CS 477: Dataflow Analysis and Abstract Interpretation

Sasa Misailovic

Based on previous slides by Martin Vechev

University of Illinois at Urbana-Champaign

The Art of Sound* Approximation: Static Program Analysis

- Define a function $F^\sharp$ such that $F^\sharp$ approximates F. This is typically done manually and can be tricky but is done once for a particular programming language.

- Then, use existing theorems which state that the least fixed point of $F^\sharp$, e.g. denote it V, approximates the least fixed point of F, e.g. denote it $[\![P]\!]$

- Finally, automatically compute a fixed point of $F^\sharp$, that is a V where $F^\sharp(V) = V$

* For a reminder and discussion about soundness and precision, see these articles:
http://www.pl-enthusiast.net/2017/10/23/what-is-soundness-in-static-analysis/
https://cacm.acm.org/blogs/blog-cacm/236068-soundness-and-completeness-with-precision/fulltext

Abstract Interpretation: step-by-step

1. Select/define an abstract domain
    - selected based on the type of **properties** you want to prove

2. Define abstract semantics **for the language** w.r.t. to the domain
    - prove **sound** w.r.t **concrete semantics**
    - involves defining abstract transformers
        - that is, effect of statement / expression on the abstract domain

3. Iterate abstract transformers over the abstract domain
    - until we reach a **fixed point**

The **fixed point** is the **over-approximation** of the program

FUNCTION APPROXIMATION

Approximating a Function

Given functions:

$$F : C \to C$$

$$F^\sharp : C \to C$$

what does it mean for $F^\sharp$ to approximate F
(for the purpose of "classical" program analysis) ?

$$\forall x \in C : F(x) \sqsubseteq_c F^\sharp(x)$$

Approximating a Function

What about when:

$$F: C \rightarrow C$$

$$F^{\sharp}: A \rightarrow A$$

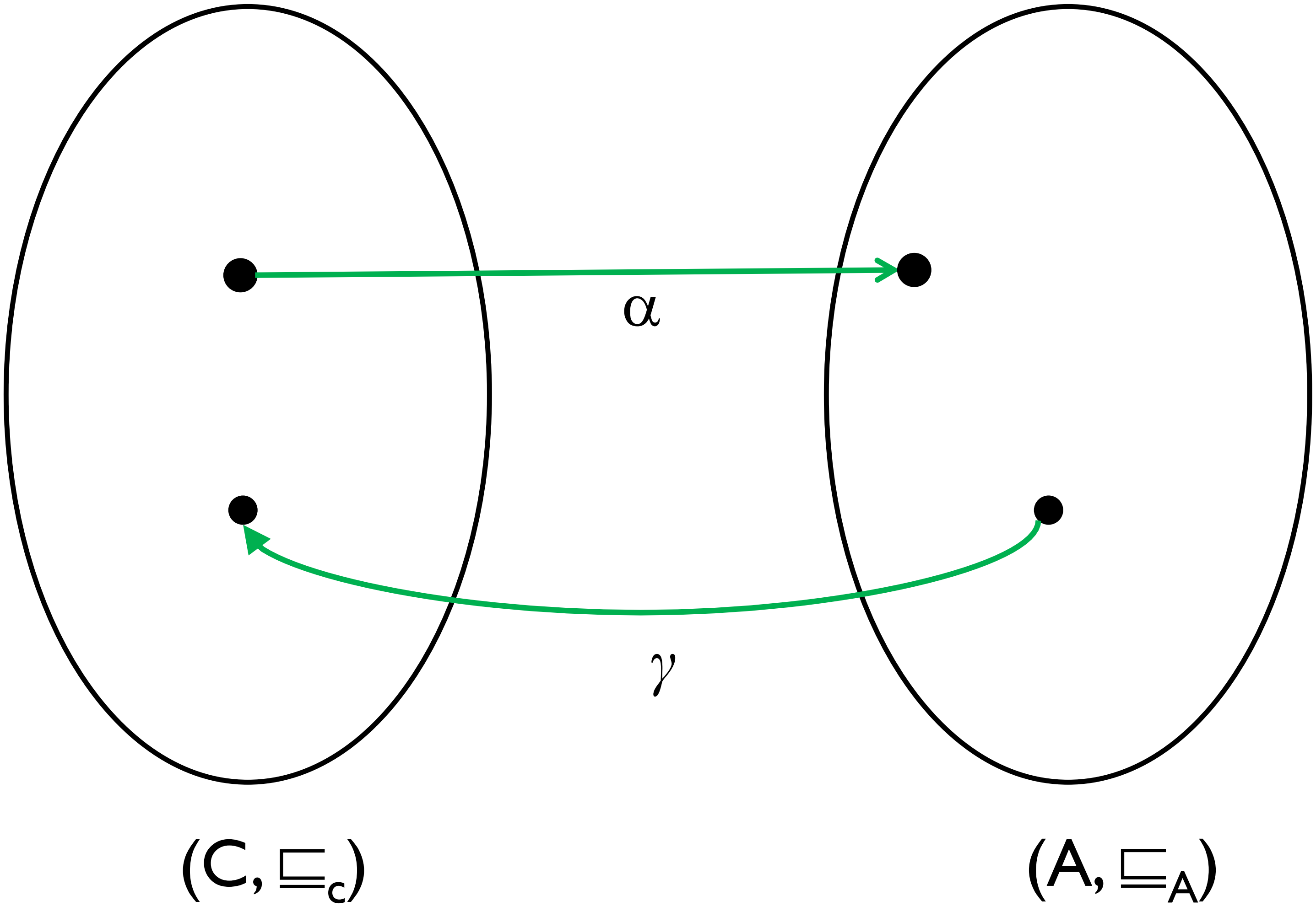
We need to connect the concrete C and the abstract A

We will connect them via two functions α and γ

$\alpha : C \rightarrow A$ is called the **abstraction** function

$\gamma : A \rightarrow C$ is called the **concretization** function

Connecting Concrete with Abstract



Approximating Function: Definition 1

So we have the 2 functions:

$$F: C \rightarrow C$$

$$F^{\sharp}: A \rightarrow A$$

If we know that α and γ form a **Galois Connection**, then we can use the following definition of approximation:

