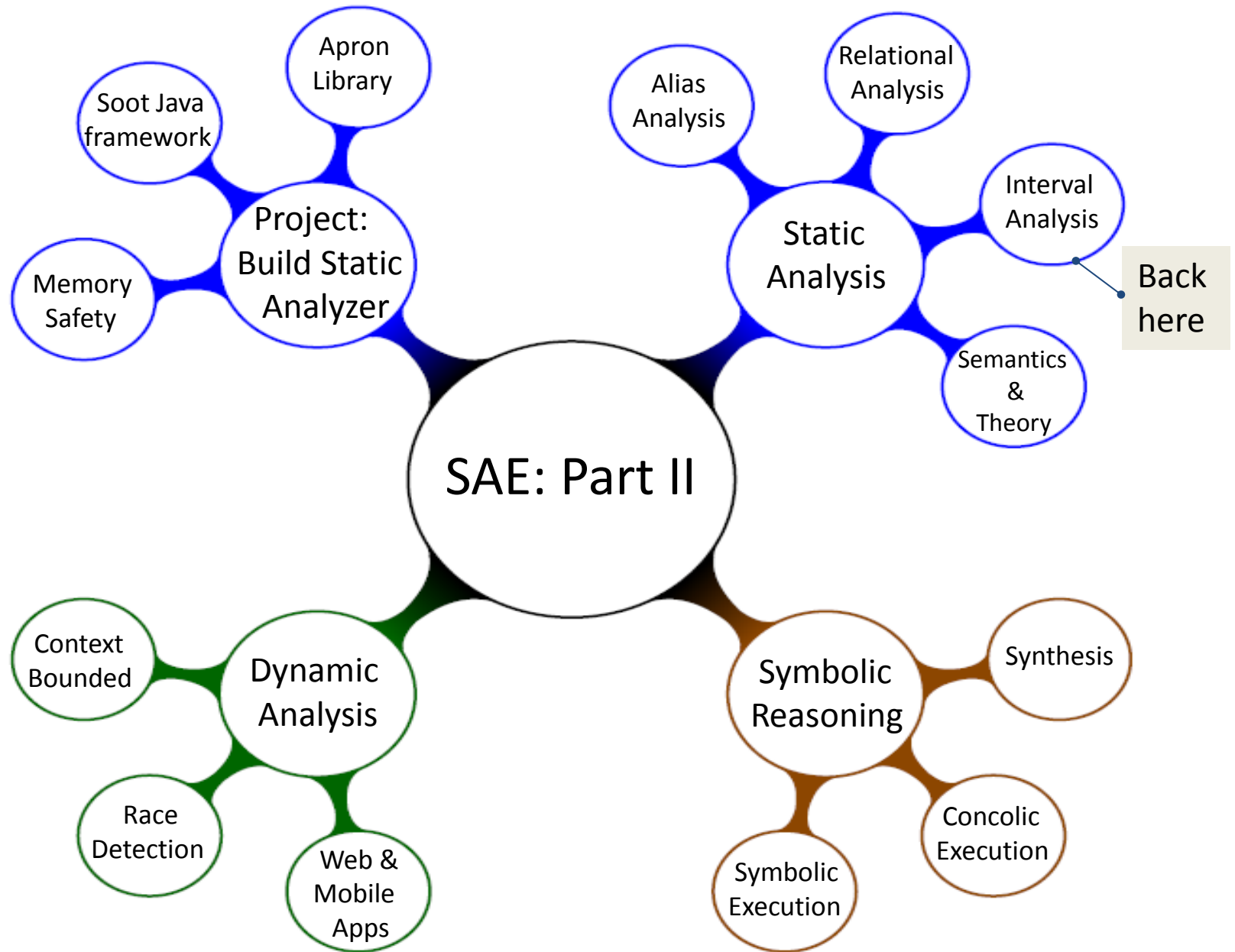


# Software Architecture and Engineering: Part II

ETH Zurich, Spring 2014  
Prof. Martin Vechev



# So Far

So far, we learned how to describe the behavior of the program as a fixed point of a function.

Then, we learn a bunch of mathematical concepts such as lattices, functions, fixed points, function approximation, etc.

Next, we will see how to put these together in order to automatically prove properties about our programs

# Starting point

Our starting point is a domain where each element of the domain is a **set of traces**. The domain of traces is a **complete lattice**:

$$(\wp(\Sigma^+), \subseteq, \cup, \cap, \emptyset, \Sigma^+)$$

$\Sigma = \text{Lab} \times \text{Store}$  (from before)

$\Sigma^+$  denotes all **finite** traces.

# Representing $\llbracket P \rrbracket$

Recall the function  $F$ :

$$F(S) = I \cup \{ \pi \cdot c \cdot c' \mid \pi \cdot c \in S \wedge c \rightarrow c' \}$$

