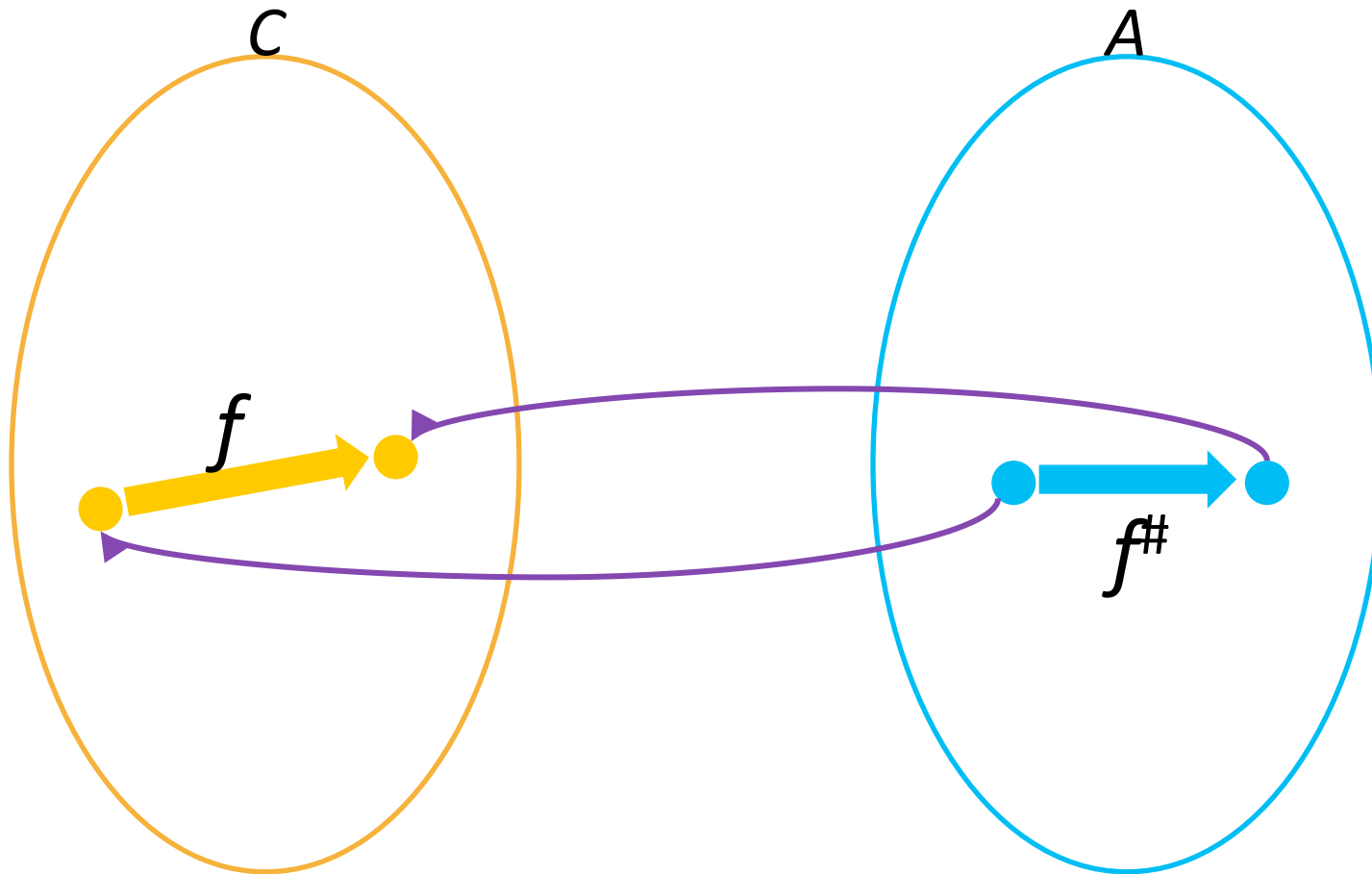
Forward complete transformer

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



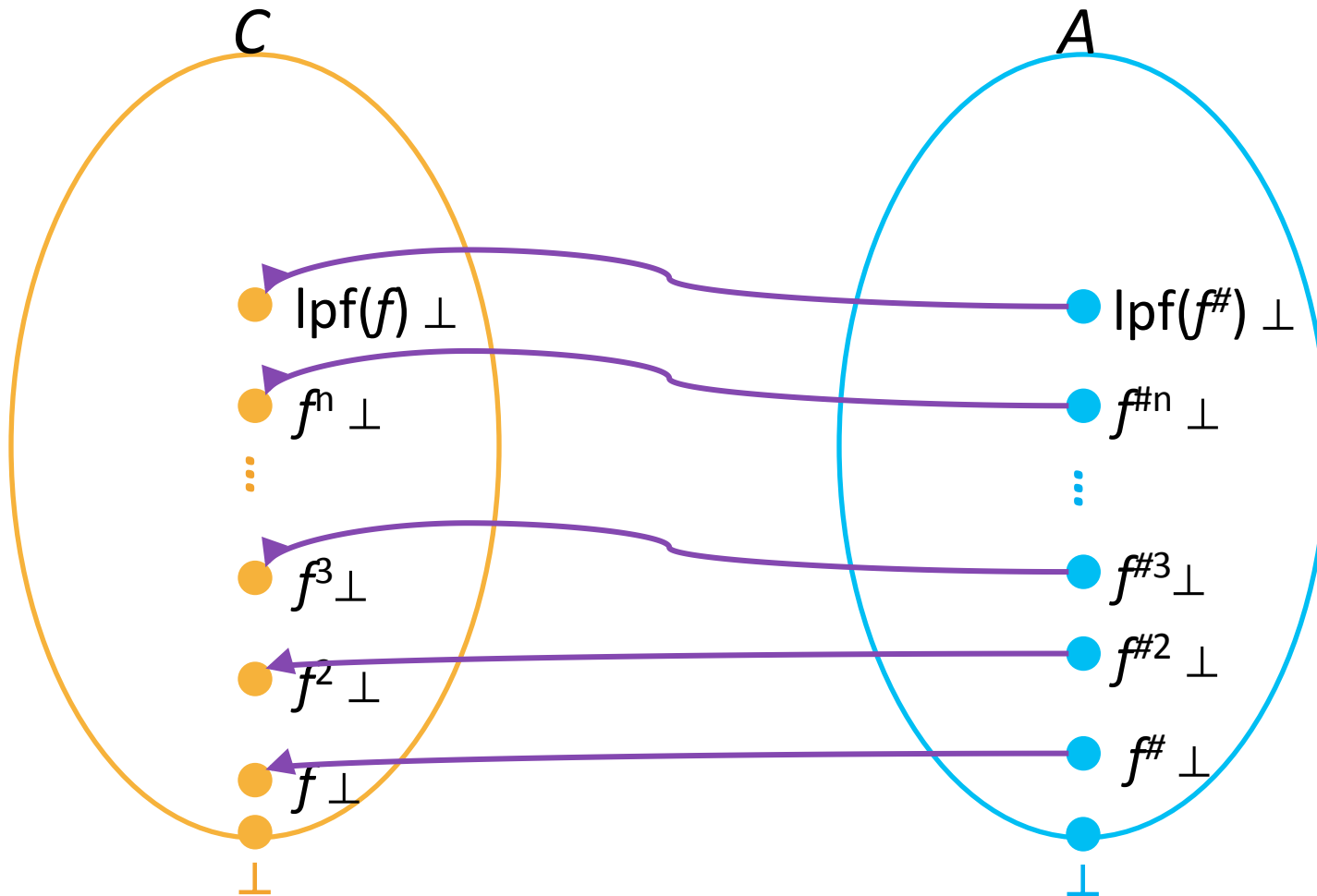
Backward complete transformer

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



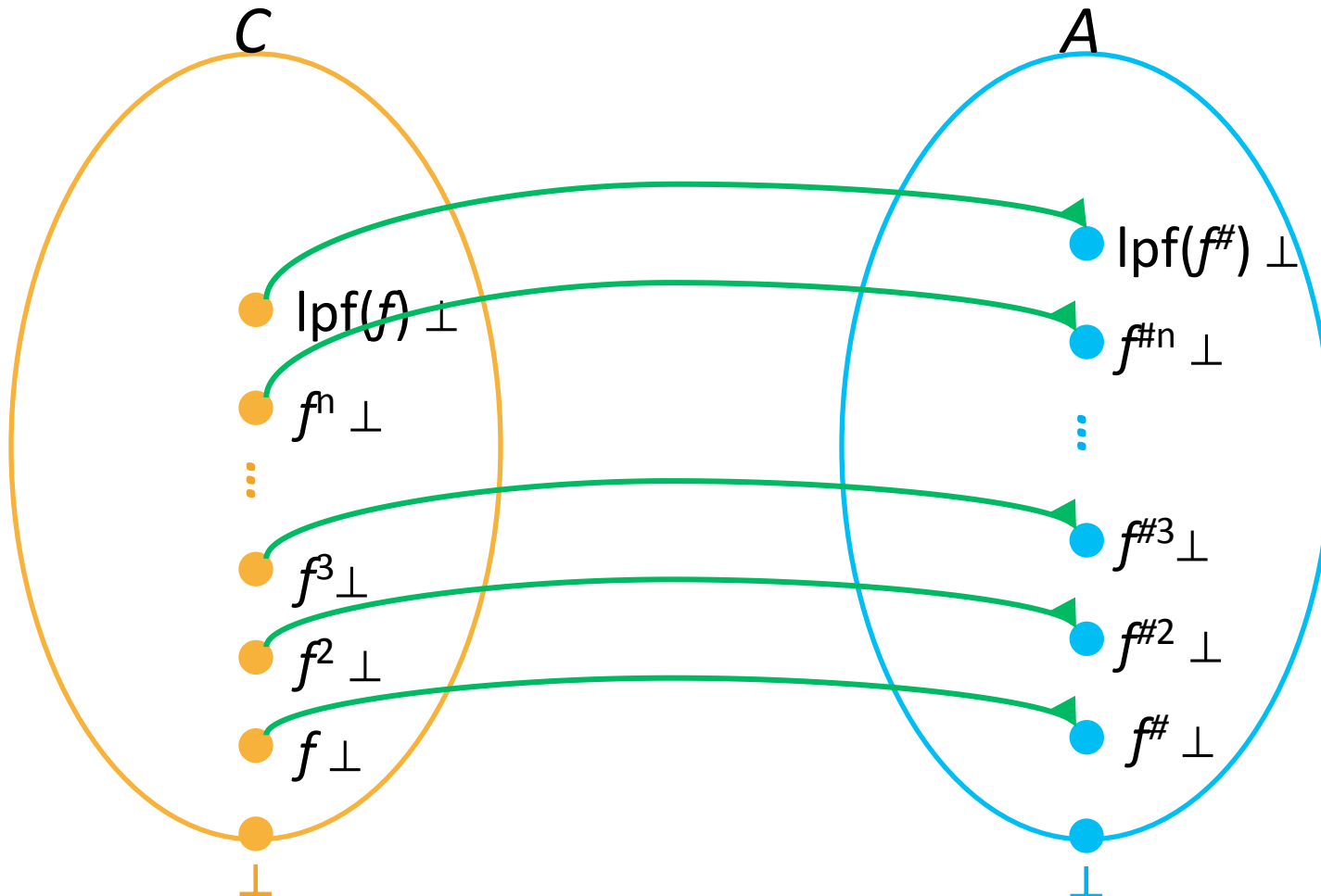
Global (backward) completeness