$$\forall z \in A : \alpha(F(\gamma(z))) \sqsubseteq_A F^{\sharp}(z)$$

Galois Connection

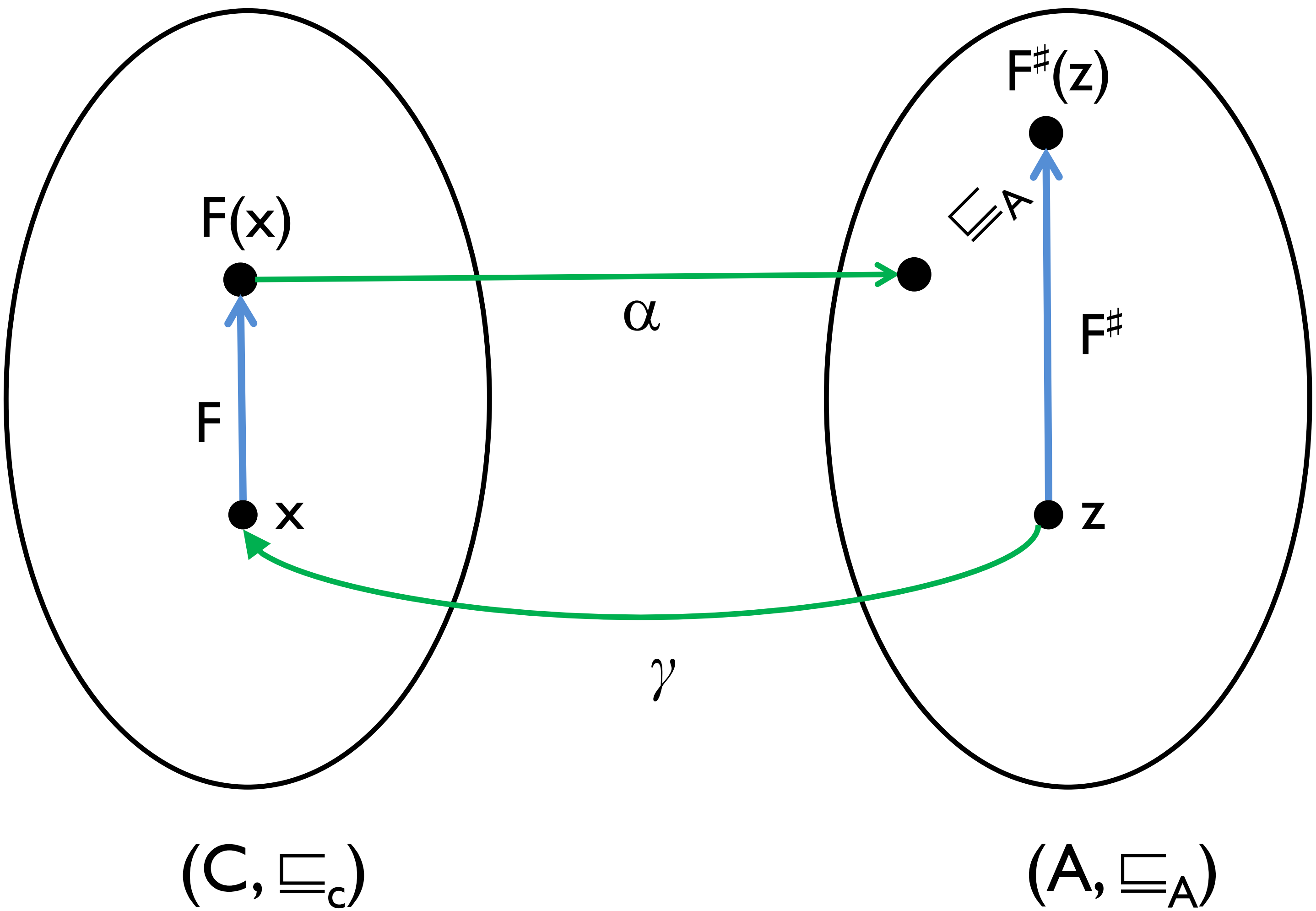
For the course, **it is not important** to know precisely what Galois Connections are.

The only point to keep in mind that is that they place some **restrictions** on α and γ.

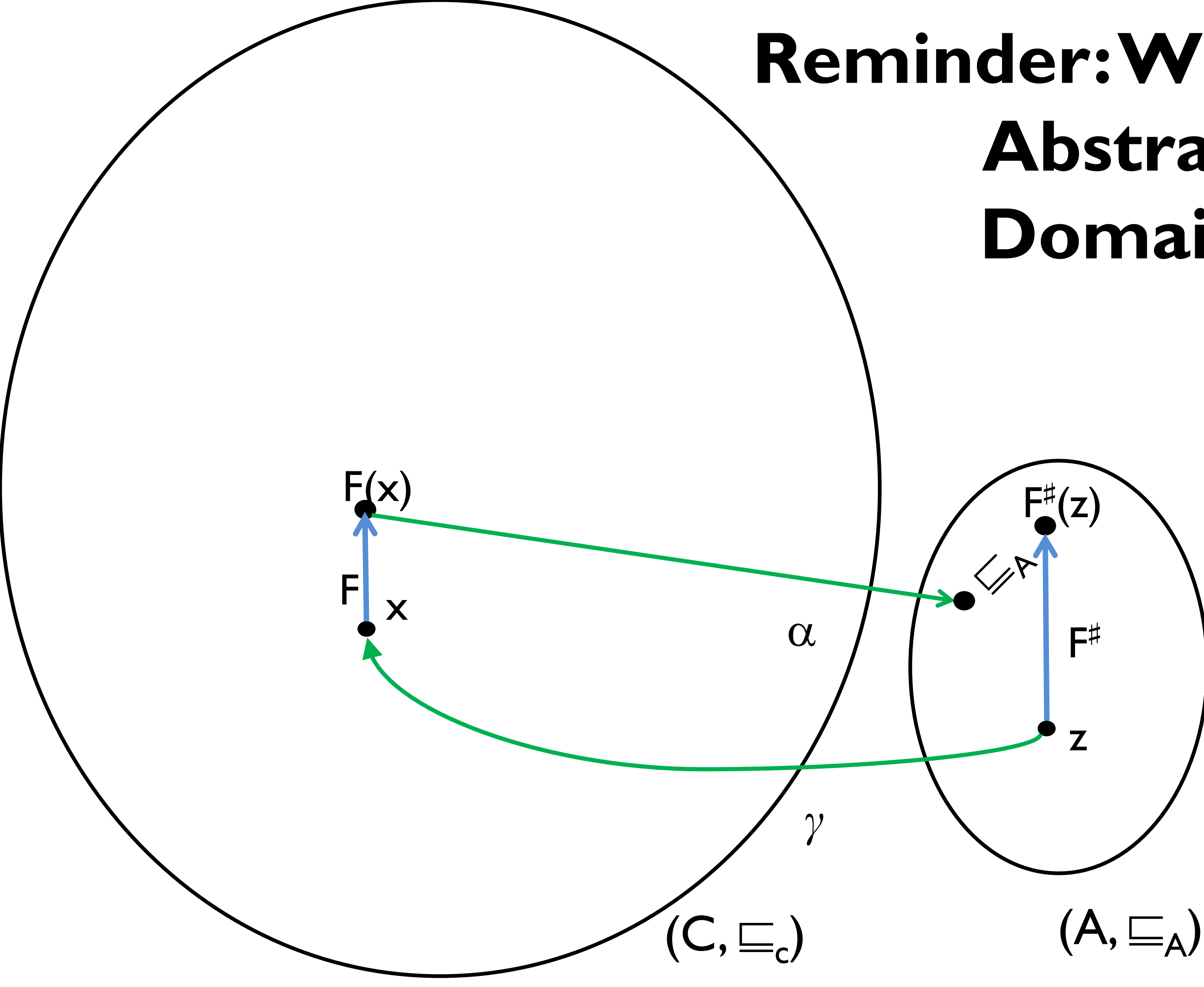
- For instance, they **require α to be monotone**.

Visualizing Definition 1

$$\forall z \in A : \alpha(F(\gamma(z))) \sqsubseteq_A F^\#(z)$$



Reminder: Why Abstract Domain?