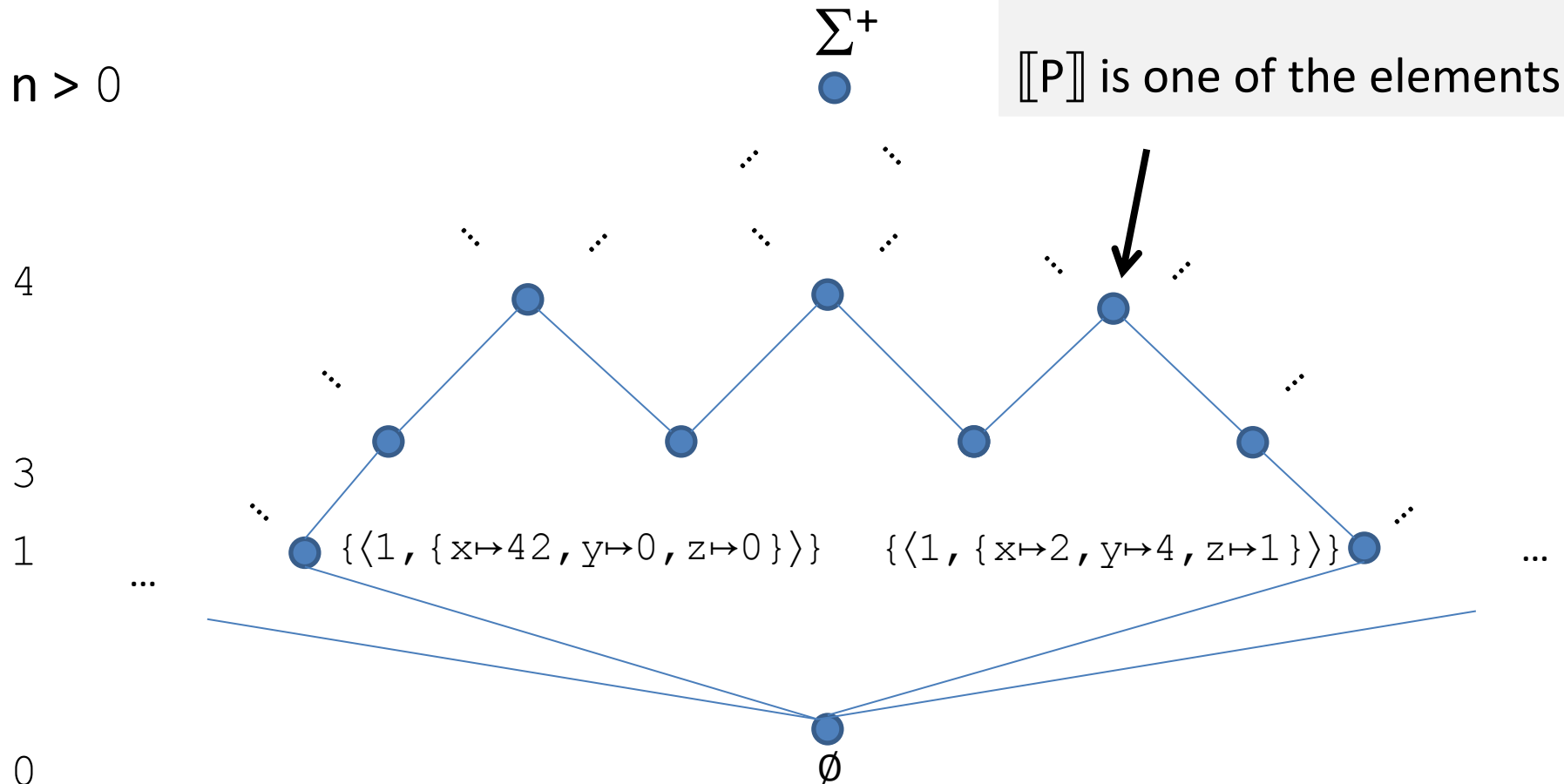
$\llbracket P \rrbracket$  is a **fixed point** of  $F$ :  $F(\llbracket P \rrbracket) = \llbracket P \rrbracket$

(in fact,  $\llbracket P \rrbracket$  is the **least fixed point** of  $F$ )

# Starting Point: Domain of Traces

Size of Set of Traces:

$n > 0$



# Abstraction

$\llbracket P \rrbracket$  can be an infinite set. The idea is to abstract it such that its abstraction is representable with a bounded set of facts. Then, we prove properties on that abstraction instead of on  $\llbracket P \rrbracket$  directly.

How we abstract  $\llbracket P \rrbracket$  depends on the **type of property** we want to prove about our program.

Here, we will focus **on state properties** (like assertions), as they are most prevalent in practice.

# Abstraction

Our first abstraction is a state abstraction. Given a **set of traces**, we record only the states of each trace in the set.

The abstract domain of states forms a complete lattice:

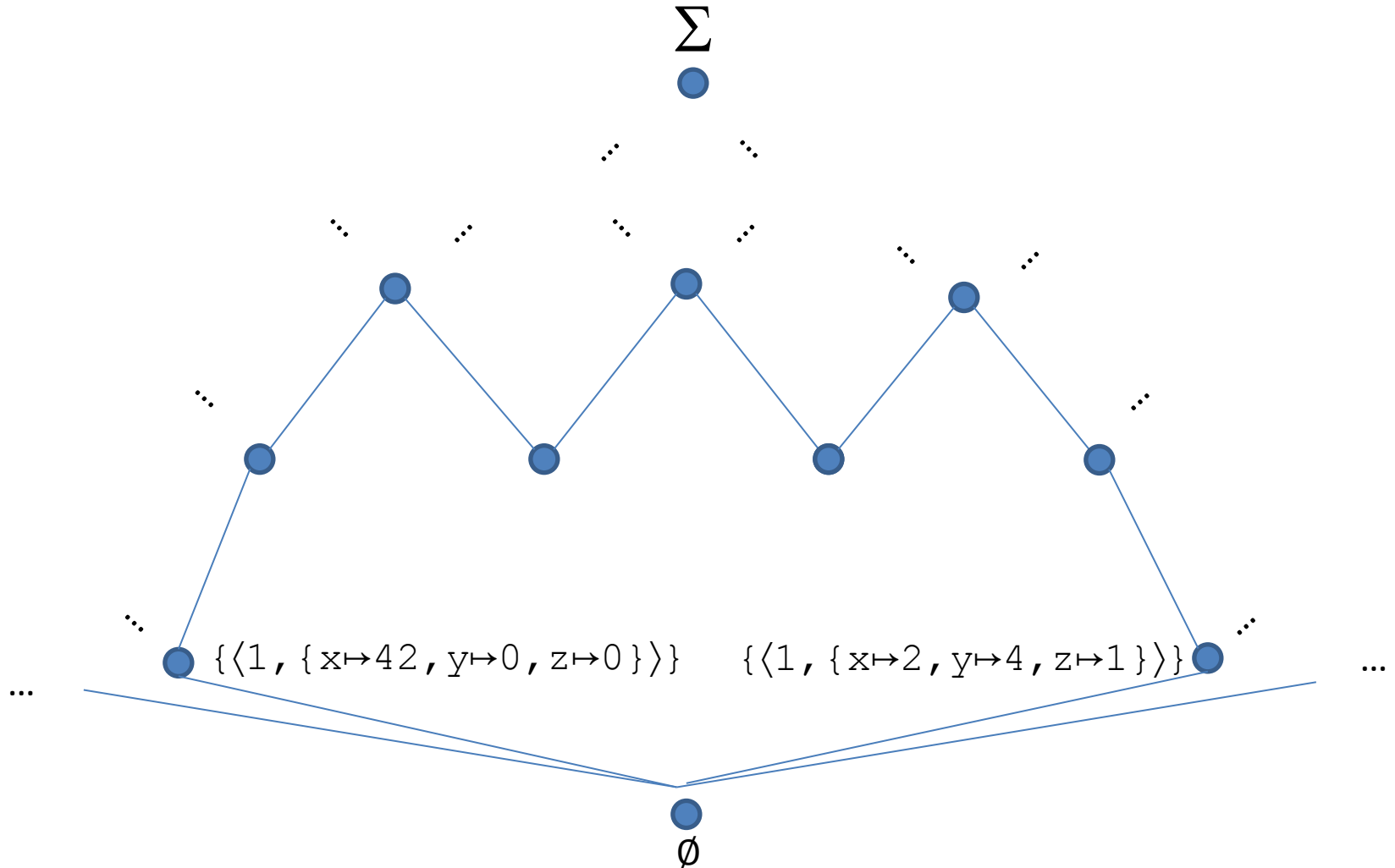
$$(\wp(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma)$$



# Domain of States

Size of Set:

$n > 0$



# Abstraction

$$\alpha^s: \wp(\Sigma^+) \rightarrow \wp(\Sigma)$$

$$\alpha^s(\text{Traces}) = \{ \pi_i \mid i \in [0, |\pi|) \wedge \pi \in \text{Traces} \}$$

Here,  $\pi_i$  denotes the state at position  $i$  in trace  $\pi$

We abstract a set of traces by taking each trace in turn and obtaining its states

# Abstraction

For each trace, we **lose the relationship** between states, as we no longer know to which trace each state belongs to and how it is related to other states in that trace.

We also **lose how many times** each state occurs in a trace.

# Example of Abstraction

$$\alpha^s(\emptyset) = \emptyset$$

$$\alpha^s(\{a \cdot b \cdot a \cdot d\}) = \{a, b, d\}$$

$$\alpha^s(\{a \cdot b \cdot c, a \cdot d\}) = \{a, b, c, d\}$$

$$\alpha^s(\{a \cdot b \cdot a \cdot d\}) = \{a, b\} \quad \text{this one is incorrect !}$$

# Concretization

$$\gamma^S: \wp(\Sigma) \rightarrow \wp(\Sigma^+)$$

$$\gamma^S(S) = \{c_0 \cdot \dots \cdot c_n \mid n \geq 0 \wedge \forall i \in [0, n]: c_i \in S\}$$

Because we do not know where the states came from, we need to build **all finite traces** that can be constructed from these states. We want to make sure that we never miss a trace whose abstraction produces a set **that is a subset of the set S**.

# Examples of Concretization

$$\gamma^s(\emptyset) = \emptyset$$

$$\gamma^s(\{a, b, d\}) = \{a, a \cdot a, a \cdot a \cdot a, b, b \cdot b, a \cdot d \cdot b, \dots\}$$