$$\begin{aligned} \forall a: f(\gamma(a)) &= \gamma(f^\#(a)) && \Rightarrow \forall a: f^n(\gamma(a)) = \gamma(f^{\#n}(a)) \\ &&& \Rightarrow \forall a \in D^A: \text{lfp}(f^n)(\gamma(a)) = \gamma(\text{lfp}(f^{\#n})(a)) \\ &&& \Rightarrow \text{lfp}(f) \perp = \text{lfp}(f^\#) \perp \end{aligned}$$

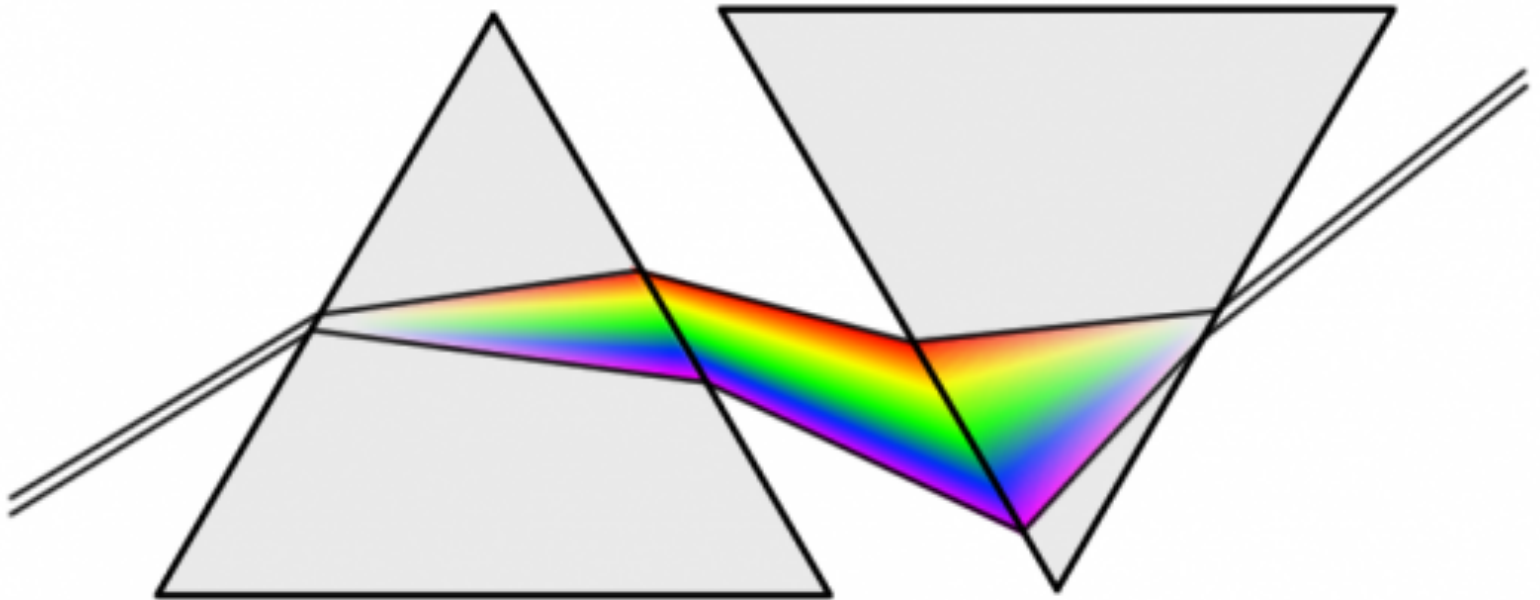


Global (forward) completeness

$$\begin{aligned} \forall c \in D^C : \alpha(f(c)) = f^\#(\alpha(c)) &\Rightarrow \forall c \in D^C : \alpha(f^n(c)) = f^{\#\,n}(\alpha(c)) \\ &\Rightarrow \forall c \in D^C : \alpha(\text{lfp}(f)(c)) = \text{lfp}(f^\#)(\alpha(c)) \\ &\Rightarrow \text{lfp}(f) \perp = \text{lfp}(f^\#) \perp \end{aligned}$$