Approximating a Function

This equation

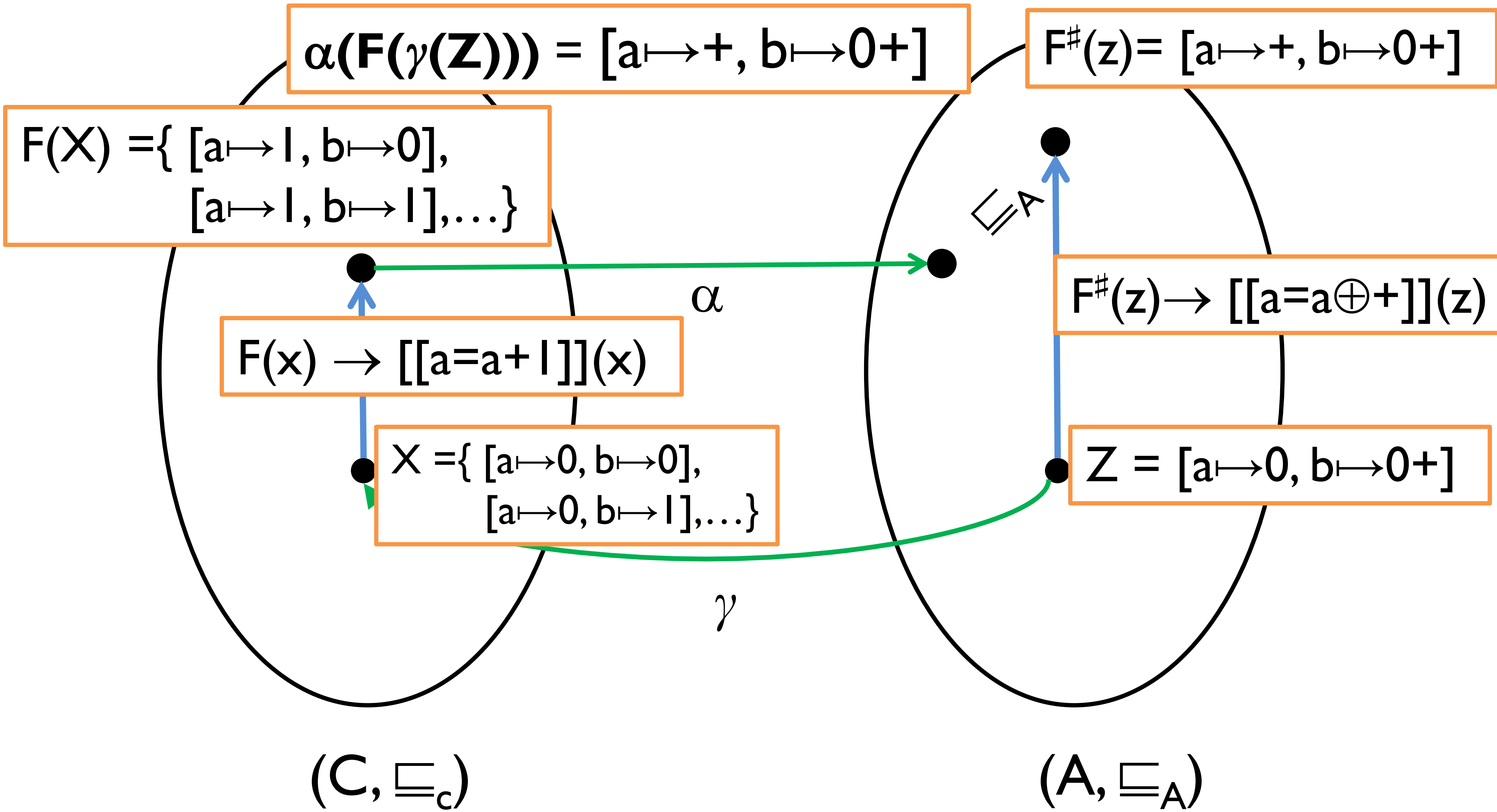
$$\forall z \in A: \ \alpha(\ F(\ \gamma(z)\ )\ )\ \sqsubseteq_A F^\sharp(z)$$

says that

- if we have some function in the abstract domain that we think **should approximate** the concrete function,
- then to check that this is indeed true, we need to prove
- that for any abstract element, (1) concretizing it, (2) applying the concrete function and (3) abstracting back again is **less than** applying the function in the abstract directly.

Visualizing Definition 1

$$\forall z \in \mathbf{A} : \alpha(\mathbf{F}(\gamma(\mathbf{z}))) \sqsubseteq_{\mathbf{A}} \mathbf{F}^{\#}(\mathbf{z})$$



Least precise approximation

To approximate F, we can always define $F^\sharp(z) = \top$

This solution is always **sound** as: $\forall z \in A : \alpha(F(\gamma(z))) \sqsubseteq_A \top$

However, it is not practically useful as it is **too imprecise**

Most precise approximation

$F^\sharp(z) = \alpha(F(\gamma(z)))$ is the **best abstract function.**

But, we often **cannot implement** best $F^\sharp(z)$ algorithmically.

However, we can come up with a $F^\sharp(z)$ that has the **same behavior** as $\alpha(F(\gamma(z)))$ but a **different implementation**.

Any such $F^\sharp(z)$ is referred to as the **best transformer**.

Key Theorem I: Least Fixed Point Approximation

If we have:

1. **monotonic** functions $F: C \rightarrow C$ and $F^{\sharp}: A \rightarrow A$
2. $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ forming a Galois Connection
3. $\forall z \in A : \alpha(F(\gamma(z))) \sqsubseteq_A F^{\sharp}(z)$ (that is, $F^{\sharp}$ approximates F)

then:

$$\alpha\ (\mathrm{lfp}(F)) \sqsubseteq_A \mathrm{lfp}\ (F^{\sharp})$$

This is important as it goes from **local** function approximation to **global** approximation. This is a key theorem in program analysis.

Least Fixed Point Approximation

The 3 premises to the theorem are usually proved **manually**.

Once proved, we can now **automatically** compute a least fixed point in the abstract and be sure that our result is **sound** !

So what is $F^\sharp$ then ?

$F^\sharp$ is to be defined for the particular abstract domain **A** that we work with. The domain **A** can be Sign, Parity, Interval, Octagon, Polyhedra, and so on.

In our setting and commonly, we simply keep a map from every label (program counter) in the program to an abstract element in **A**

Then $F^\sharp$ simply updates the mapping from labels to abstract elements.

$F^\sharp$

$$F^\sharp : (\mathtt{Lab} \longrightarrow \mathbb{A}) \longrightarrow (\mathtt{Lab} \longrightarrow \mathbb{A})$$

$$F^\sharp(m)\ell = \begin{cases} \top & \text{if } \ell \text{ is initial label} \\ \bigsqcup_{(\ell', \text{action}, \ell)} [\![\text{action}]\!](m(\ell')) & \text{otherwise} \end{cases}$$

$$[\![\text{action}]\!] : \mathbb{A} \longrightarrow \mathbb{A}$$

$[\![\text{action}]\!]$ is the key ingredient here. It captures the effect of a language statement on the abstract domain $\mathbb{A}$. Once we define it, we have $F^\sharp$

$[\![\text{action}]\!]$ is often referred to as the **abstract transformer** (cf. transfer function in dataflow analyses).