$$\gamma^s(\{d\}) = \{d, d \cdot d, d \cdot d \cdot d, \dots\}$$

$$\gamma^s(\{d\}) = \{d\} \quad \text{This one is incorrect}$$

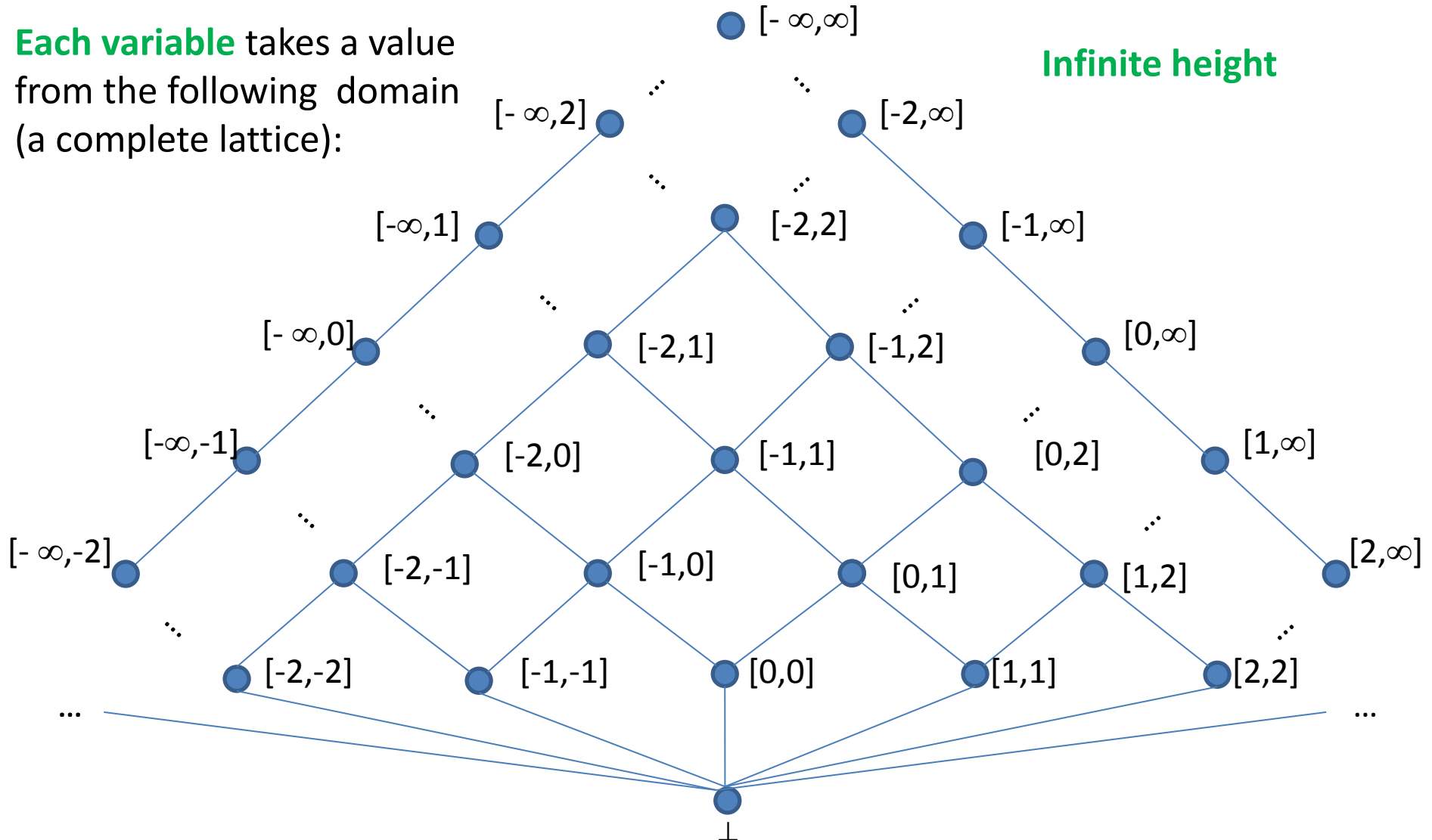
# Interval Abstraction

If we are interested in properties that involve the range of values that a variable can take, we can abstract the set of states into a map which captures the range of values that a variable can take.

# Interval Domain

Each variable takes a value from the following domain (a complete lattice):

Infinite height





# Interval Domain: Lets Define it

Let the interval domain be:  $(L^i, \sqsubseteq_i, \sqcup_i, \sqcap_i, \perp_i, [-\infty, \infty])$

We denote  $Z^\infty = Z \cup \{-\infty, \infty\}$

The set  $L^i = \{ [x, y] \mid x, y \in Z^\infty, x \leq y \} \cup \{\perp_i\}$

For a set  $S \subseteq Z^\infty$ ,  $\min(S)$  returns the minimum number in  $S$ ,  $\max(S)$  returns the maximum number in  $S$ .

- $[a, b] \sqsubseteq_i [c, d]$  if  $c \leq a$  and  $b \leq d$
- $[a, b] \sqcup_i [c, d] = [\min(\{a, c\}), \max(\{b, d\})]$
- $[a, b] \sqcap_i [c, d] = [\text{meet}(\max(\{a, c\}), \min(\{b, d\}))]$   
where  $\text{meet}(a, b)$  returns  $a$  if  $a \leq b$  and  $\perp_i$  otherwise

# Intervals: Applied to Programs

The  $\mathbb{L}^i$  domain simply defines intervals, but to apply it to programs we need to take into account program labels (program counters) and program variables.

Therefore, for programs, we use the domain  $\text{Lab} \rightarrow (\text{Var} \rightarrow \mathbb{L}^i)$

That is, at each label and for each variable, we will keep the range for that variable. This domain is also a **complete lattice**.

The operators of  $\mathbb{L}^i$   $\sqsubseteq_i, \sqcup_i, \sqcap_i$  are **lifted directly** to both domains:

- $\text{Var} \rightarrow \mathbb{L}^i$
- $\text{Lab} \rightarrow (\text{Var} \rightarrow \mathbb{L}^i)$

# Intervals: Applied to Programs

$$\begin{aligned}\alpha^{\perp}: \quad \wp(\Sigma) &\rightarrow (\text{Lab} \rightarrow (\text{Var} \rightarrow L^{\perp})) \\ \gamma^{\perp}: \quad (\text{Lab} \rightarrow (\text{Var} \rightarrow L^{\perp})) &\rightarrow \wp(\Sigma)\end{aligned}$$

Using  $\alpha^{\perp}$ , we abstract a **set of states** into a map from program labels to interval ranges for each variable.