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!");  
}
```

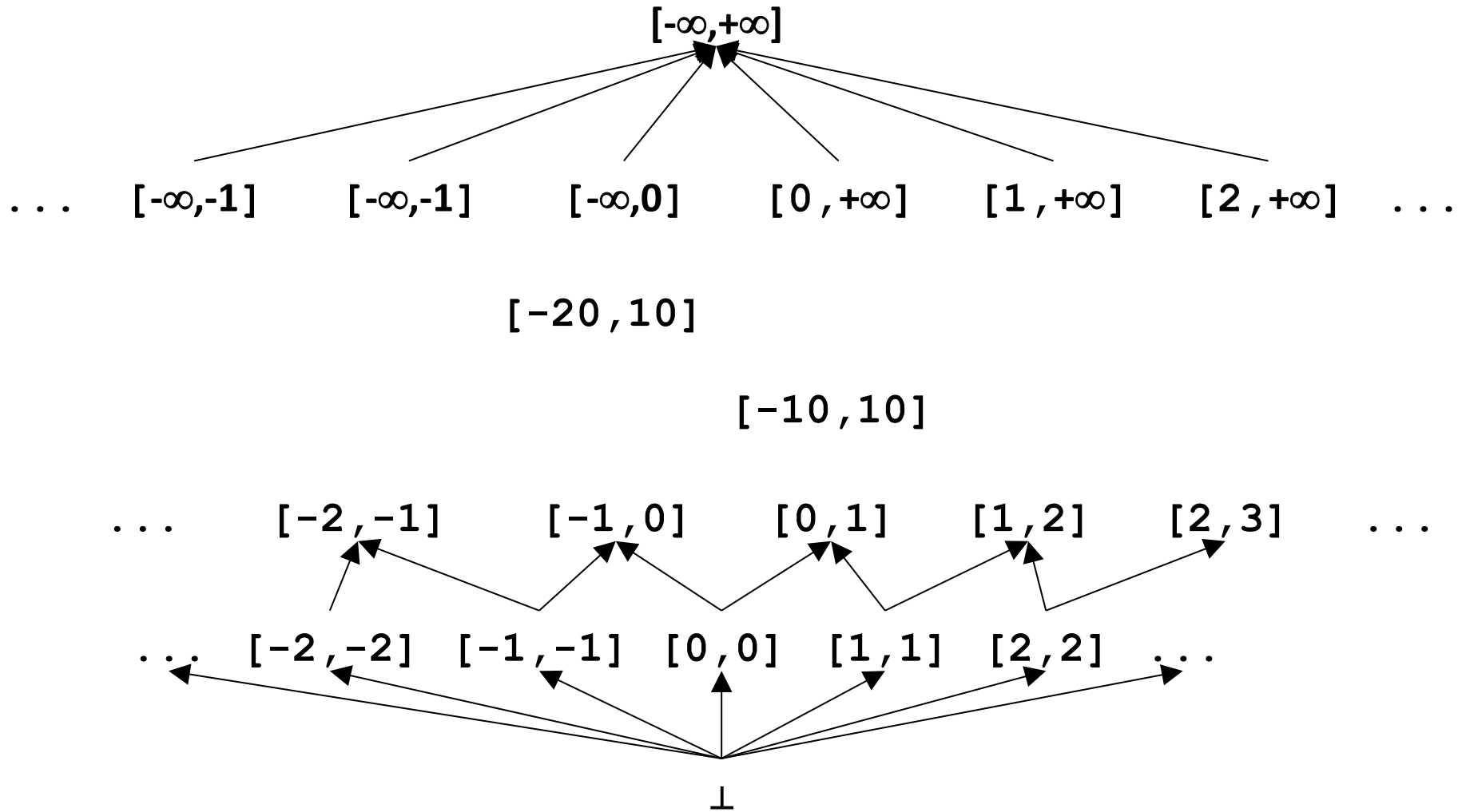
RelProd(CP, VE)

```
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 or $-\infty$
 - H is either an integer or $+\infty$
- A (non-relational) numeric domain

Intervals lattice for variable x



Intervals lattice for variable x

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

Intervals lattice for variable x

- $D^{\text{int}}[x] = \{ (L,H) \mid L \in -\infty, \mathbf{Z} \text{ and } H \in \mathbf{Z}, +\infty \text{ and } L \leq H \}$
- \perp
- $\top = [-\infty, +\infty]$
- $\sqsubseteq = ?$
 - $[1,2] \sqsubseteq [3,4]$ **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 \perp
 - $[1,2] \sqcap [3,4] = \perp$
 - $[1,4] \sqcap [3,4] = [3,4]$
 - $[1,1] \sqcap [1,+\infty] = [1,1]$
- Check that indeed $x \sqsubseteq y$ if and only if $x \sqcup y = y$

Interval domain for programs

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

Interval domain for programs

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

Interval domain for programs

- $D^{\text{int}}[x] = \{ (L,H) \mid L \in -\infty, \mathbf{Z} \text{ and } H \in \mathbf{Z}, +\infty \text{ and } L \leq H \}$
- For a program with variables $Var = \{x_1, \dots, x_k\}$
- $D^{\text{int}}[Var] = D^{\text{int}}[x_1] \times \dots \times D^{\text{int}}[x_k]$
- 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
 - $c + +\infty = +\infty$
 - $\text{remove}(S, x) = S$ without any x -constraints
 - $\text{lb}(S, x) =$