What is (ℓ',action,ℓ) ?

```
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 action ?

An **action** can be:

- $b \in$ `BExp` — boolean expression in a conditional
- x:= a — here, $a \in$ `AExp`
- skip

In performing an action, the assignment and the boolean expression of a conditional is **fully evaluated**

$$\{x \mapsto 2,\ y \mapsto 0\} \xrightarrow{\texttt{x:=y+x}} \{x \mapsto 4,\ y \mapsto 0\} \qquad \{x \mapsto 2,\ y \mapsto 0\} \xrightarrow{\texttt{if (x > 5) ...}}$$

Defining $[\![\text{action}]\!]$

$[\![\text{action}]\!]$ captures the abstract semantics of the language for a particular abstract domain.

We will see precise definitions for some actions in the Interval domain. Defining $[\![\text{action}]\!]$ for complex domains such as Octagon can be quite tricky.

Cheat Sheet: Connecting Math and Analysis

Mathematical Concept	Use in Static 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 (T)	Used for initialization of initial elements
Widening ($\triangledown$)	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.

Checkpoint

So far, we have seen 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 build static analyzers.

Domain of Program States

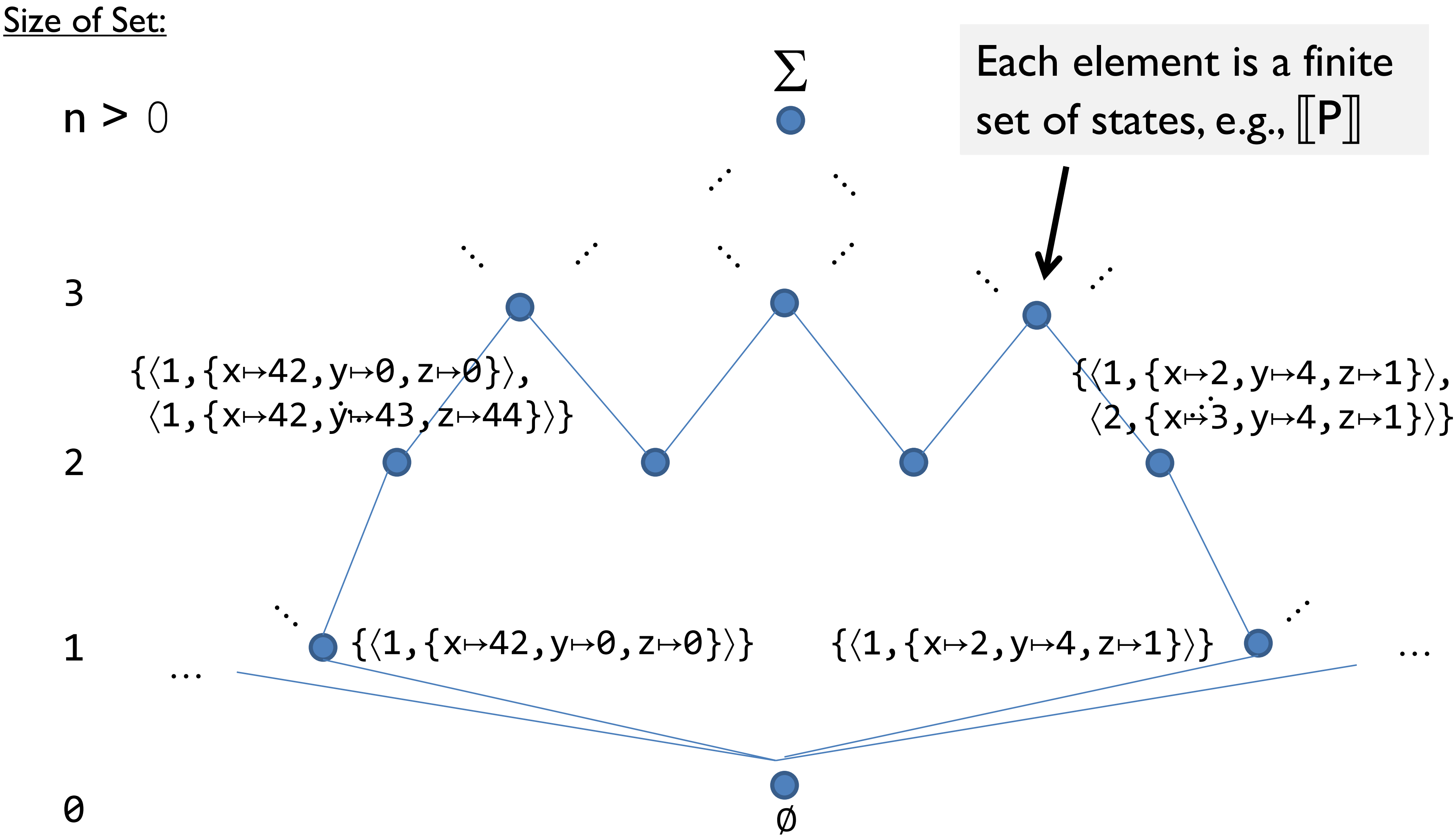
Our starting point is a domain where each element of the domain is a **set of states**. The domain of states is a **complete lattice**:

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

$$\Sigma = \text{Label} \times \text{Store}$$

Domain of Program States

Each element is a finite set of states, e.g., ⟦P⟧



Representing $[\![P]\!]$

Let $[\![P]\!]$ be the **set of reachable states** of a program P. ***(we discussed this in the Operational semantics lecture)***

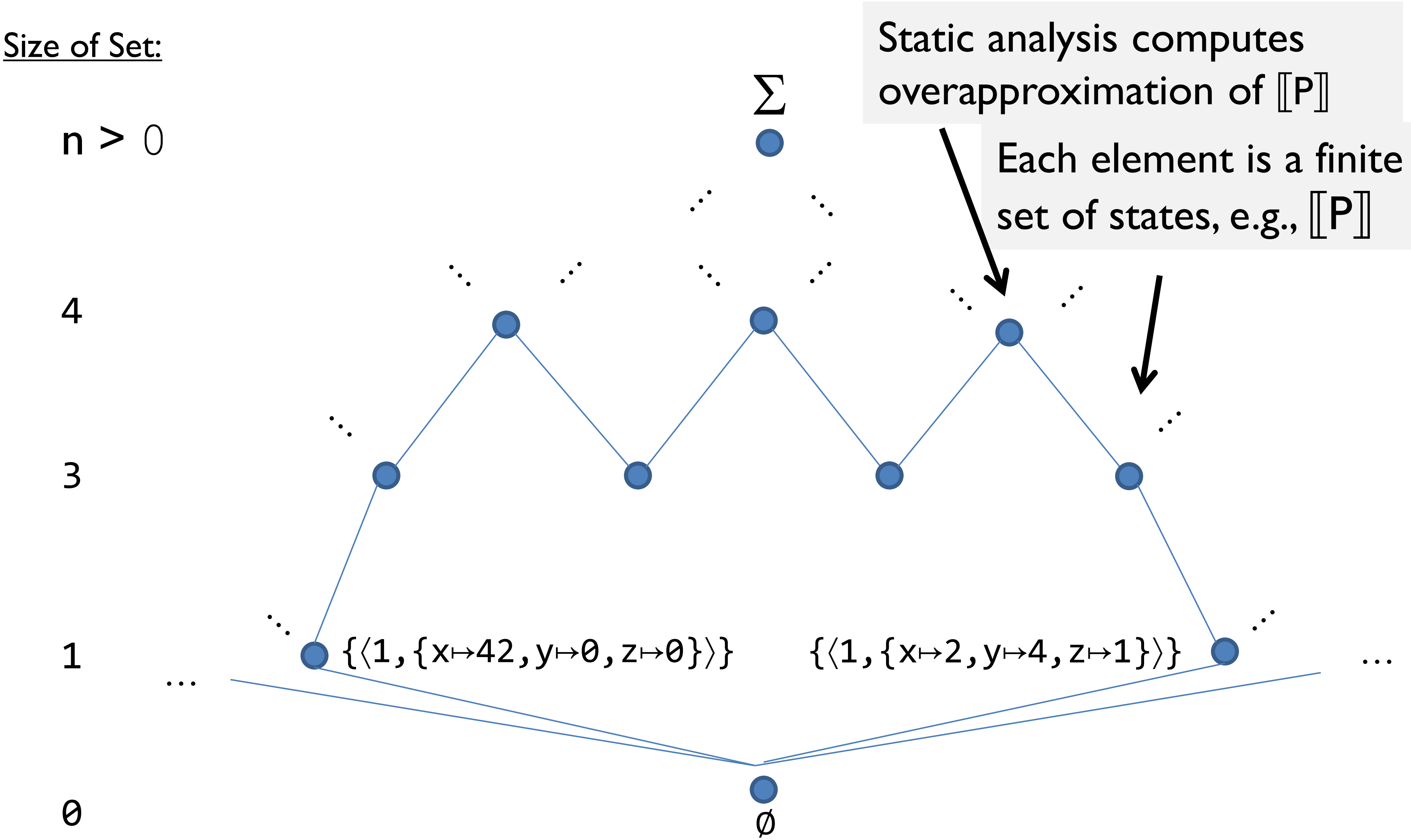
Def. Let function F be (where I is an initial set of states):

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

Then, $[\![P]\!]$ is a **fixed point** of F: i.e., $F([\![P]\!]) = [\![P]\!]$

(in fact, $[\![P]\!]$ is the least fixed point of F)

Starting Point: Domain of States



Abstract Interpretation: step-by-step

1. select/define an abstract domain
   - selected based on the type of **properties** you want to prove

2. define abstract semantics **for the language** w.r.t. to the domain
   - prove **sound** w.r.t **concrete semantics**
   - involves defining abstract transformers
     - that is, effect of statement / expression on the abstract domain

3. iterate abstract transformers over the abstract domain
   - until we reach a **fixed point**

The **fixed point** is the **over-approximation** of the program

Abstract Interpretation: Step 1

1. select/define an abstract domain
   - selected based on the type of **properties** you want to prove

Interval Domain

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.

Reminder: Interval Arithmetics

Intervals: X = [Xmin, Xmax] and Y = [Ymin, Ymax]

Operations:

X + Y = [Xmin + Ymin, Xmax + Ymax]

X * Y = [min (Xmin * Ymin, Xmin* Ymax, Xmax * Ymin, Xmax*Ymax),
max (Xmin * Ymin, Xmin* Ymax, Xmax * Ymin, Xmax*Ymax)]

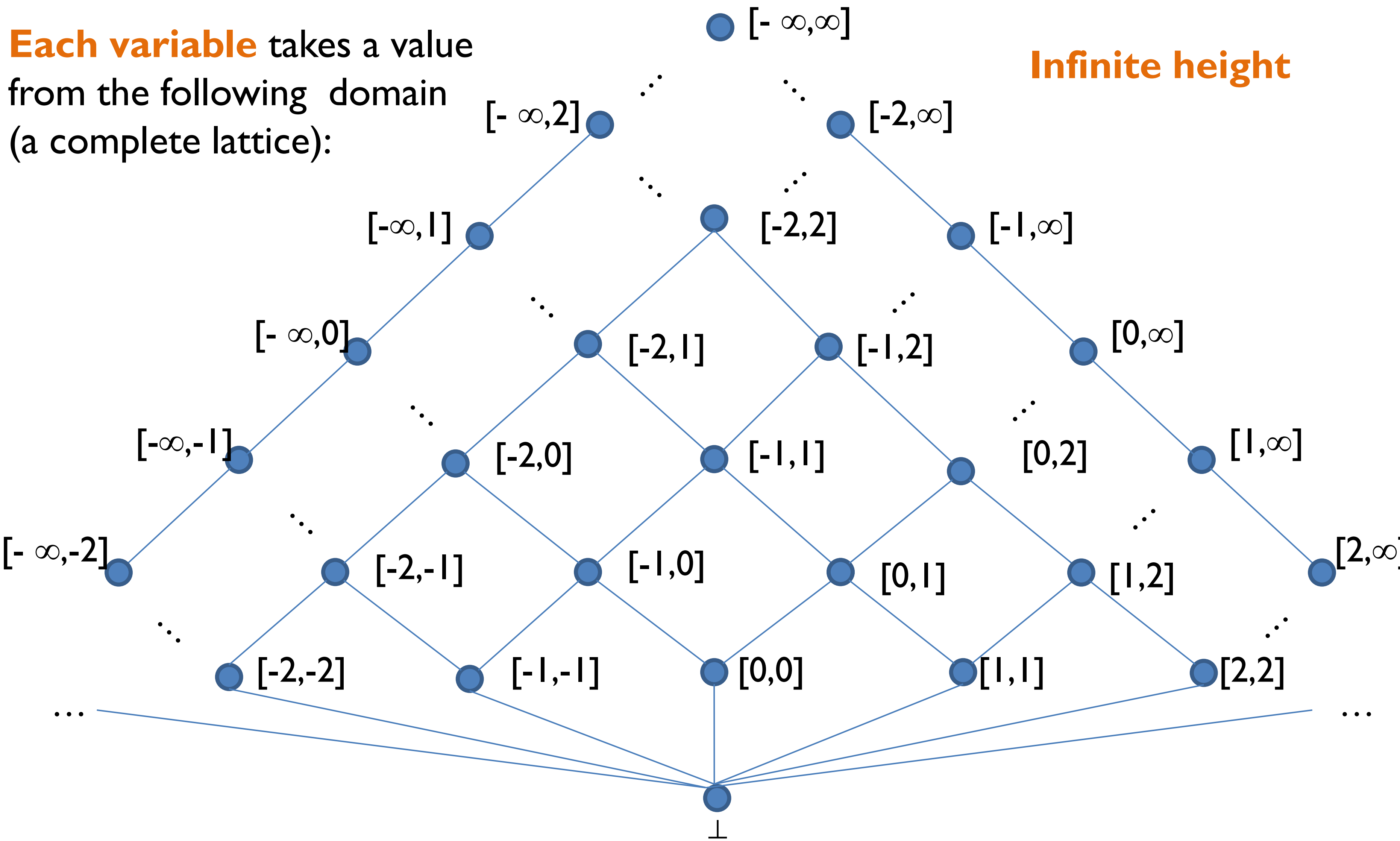
The definition of subtraction can then be X + [-1, -1] * Y

The definition for division X / Y is similar to multiplication, but defined properly only when 0 is not in the range Y.