Using  $\gamma^{\perp}$ , we concretize the intervals maps to a set of **states**

Formal definition of  $\alpha^{\perp}$  and  $\gamma^{\perp}$  is left as an exercise.

Let us consider an example to give an intuition.

# Example of Abstraction and Concretization

$$\begin{aligned}
 & \alpha^i ( \\
 & \{ \langle 1, \{x \mapsto 1, y \mapsto 9, q \mapsto -2\} \rangle, \langle 1, \{x \mapsto 5, y \mapsto 9, q \mapsto -2\} \rangle, \langle 1, \{x \mapsto 8, y \mapsto 9, q \mapsto -2\} \rangle, \\
 & \quad \langle 1, \{x \mapsto 1, y \mapsto 9, q \mapsto 4\} \rangle, \langle 1, \{x \mapsto 5, y \mapsto 9, q \mapsto 4\} \rangle, \langle 1, \{x \mapsto 8, y \mapsto 9, q \mapsto 4\} \rangle \} \\
 & ) \\
 & = 1 \rightarrow (x \mapsto [1, 8], y \mapsto [9, 9], q \mapsto [-2, 4])
 \end{aligned}$$

$$\begin{aligned}
 & \gamma^i (1 \rightarrow (x \mapsto [1, 8], y \mapsto [9, 9], q \mapsto [-2, 4])) \\
 & = \{ \langle 1, \{x \mapsto 1, y \mapsto 9, q \mapsto -2\} \rangle, \langle 1, \{x \mapsto 5, y \mapsto 9, q \mapsto -2\} \rangle, \langle 1, \{x \mapsto 8, y \mapsto 9, q \mapsto -2\} \rangle, \\
 & \quad \langle 1, \{x \mapsto 1, y \mapsto 9, q \mapsto 4\} \rangle, \langle 1, \{x \mapsto 5, y \mapsto 9, q \mapsto 4\} \rangle, \langle 1, \{x \mapsto 8, y \mapsto 9, q \mapsto 4\} \rangle, \\
 & \quad \langle 1, \{x \mapsto 7, y \mapsto 9, q \mapsto 3\} \rangle, \langle 1, \{x \mapsto 3, y \mapsto 9, q \mapsto 4\} \rangle, \langle 1, \{x \mapsto 1, y \mapsto 9, q \mapsto -1\} \rangle, \\
 & \quad \dots, \dots, \dots \}
 \end{aligned}$$

Concretization includes many more states (in red) than what was abstracted...

we still need to actually **compute**  $\alpha^i \circ \alpha^s (\llbracket P \rrbracket)$   
(or an **over-approximation** of it)

# Lets re-call F again:

Recall the function F:

$$F(S) = \bot \cup \{ \pi \cdot c \cdot c' \mid \pi \cdot c \in S \wedge c \rightarrow c' \}$$

$\llbracket P \rrbracket$  is a **fixed point** of F:  $F(\llbracket P \rrbracket) = \llbracket P \rrbracket$

(in fact,  $\llbracket P \rrbracket$  is the **least fixed point** of F)

# We need to approximate $F$

We want a function  $F^i$  where:

$$F^i : (\text{Lab} \rightarrow (\text{Var} \rightarrow L^i)) \rightarrow (\text{Lab} \rightarrow (\text{Var} \rightarrow L^i))$$

such that:  $\alpha^i \circ \alpha^s (\text{lfp } F) \sqsubseteq \text{lfp } F^i$

The best transformer is:  $\alpha^i \circ \alpha^s \circ F \circ \gamma^s \circ \gamma^i$

# Lets define $F^i$

$$F^i : (\text{Lab} \rightarrow (\text{Var} \rightarrow L^i)) \rightarrow (\text{Lab} \rightarrow (\text{Var} \rightarrow L^i))$$

Here is a definition of  $F^i$  which **approximates** the best transformer but **works only on the abstract domain**:

$$F^i(m)\ell = \begin{cases} \lambda v. [-\infty, \infty] & \text{if } \ell \text{ is initial label} \\ \bigsqcup_{(\ell', \text{action}, \ell)} \llbracket \text{action} \rrbracket_i(m(\ell')) & \text{otherwise} \end{cases}$$

$$\llbracket \text{action} \rrbracket_i : (\text{Var} \rightarrow L^i) \rightarrow (\text{Var} \rightarrow L^i)$$



# what is $(\ell', \text{action}, \ell)$ ?

```
foo (int i) {  
  
1: int x := 5;  
2: int y := 7;  
  
3: if (0 ≤ i) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7: }
```

## Actions:

```
(1, x := 5, 2)  
(2, y := 7, 3)  
(3, 0 ≤ i, 4)  
(3, 0 > i, 7)  
(4, y = y + 1, 5)  
(5, i := i - 1, 6)  
(6, goto 3, 3)
```

Multiple (two) actions reach label 3

# what is $(\ell', \text{action}, \ell)$ ?

- $(\ell', \text{action}, \ell)$  is an edge in the control-flow graph
- More formally, if there exists a transition  $t = \langle \ell', \sigma' \rangle \rightarrow \langle \ell, \sigma \rangle$  in a program trace in  $\llbracket P \rrbracket$ , where  $t$  was performed by statement called **action**, then  $(\ell', \text{action}, \ell)$  must exist. This says that we are **sound**: we never miss a flow.
- However,  $(\ell', \text{action}, \ell)$  may exist even if no such transition  $t$  above occurs. In this case, the analysis would be imprecise as we would unnecessarily create more flows.