Assignment transformers

- $\llbracket x := c \rrbracket \# S = ?$
- $\llbracket x := y \rrbracket \# S = ?$
- $\llbracket x := y+c \rrbracket \# S = ?$
- $\llbracket x := y+z \rrbracket \# S = ?$
- $\llbracket x := y*c \rrbracket \# S = ?$
- $\llbracket x := y*z \rrbracket \# S = ?$

Assignment transformers

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

assume transformers

- $\llbracket \text{assume } x=c \rrbracket \# S = ?$
- $\llbracket \text{assume } x < c \rrbracket \# S = ?$
- $\llbracket \text{assume } x=y \rrbracket \# S = ?$
- $\llbracket \text{assume } x \neq c \rrbracket \# S = ?$

assume transformers

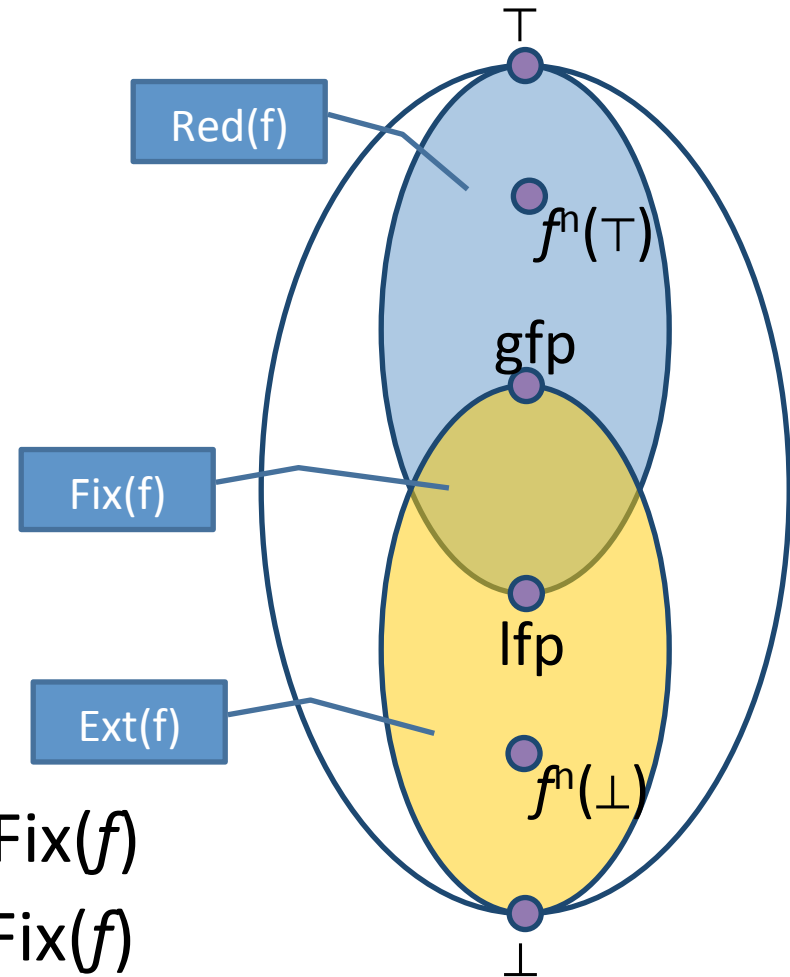
- $\llbracket \text{assume } x=c \rrbracket \# S = S \sqcap \{x \geq c, x \leq c\}$
- $\llbracket \text{assume } x < c \rrbracket \# S = S \sqcap \{x \leq c-1\}$
- $\llbracket \text{assume } x=y \rrbracket \# S = S \sqcap \{x \geq \text{lb}(S,y), x \leq \text{ub}(S,y)\}$
- $\llbracket \text{assume } x \neq c \rrbracket \# S = ?$

assume transformers

- $\llbracket \text{assume } x=c \rrbracket \# S = S \sqcap \{x \geq c, x \leq c\}$
- $\llbracket \text{assume } x < c \rrbracket \# S = S \sqcap \{x \leq c-1\}$
- $\llbracket \text{assume } x=y \rrbracket \# S = S \sqcap \{x \geq \text{lb}(S,y), x \leq \text{ub}(S,y)\}$
- $\llbracket \text{assume } x \neq c \rrbracket \# S = (S \sqcap \{x \leq c-1\}) \sqcup (S \sqcap \{x \geq c+1\})$