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 **on integers** be a lattice: $(L^i, \sqsubseteq_i, \sqcup_i, \sqcap_i, \bot_i, [-\infty,\infty])$

We denote $\mathbf{Z}^\infty = Z \cup \{-\infty,\infty\}$
The set $\mathbf{L}^i = \{[x,y] \mid x,y \in Z^\infty, x \leq y\} \cup \{\bot_i\}$

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

Operations ($\sqsubseteq_i$, $\sqcup_i$, $\sqcap_i$):

- $[a,b] \sqsubseteq_i [c,d]$ 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 $\text{meet}(x,y)$ returns $[x,y]$ if $x \leq y$ and $\bot_i$ otherwise

Intervals: Applied to Programs

The $\mathtt{L^i}$ 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** $\mathtt{Lab} \rightarrow (\mathtt{Var} \rightarrow \mathtt{L^i})$

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 $\mathtt{L^i}$ $\sqsubseteq_i, \sqcup_i, \sqcap_i$ are **lifted directly** to both domains:

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

Intervals: Applied to Programs

$$\alpha^{\mathtt{i}}: \quad \wp(\Sigma) \longrightarrow (\mathtt{Lab} \longrightarrow (\mathtt{Var} \longrightarrow \mathtt{L}^{\mathtt{i}}))$$
$$\gamma^{\mathtt{i}}: \quad (\mathtt{Lab} \longrightarrow (\mathtt{Var} \longrightarrow \mathtt{L}^{\mathtt{i}})) \longrightarrow \wp(\Sigma)$$

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

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

Example of Abstraction and Concretization

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

$$\gamma^{i}(1 \longrightarrow (x \mapsto [1,8],\ y \mapsto [9,9],\ q \mapsto [-2,4])$$

$$= \{\ \langle 1,\{x\mapsto 1,y\mapsto 9,q\mapsto -2\}\rangle,\ \langle 1,\{x\mapsto 5,y\mapsto 9,q\mapsto -2\}\rangle,\ \langle 1,\{x\mapsto 8,y\mapsto 9,q\mapsto -2\}\rangle,$$
$$\langle 1,\{x\mapsto 1,y\mapsto 9,q\mapsto 4\}\rangle,\ \langle 1,\{x\mapsto 5,y\mapsto 9,q\mapsto 4\}\rangle,\ \langle 1,\{x\mapsto 8,y\mapsto 9,q\mapsto 4\}\rangle,$$
$$\langle 1,\{x\mapsto 7,y\mapsto 9,q\mapsto 3\}\rangle,\ \langle 1,\{x\mapsto 3,y\mapsto 9,q\mapsto 4\}\rangle,\ \langle 1,\{x\mapsto 1,y\mapsto 9,q\mapsto -1\}\rangle,$$
$$\ldots,\ \ldots,\ \ldots\ \}$$

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

Abstract Interpretation: Step 2

1. select/define an abstract domain
   - selected based on the type of **properties** you want to prove

2. define abstract semantics **for the language** w.r.t. to the domain
   - prove **sound** w.r.t **concrete semantics**
   - involves defining abstract transformers
     - that is, effect of statement / expression on the abstract domain

we still need to actually compute $\alpha^i(⟦P⟧)$

(or an over-approximation of it)

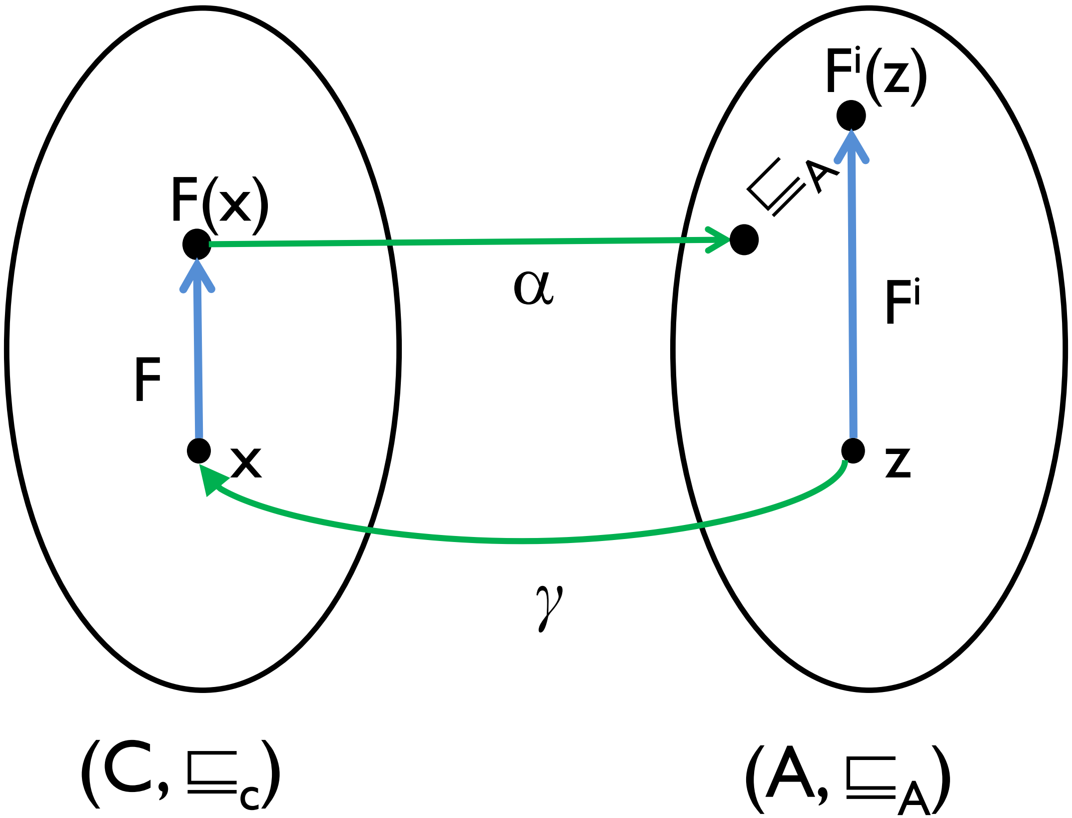
We need to approximate F

We want a function F^i where:

$F^i : (\texttt{Lab} \to (\texttt{Var} \to \texttt{L}^i)) \to (\texttt{Lab} \to (\texttt{Var} \to \texttt{L}^i))$

such that:

$\boldsymbol{\alpha^i\ (\mathbf{lfp}\ F) \sqsubseteq\ \mathbf{lfp}\ F^i}$



Lets define F^i

$$F^i : (\mathtt{Label} \rightarrow (\mathtt{Var} \rightarrow \mathtt{L}^i)) \rightarrow (\mathtt{Label} \rightarrow (\mathtt{Var} \rightarrow \mathtt{L}^i))$$

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

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

$$[\![action]\!]_i : (\mathtt{Var} \rightarrow \mathtt{L}^i) \rightarrow (\mathtt{Var} \rightarrow \mathtt{L}^i)$$

What is (ℓ',action,ℓ) ?