# what is $(\ell', \text{action}, \ell)$ ?

An **action** can be:

- $b \in \text{BExp}$                       boolean expression in a conditional
- $x := a$                               here,  $a \in \text{AExp}$
- skip

Next, we will define the effect of some of these things formally, while with others we will proceed by example.

The key point is to make sure that  $\llbracket \text{action} \rrbracket_i$  produces sound and precise results.

$$\llbracket x := a \rrbracket_i$$

$$\llbracket x := a \rrbracket_i (m) = m [x \mapsto v] \quad , \quad \text{where } \langle a, m \rangle \Downarrow_i v$$

$\langle a, m \rangle \Downarrow_i v$  says that given a map  $m$ , the expression  $a$  evaluates to a value  $v \in \mathbb{L}^i$

The operational semantics rules for expression evaluation are as before, except:

- any constant  $Z$  is abstracted to an element in  $\mathbb{L}^i$
- operators  $+$ ,  $-$  and  $*$  are re-defined for the Interval domain

# Arithmetic Expressions

If we add  $\perp_i$  to any other element, we get  $\perp_i$ .

If both operands are not  $\perp_i$ , we get:

$$[x, y] + [z, q] = [x + z, y + q]$$

what about  $*$  ?

is  $[x, y] * [z, q] = [x * z, y * q]$  **sound** ?

$$\llbracket b \rrbracket_i$$

Let us first look at the expression:  $a_1 \text{ c } a_2$

For a map  $m$ , we need to define :  $\llbracket a_1 \text{ c } a_2 \rrbracket_i(m)$

which produces another map as a result.

Here,  $c$  is a condition such as:  $\leq, =, <$

# Example: what is $\llbracket x \leq y \rrbracket$ ?

Suppose we have the program:

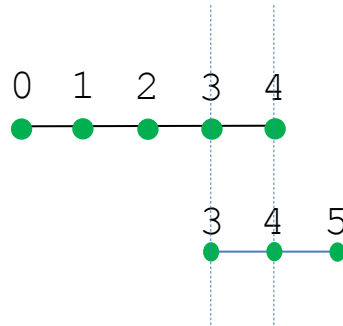
```
// Here, x is [0,4] and y is [3,5]
if (x ≤ y) {
  1: ...
}
```

What does  $\llbracket x \leq y \rrbracket$  produce at label 1 ?

That is, what are the intervals for  $x$  and  $y$  at label 1 ?

# Definition of $[l_1, u_1] \leq [l_2, u_2]$

what should  $[0, 4] \leq [3, 5]$  produce ?



one answer is:  $([0, 3], [3, 5])$ . Is it **sound** ?

another non-comparable answer is:  $([0, 4], [4, 5])$ . Is it **sound** ?

can you find a **more precise** answer ?



# Definition of $[l_1, u_1] \leq [l_2, u_2]$

$$[l_1, u_1] \leq [l_2, u_2] = ([l_1, u_1] \sqcap_i [-\infty, u_2], [l_1, \infty] \sqcap_i [l_2, u_2])$$

$$\begin{aligned} [0, 4] \leq [3, 5] &= ([0, 4] \sqcap_i [-\infty, 5], [0, \infty] \sqcap_i [3, 5]) \\ &= ([0, 4], [3, 5]) \end{aligned}$$

Exercise: define  $<$  and  $=$

# Evaluating $\llbracket b \rrbracket_i$

$$\llbracket b_1 \vee b_2 \rrbracket_i(m) = \llbracket b_1 \rrbracket_i(m) \sqcup \llbracket b_2 \rrbracket_i(m)$$

$$\llbracket b_1 \wedge b_2 \rrbracket_i(m) = \llbracket b_1 \rrbracket_i(m) \sqcap \llbracket b_2 \rrbracket_i(m)$$