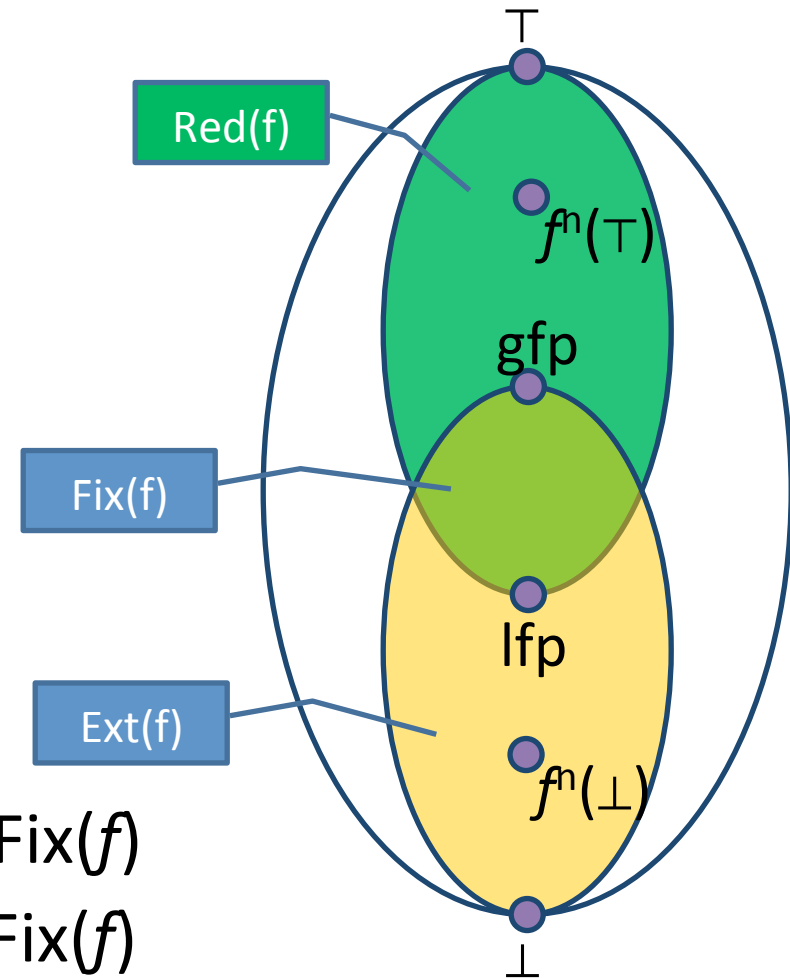
Effect of function f on lattice elements

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$
- $f: D \rightarrow D$ **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

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$
- $f: D \rightarrow D$ **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 $L = (D, \sqsubseteq, \sqcup, \perp)$ 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:

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

Fixed-point theorem [Kleene]

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

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

Resulting algorithm

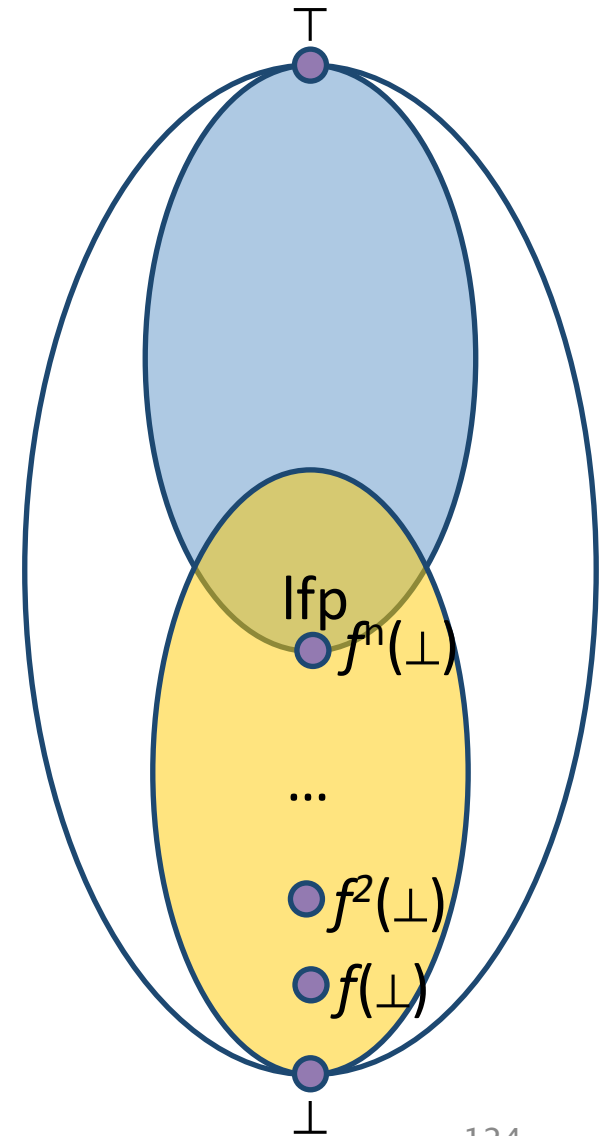
- Kleene's fixed point theorem gives a constructive method for computing the lfp

Mathematical definition

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

Algorithm

```
 $d := \perp$   
while  $f(d) \neq d$  do  
     $d := d \sqcup f(d)$   
return  $d$ 
```



Chaotic iteration

- Input:
 - A cpo $L = (D, \sqsubseteq, \sqcup, \perp)$ satisfying ACC
 - $L^n = L \times L \times \dots \times L$
 - 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: $\text{lfp}(f)$
- A worklist-based algorithm

```
for i:=1 to n do
  X[i] :=  $\perp$ 
WL = {1,...,n}
while WL  $\neq \emptyset$  do
  j := pop WL // choose index non-deterministically
  N := F[i](X)
  if N  $\neq$  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!");  
}
```

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

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!");  
}
```

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

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!");  
}
```