```
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 P, 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 ∈ BExp` — boolean expression in a conditional
- x:= a — here, `a ∈ 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 $[\![\text{action}]\!]_i$ produces sound and precise results.

F^i on an example

$F^i : (\texttt{Label} \to (\texttt{Var} \to L^i)) \to (\texttt{Label} \to (\texttt{Var} \to L^i))$

```
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:
}
```

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

$F^i(m)\ 2 = [\![x := 5]\!]_i\ (m(1))$

$F^i(m)\ 3 = [\![y := 7]\!]_i\ (m(2)) \sqcup [\![\text{goto } 3]\!]_i\ (m(6))$

$F^i(m)\ 4 = [\![i \geq 0]\!]_i\ (m(3))$

$F^i(m)\ 5 = [\![y := y + 1]\!]_i\ (m(4))$

$F^i(m)\ 6 = [\![i := i - 1]\!]_i\ (m(5))$

$F^i(m)\ 7 = [\![i < 0]\!]_i\ (m(3))$

$[\![x := a]\!]_i$

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

$\langle a , m\rangle \Downarrow_i v$ says that given a **map m**, the **expression a** evaluates to a value $v \in L^i$ (using interval arithmetic)

The operational semantics rules for expression evaluation:

- any constant Z is abstracted to an element in 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 ?