# $F^i$ on an example

```
foo (int i) {  
  
1:  int x := 5;  
2:  int y := 7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

$$F^i(m)1 = \lambda v. [-\infty, \infty]$$

$$F^i(m)2 = \llbracket x := 5 \rrbracket_i (m(1))$$

$$F^i(m)3 = \llbracket y := 7 \rrbracket_i (m(2)) \sqcup \llbracket \text{goto } 3 \rrbracket_i (m(6))$$

$$F^i(m)4 = \llbracket i \geq 0 \rrbracket_i (m(3))$$

$$F^i(m)5 = \llbracket y := y + 1 \rrbracket_i (m(4))$$

$$F^i(m)6 = \llbracket i := i - 1 \rrbracket_i (m(5))$$

$$F^i(m)7 = \llbracket i < 0 \rrbracket_i (m(3))$$

# Fixed point of $F^i$

Let us compute the least fixed point of  $F^i$

# Iterate 0

The collection of these lines denote the current iterate. The iterate is a map

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

2:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 1

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 2

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 3

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$



# Iterate 4

Notice how we propagated to both labels 4 and 7 **at the same time**

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 5

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 6

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 7

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
3:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, \infty]$   
4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$   
5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$   
6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$   
7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 8

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, -1]$

# Iterate 9

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, -1]$

# Iterate 10

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
3:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, \infty]$   
4:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [0, \infty]$   
5:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [0, \infty]$   
6:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [-1, \infty]$   
7:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, -1]$

# Iterate 11

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, -1]$



# Iterate 12

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, -1]$

# Iterate 13

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 9], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, -1]$

# Iterate 14

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
3:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, \infty]$   
4:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [0, \infty]$   
5:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [0, \infty]$   
6:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [-1, \infty]$   
7:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, -1]$

# Iterate 15

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
3:  $x \rightarrow [5, 5], y \rightarrow [7, 10], i \rightarrow [-\infty, \infty]$   
4:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [0, \infty]$   
5:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [0, \infty]$   
6:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [-1, \infty]$   
7:  $x \rightarrow [5, 5], y \rightarrow [7, 9], i \rightarrow [-\infty, -1]$

# Iterate 16

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 10], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 10], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 10], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 10], i \rightarrow [-\infty, -1]$

The issue is that the iterates:

$$F^{(0)}(\lambda \ell . \lambda v. \perp_i), F^{(1)}(\lambda \ell . \lambda v. \perp_i), F^{(2)}(\lambda \ell . \lambda v. \perp_i), \dots$$

will keep going on forever as the value of variable **y** will keep increasing. Hence, we will not be able to compute all of the iterates that we need in order to apply the fixed point theorem.

what should we do in this case ?

Generally, if we have a complete lattice  $(L, \sqsubseteq, \sqcup, \sqcap)$  and a monotone function  $F$ , if the height is infinite or the computation of the iterates of  $F$  takes too long, we need to find a way to **approximate** the least fixed point of  $F$ .

The interval domain and its function  $F^i$  is an example of this case.

We need to find a way to compute an element  $A$  such that:

$$\text{lfp}^{\sqsubseteq} F \sqsubseteq A$$

# Widening Operator

The operator  $\nabla: L \times L \rightarrow L$  is called a **widening** operator if:

- $\forall a, b \in L: a \sqcup b \sqsubseteq a \nabla b$  (widening approximates the join)
- if  $x^0 \sqsubseteq x^1 \sqsubseteq x^2 \sqsubseteq \dots \sqsubseteq x^n \sqsubseteq \dots$  is an increasing sequence then  $y^0 \sqsubseteq y^1 \sqsubseteq y^2 \sqsubseteq \dots \sqsubseteq y^n$  **stabilizes** after a **finite number of steps**

where  $y^0 = x^0$  and  $\forall i \geq 0: y^{i+1} = y^i \nabla x^{i+1}$

Note that widening is completely **independent of the function F.**

Much like the join, it is an operator defined for the **particular domain.**



# Useful Theorem

If  $L$  is a complete lattice,  $\nabla: L \times L \rightarrow L$ ,  $F: L \rightarrow L$  is monotone  
Then the sequence:

$$\begin{aligned}y^0 &= \perp \\y^1 &= y^0 \nabla F(y^0) \\y^2 &= y^1 \nabla F(y^1) \\&\dots \\y^n &= y^{n-1} \nabla F(y^{n-1})\end{aligned}$$

will stabilize after a finite number of steps with  $y^n$  being a **post-fixedpoint** of  $F$ .

By Tarski's theorem, we know that a post-fixedpoint is above the least fixed point. Hence, it follows that:  $\text{lfp} \sqsubseteq F \sqsubseteq y^n$

# Widening for Interval Domain

Let us define a widening operator for the intervals

$$\nabla_i: \mathbb{L}^i \times \mathbb{L}^i \rightarrow \mathbb{L}^i$$

$[a, b] \nabla_i [c, d] = [e, f]$  where:

if  $c < a$ , then  $e = -\infty$ , else  $e = a$

if  $d > b$ , then  $f = \infty$ , else  $f = b$

if one of the operands is  $\perp$  the result is the other operand.

The basic intuition is that if we see that an end point is unstable, we move its value to the extreme case.

**Exercise:** show this operator satisfies the conditions for widening.

# Examples: widening for Interval

$$[1, 2] \nabla_i [0, 2] =$$

$$[0, 2] \nabla_i [1, 2] =$$

$$[1, 5] \nabla_i [1, 5] =$$

$$[2, 3] \nabla_i [2, 4] =$$

# Examples: widening for Interval

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

$$[0, 2] \nabla_i [1, 2] = [0, 2]$$

$$[1, 5] \nabla_i [1, 5] = [1, 5]$$

$$[2, 3] \nabla_i [2, 4] = [2, \infty]$$

Let us again consider our program with the Interval domain  
but this time we will apply the widening operator

# Iterate 0

$$it^0 = \perp$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
2:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$   
7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 1

$$\begin{aligned}it^1 &= it^0 \nabla F(it^0) \\ &= \perp \nabla F(\perp) \\ &= F(\perp)\end{aligned}$$

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
   }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 2

$$it^2 = it^1 \nabla F(it^1)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$



# Iterate 3

$$it^3 = it^2 \nabla F(it^2)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

5:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

# Iterate 4

Notice how we propagated to both labels 4 and 7 **at the same time**

$$it^4 = it^3 \nabla F(it^3)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
  }  
7:  
}
```

$$1: x \rightarrow [-\infty, \infty], \quad y \rightarrow [-\infty, \infty], \quad i \rightarrow [-\infty, \infty]$$

$$2: x \rightarrow [5, 5], \quad y \rightarrow [-\infty, \infty], \quad i \rightarrow [-\infty, \infty]$$

$$3: x \rightarrow [5, 5], \quad y \rightarrow [7, 7], \quad i \rightarrow [-\infty, \infty]$$

$$4: x \rightarrow [5, 5], \quad y \rightarrow [7, 7], \quad i \rightarrow [0, \infty]$$

$$5: x \rightarrow \perp_i, \quad y \rightarrow \perp_i, \quad i \rightarrow \perp_i$$

$$6: x \rightarrow \perp_i, \quad y \rightarrow \perp_i, \quad i \rightarrow \perp_i$$

$$7: x \rightarrow [5, 5], \quad y \rightarrow [7, 7], \quad i \rightarrow [-\infty, -1]$$