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

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!");  
    }  
}
```

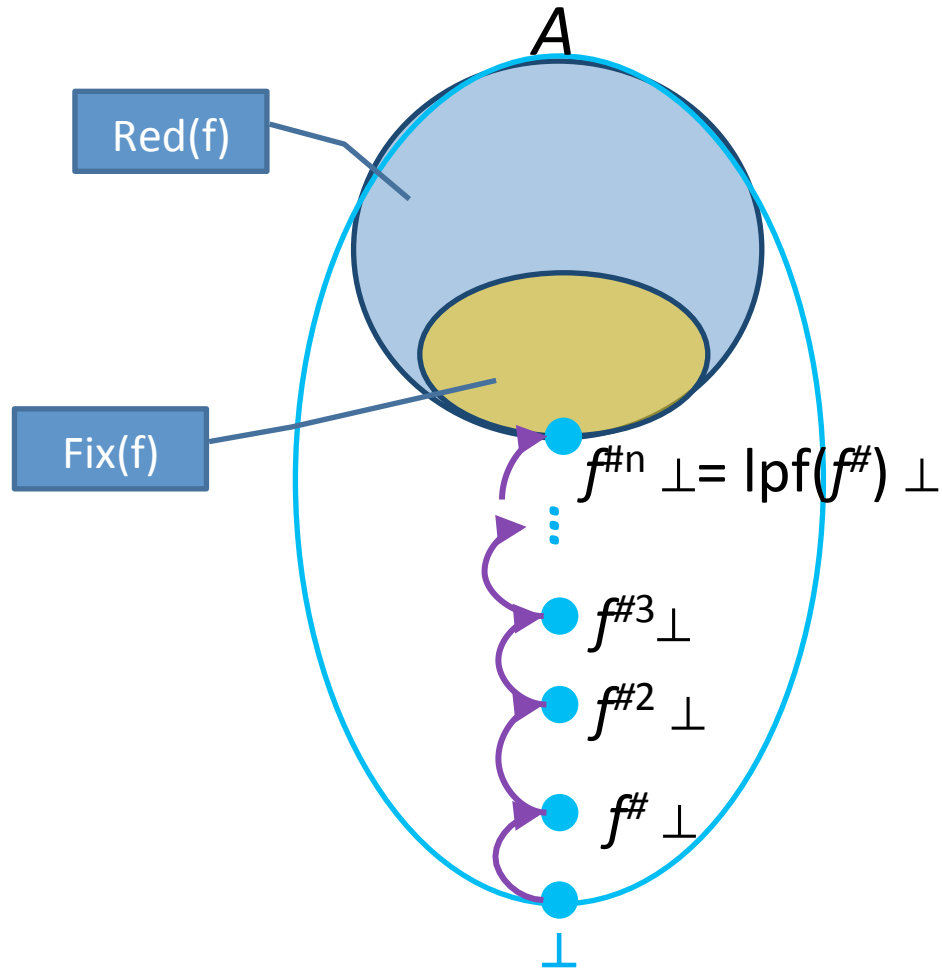
Widening

- 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 than with finite-height domains (widening non-monotone)

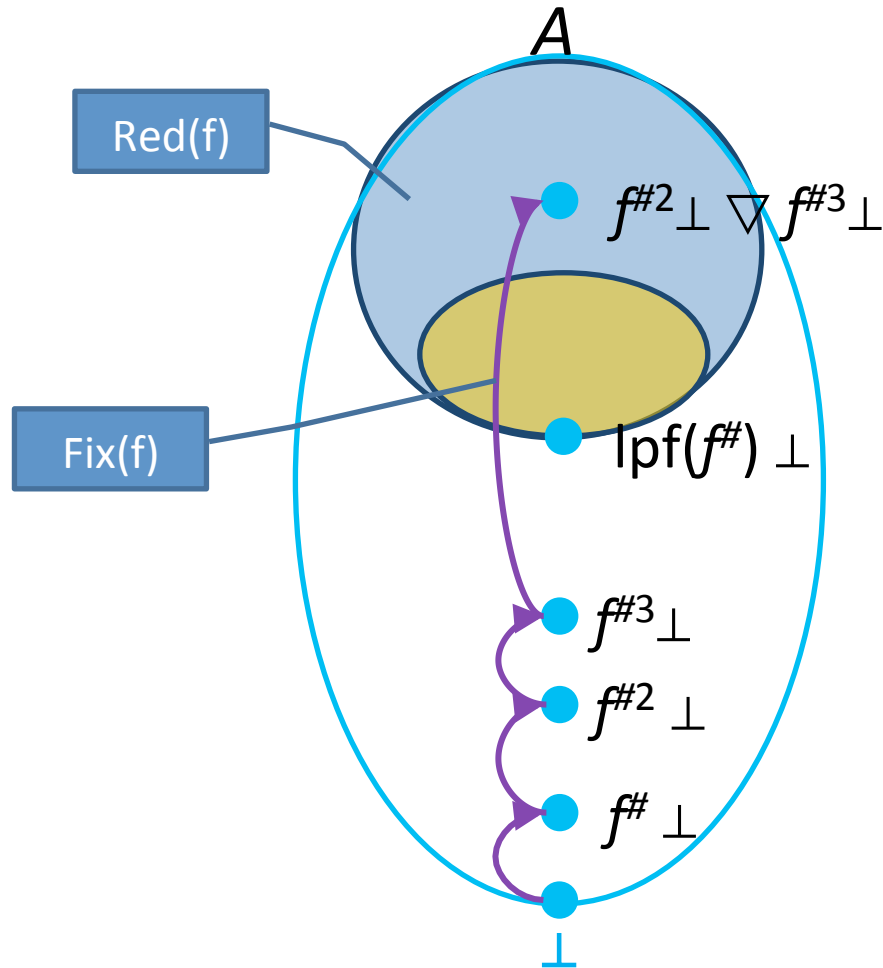
Formal definition

- For all elements $d_1 \sqcup d_2 \sqsubseteq d_1 \nabla d_2$
- For all ascending chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \dots$ the following sequence is finite
 - $y_0 = d_0$
 - $y_{i+1} = y_i \nabla d_{i+1}$
- For a monotone function $f : D \rightarrow D$ define
 - $x_0 = \perp$
 - $x_{i+1} = x_i \nabla f(x_i)$
- Theorem:
 - There exists 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

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

Semantic equations with widening

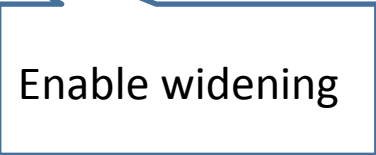
```
public void loopExample() {  
R[0]  int x = 7; R[1]  
R[2]  while (x < 1000) {  
R[3]      ++x; R[4]  
      }  
R[5]  if (!(x == 1000))  
R[6]      error("Unable to prove x == 1000!");  
}
```

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

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

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

Non monotonicity of widening

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

Analysis results with widening

Analyzing method loopExample

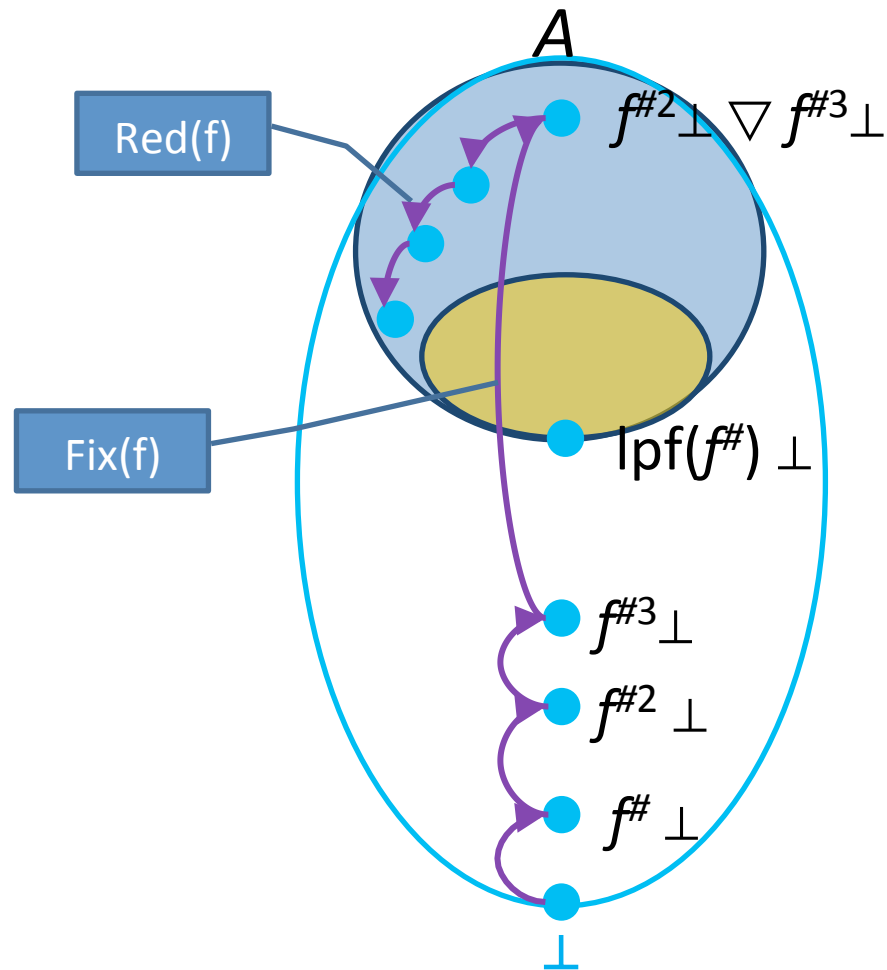
```
-----  
Solving the following equation system =  
V[0] = true // this := @this: IntervalExample  
V[1] = AssignTopTransformer(V[0]) // this := @this: IntervalExample  
V[2] = AssignConstantToVarTransformer(V[1]) // x = 7  
V[3] = V[2] // goto [?= (branch)]  
V[4] = AssignAddExprToVarTransformer(V[5]) // 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
- $y \sqsubseteq x \Rightarrow y \sqsubseteq (x \Delta y) \sqsubseteq x$
- For all decreasing chains $x_0 \supseteq x_1 \supseteq \dots$ the following sequence is finite
 - $y_0 = x_0$
 - $y_{i+1} = y_i \Delta x_{i+1}$
- For a monotone function $f: D \rightarrow D$ and $x_k \in \text{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$ define
 - $y_0 = x$
 - $y_{i+1} = y_i \Delta f(y_i)$
- Theorem:
 - There exists k such that $y_{k+1} = y_k$
 - $y_k \in \text{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$

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:

$R[0] = \top$

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

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

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

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

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

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

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

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.
```



Precise invariant