Look for all four combinations!

$[\![\mathbf{b}]\!]_i$

Let us first look at the expression: $a_1 \; c \; a_2$

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

For a memory map m, we need to define : $[\![a_1 \; c \; a_2]\!]_i(m)$ which produces another map as the result.

What is $[\![x \leq y]\!]$?

Easy case: $x_{max} \leq y_{min}$

- We simply keep the intervals of x and y

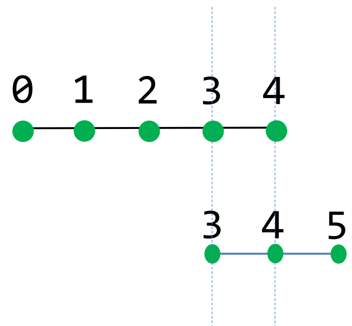
But, suppose we have the program:

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

What are the intervals for x and y at label 1 ?

Definition of $[l_1,u_1] \le [l_2,u_2]$

what should $[0,4] \le [3,5]$ produce at label 1 ?




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

one answer is: `(x=[0,3],y=[3,5])`. Is it sound ?

another non-comparable answer is: `(x=[0,4],y=[4,5])`. Is it sound ?

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

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

$[0,4] \leq [3,5] = (x=[0,4] \sqcap_i [-\infty,5], y=[0, \infty] \sqcap_i [3, 5])$

$= (x=[0,4], y=[3,5])$

Exercise: define < and =

Evaluating $[\![b]\!]_i$

$$[\![b_1 \vee b_2]\!]_i\ (m) = [\![b_1]\!]_i\ (m) \sqcup [\![b_2]\!]_i\ (m)$$

$$[\![b_1 \wedge b_2]\!]_i\ (m) = [\![b_1]\!]_i\ (m) \sqcap [\![b_2]\!]_i\ (m)$$

Abstract Interpretation: Step 3

1. select/define an abstract domain
   - selected based on the type of **properties** you want to prove

2. define abstract semantics **for the language** w.r.t. to the domain
   - prove **sound** w.r.t **concrete semantics**
   - involves defining abstract transformers
     - that is, effect of statement / expression on the abstract domain

3. iterate abstract transformers over the abstract domain
   - until we reach a **fixed point**

Chaotic (Asynchronous) Iteration

```
x₁ := ⊥; x₂ = ⊥; …; xₙ = ⊥;
W := {1,…,n};

while (W ≠ {}) do {
   ℓ := removeLabel(W);
   prev_ℓ := x_ℓ;
   x_ℓ := f_ℓ (x₁,…,xₙ) ;
   if (x_ℓ ≠ prev_ℓ)
      W := W ∪ influence(ℓ);
}
```

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

F^i on an example

F^i : (Label → (Var → L^i))→(Label → (Var → L^i))

```
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:
}
```

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

$F^i(m)\ 2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)\ 3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)\ 4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)\ 5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)\ 6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)\ 7 = [\![i < 0]\!]_i\,(m(3))$

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

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

$F^i(m)2 = [\![x := 5]\!]_i\ (m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\ (m(2)) \sqcup [\![goto\ 3]\!]_i\ (m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\ (m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\ (m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\ (m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\ (m(3))$

```
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: $x \rightarrow \bot_i,\ y \rightarrow \bot_i,\ i \rightarrow \bot_i$

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

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

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

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

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

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

Iterate 1

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

$F^i(m)2 = [\![x := 5]\!]_i\ (m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\ (m(2)) \sqcup [\![\text{goto } 3]\!]_i\ (m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\ (m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\ (m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\ (m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\ (m(3))$

```
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: $x \rightarrow \bot_i$, $y \rightarrow \bot_i$, $i \rightarrow \bot_i$

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

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

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

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

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

Iterate 2

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

```
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: $x \rightarrow \bot_i,\ y \rightarrow \bot_i,\ i \rightarrow \bot_i$

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

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

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

```
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: $x \to [-\infty,\infty]$, $y \to [-\infty,\infty]$, $i \to [-\infty,\infty]$

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

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

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

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

6: $x \to \bot_i$, $y \to \bot_i$, $i \to \bot_i$

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

Iterate 6

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

```
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: $x \to [-\infty,\infty]$, $y \to [-\infty,\infty]$, $i \to [-\infty,\infty]$

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

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

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

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

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

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

Iterate 8

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i (m(1))$

$F^i(m)3 = [\![y := 7]\!]_i (m(2)) \sqcup [\![\text{goto } 3]\!]_i (m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i (m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i (m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i (m(5))$

$F^i(m)7 = [\![i < 0]\!]_i (m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i (m(1))$

$F^i(m)3 = [\![y := 7]\!]_i (m(2)) \sqcup [\![\text{goto } 3]\!]_i (m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i (m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i (m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i (m(5))$

$F^i(m)7 = [\![i < 0]\!]_i (m(3))$

```
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: $x \to [-\infty,\infty]$, $y \to [-\infty,\infty]$, $i \to [-\infty,\infty]$

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

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

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

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

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

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

Iterate 14

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

$F^i(m)2 = [\![x := 5]\!]_i\ (m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\ (m(2)) \sqcup [\![\text{goto } 3]\!]_i\ (m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\ (m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\ (m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\ (m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\ (m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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

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

$F^i(m)2 = [\![x := 5]\!]_i\,(m(1))$

$F^i(m)3 = [\![y := 7]\!]_i\,(m(2)) \sqcup [\![\text{goto } 3]\!]_i\,(m(6))$

$F^i(m)4 = [\![i \geq 0]\!]_i\,(m(3))$

$F^i(m)5 = [\![y := y + 1]\!]_i\,(m(4))$

$F^i(m)6 = [\![i := i - 1]\!]_i\,(m(5))$

$F^i(m)7 = [\![i < 0]\!]_i\,(m(3))$

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

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: $F^{i\ (1)}$, $F^{i\ (2)}$, $F^{i\ (3)}$,…

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,
then when 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:

$$\text{lfp}(F) \sqsubseteq A$$

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 $x^0 \sqsubseteq x^1 \sqsubseteq x^2 \sqsubseteq \ldots \sqsubseteq x^n \sqsubseteq \ldots$ is an increasing sequence then $y^0 \sqsubseteq y^1 \sqsubseteq y^2 \sqsubseteq \ldots \sqsubseteq y^n$ stabilizes after a finite number of steps

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

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, ∇: $L \times L \rightarrow L$, F: $L \rightarrow L$ is monotone

Then the sequence:

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

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: $lfp(F) \sqsubseteq y^n$

Widening for Interval Domain

Let us define a widening operator for the intervals

$$\nabla_i \colon \mathtt{L^i} \times \mathtt{L^i} \rightarrow \mathtt{L^i}$$

[a, b] ∇_i [c, d] = [e, f] where:

if c < a, then e = -∞, else e = a

if d > b, then f = ∞, else f = b

if one of the operands is ⊥ 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