# Iterate 5

$$it^5 = it^4 \nabla F(it^4)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow \perp_i, y \rightarrow \perp_i, i \rightarrow \perp_i$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 6

$$it^6 = it^5 \nabla F(it^5)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 7: first compute $F(it^6)$

$$it^7 = it^6 \nabla F(it^6)$$

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, 8], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 7: then $it^6 \nabla F(it^6)$

we have:

$$3: x \rightarrow [5, 5], \quad y \rightarrow [7, 7], \quad i \rightarrow [-\infty, \infty]$$

$$\nabla$$

$$3: x \rightarrow [5, 5], \quad y \rightarrow [7, 8], \quad i \rightarrow [-\infty, \infty]$$

$$=$$

$$3: x \rightarrow [5, 5], \quad y \rightarrow [7, \infty], \quad i \rightarrow [-\infty, \infty]$$

# Iterate 7: final result

$$it^7 = it^6 \nabla F(it^6)$$

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, 7], i \rightarrow [-\infty, -1]$

# Iterate 8

$$it^8 = it^7 \nabla F(it^7)$$

```
foo (int i) {  
  
1: int x :=5;  
2: int y :=7;  
  
3: if (i ≥ 0) {  
4:   y := y + 1;  
5:   i := i - 1;  
6:   goto 3;  
7: }  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$   
3:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, \infty]$   
4:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [0, \infty]$   
5:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [0, \infty]$   
6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$   
7:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, -1]$



# Iterate 9

$$it^9 = it^8 \nabla F(it^8)$$

```
foo (int i) {  
  
  1: int x :=5;  
  2: int y :=7;  
  
  3: if (i ≥ 0) {  
  4:   y := y + 1;  
  5:   i := i - 1;  
  6:   goto 3;  
  }  
  7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, \infty], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, 8], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, -1]$

# Iterate 10

$$it^{10} = it^9 \nabla F(it^9)$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], y \rightarrow [-\infty, \infty], i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], y \rightarrow [8, \infty], i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], y \rightarrow [8, \infty], i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], y \rightarrow [7, \infty], i \rightarrow [-\infty, -1]$

# Iterate 11: a post fixed point is reached

$$it^{11} = it^{10} \nabla F(it^{10})$$

```
foo (int i) {  
  
1:  int x :=5;  
2:  int y :=7;  
  
3:  if (i ≥ 0) {  
4:    y := y + 1;  
5:    i := i - 1;  
6:    goto 3;  
    }  
7:  
}
```

1:  $x \rightarrow [-\infty, \infty], \quad y \rightarrow [-\infty, \infty], \quad i \rightarrow [-\infty, \infty]$

2:  $x \rightarrow [5, 5], \quad y \rightarrow [-\infty, \infty], \quad i \rightarrow [-\infty, \infty]$

3:  $x \rightarrow [5, 5], \quad y \rightarrow [7, \infty], \quad i \rightarrow [-\infty, \infty]$

4:  $x \rightarrow [5, 5], \quad y \rightarrow [7, \infty], \quad i \rightarrow [0, \infty]$

5:  $x \rightarrow [5, 5], \quad y \rightarrow [8, \infty], \quad i \rightarrow [0, \infty]$

6:  $x \rightarrow [5, 5], \quad y \rightarrow [8, \infty], \quad i \rightarrow [-1, \infty]$

7:  $x \rightarrow [5, 5], \quad y \rightarrow [7, \infty], \quad i \rightarrow [-\infty, -1]$

# Chaotic (Asynchronous) Iteration

```
 $x_1 := \perp; x_2 := \perp; \dots; x_n := \perp;$   
 $W := \{1, \dots, n\};$ 
```

```
while ( $W \neq \{\}$ ) do {  
   $\ell := \text{removeLabel}(W);$   
   $\text{prev}_\ell := x_\ell;$   
   $x_\ell := \text{prev}_\ell \nabla f_\ell(x_1, \dots, x_n);$   
  if ( $x_\ell \neq \text{prev}_\ell$ )  
     $W := W \cup \text{influence}(\ell);$   
}
```

- $W$  is the worklist, a set of labels left to be processed
- $\text{influence}(\ell)$  returns the set of labels where the value at those labels is influenced by the result at  $\ell$
- Re-compute only when necessary, thanks to  $\text{influence}(\ell)$
- Asynchronous computation can be parallelized

# Summary: Connecting Math and Analysis

Mathematical Concept	Use in Program Analysis
Complete Lattice	Defines Abstract Domain and ensure joins exist.
Joins ( $\sqcup$ )	Combines facts arriving at a program point
Bottom ( $\perp$ )	Used for initialization of all but initial elements
Top ( $\top$ )	Used for initialization of initial elements
Widening ( $\nabla$ )	Used to guarantee analysis termination
Function Approximation	Critical to make sure abstract semantics approximate the concrete semantics
Fixed Points	This is what is computed by the analysis
Tarski's Theorem	Ensures fixed points exist.