Let us define a widening operator for the intervals

$$\nabla_i : L^i \times L^i \rightarrow L^i$$

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

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

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

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

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

Examples: Widening for Interval

Let us define a widening operator for the intervals

$$\nabla_i: L^i \times L^i \rightarrow L^i$$

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

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

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

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

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

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

Iterate 0

$it^0 = \bot$

```
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: $x \rightarrow \bot_i$, $y \rightarrow \bot_i$, $i \rightarrow \bot_i$

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

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

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

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

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

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

Iterate 1

$$it^1 = it^0 \triangledown F(it^0)$$
$$= \bot \triangledown F(\bot)$$
$$= F(\bot)$$

```
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: $x \rightarrow \bot_i$, $y \rightarrow \bot_i$, $i \rightarrow \bot_i$

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

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

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

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

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

Iterate 2

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

```
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

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

```
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

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

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

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

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

```
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

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

```
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 $F(it^6)$

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

```
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:

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

$$\nabla$$

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

$$=$$

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

Iterate 7: final result

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

```
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

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

```
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

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

```
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

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

```
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: $x \to [-\infty,\infty]$, $y \to [-\infty,\infty]$, $i \to [-\infty,\infty]$

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

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

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

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

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

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

Chaotic (Asynchronous) Iteration

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

while $(W \neq \{\})$ **do** {
  $\ell :=$ removeLabel(W);
  $prev_\ell := x_\ell;$
  $x_\ell := f_\ell(x_1,\ldots,x_n)$;
  if $(x_\ell \neq prev_\ell)$
    $W := W \cup$ influence(ℓ);
}

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

Chaotic (Asynchronous) Iteration With Widening

```
x1 := ⊥;  x2 = ⊥; …; xn = ⊥;
W := {1,…,n};

while (W ≠ {}) do  {
   ℓ :=  removeLabel(W);
   prevℓ  :=  xℓ;
   xℓ := prevℓ ▽ fℓ (x1,…,xn) ;
   if (xℓ ≠  prevℓ)
     W :=  W  ∪  influence(ℓ);
}
```

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

HEAP ANALYSIS

Pointer Analysis

Pointer and Alias Analysis are fundamental to reasoning about heap manipulating programs (pretty much all programs today).

- **Pointer Analysis:**
  - What objects does each pointer points to?
  - Also called points-to analysis
- **Alias Analysis:**
  - Can two pointers point to the same location?
  - Client of pointer analysis

Points-to Pair: pair (r1, r2) denoting that one of the memory locations of r1 An ordered may hold the address of one of the memory locations of r2.

Example

Points-to pairs

X = 1

P = &X // (P, X)

Q = P // { (P, X), (Q, X) }

*P = 2

Points-to Pair: pair (r1, r2) denoting that one of the memory locations of r1 An ordered may hold the address of one of the memory locations of r2.

Example

Points-to pairs

```
X = 1
P = &X      // (P, X)
Q = P       // { (P, X), (Q, X) }
R = Q       // { (P, X), (Q, X), (R,X) }
```

"Short notation": vs the long one that would list all the aliases.

Challenges of Points-To Analysis

- **Pointers to pointers**, which can occur in many ways: take address of pointer; pointer to structure containing pointer; pass a pointer to a procedure by reference
- **Aggregate objects**: structures and arrays containing pointers
- **Recursive data structures** (lists, trees, graphs, etc.) closely related problem: anonymous heap locations
- **Control-flow:** analyzing different data paths
- **Interprocedural:** a location is often accessed from multiple functions; a common pattern (e.g., pass by reference)
- Compile-time cost
  - Number of variables, $|V|$, can be large
  - Number of alias pairs at a point can be $O(|V|^2)$

Naming Schemes for Heap Objects

The Naming Problem: Example 1

```
int main() {
   // Two distinct objects
   T* p = create(n);
   T* q = create(m);
}

T* create(int num) {
   // Many objects allocated here
   return new T(num);
}
```

Q. Should we try to distinguish the objects created in main()?

Naming Schemes for Heap Objects

The Naming Problem: Example 2

```
T* makelist(int len) {
  T* newObj = new T; // Many distinct objects
                     // allocated here
  newObj->next = (--len == 0)? NULL :
                               makelist(len);
}
```

Q. Can we distinguish the objects created in makelist()?

Possible Naming **Abstractions**

H_0 : One name for the entire heap

H_T : One name per type T (for type-safe languages)

H_L : One name per heap allocation site L (line number)

H_C : One name per (acyclic) call path C ("*cloning*")

H_F : One name per immediate caller F or call-site ("*one-level cloning")*

Program States for Points-To Analysis

```
// initially x = z = p = q = null
for  (i = 0; i < 2; i++) {
  // allocate O1, O2
  A:   x := newObject T1;
  if (i == 0)
     p := x;
  else
     z := x;
}
// allocate O3
B:   x := newObject  T1;
     z.f := x;
// allocate O4
C:   q := newObject  T1;
x := null;
```

Abstraction:

{ pointer $\rightarrow$ {Allocation Sites}, … }

e.g. { p $\rightarrow$ {A}, x $\rightarrow$ {A}, z $\rightarrow$ {A} }

Concretization:

{ pointer $\rightarrow$ {Objects allocated} … }

e.g., { p $\rightarrow$ {O_1, O_2},
x $\rightarrow$ {O_1, O_2},
z $\rightarrow$ {O_1, O_2} }

Program States for Points-To Analysis

```
// initially x = z = p = q = null
for  (i = 0; i < 2; i++) {
  // allocate O1, O2
  A:  x := newObject T1;
  if (i == 0)
     p := x;
  else
     z := x;
}
// allocate O3
B:  x := newObject  T1;
    z.f := x;
// allocate O4
C:  q := newObject  T1;
x := null;
```

The result of pointer analysis
<u>at the fixed point</u>:

$p \mapsto \varnothing, q \mapsto \varnothing, x \mapsto \varnothing, z \mapsto \varnothing$

$p \mapsto \{A\}, q \mapsto \varnothing, x \mapsto \{A\}, z \mapsto \{A\}$

$p \mapsto \{A\}, q \mapsto \varnothing, x \mapsto \{A\}, z \mapsto \{A\}$

$p \mapsto \{A\}, q \mapsto \varnothing, x \mapsto \{A\}, z \mapsto \{A\}$

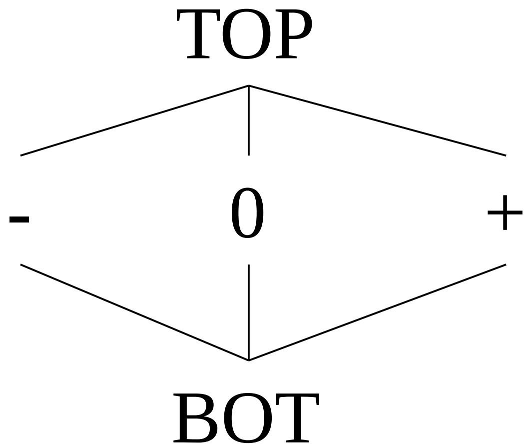
$p \mapsto \{A\}, q \mapsto \varnothing, x \mapsto \{B\}, z \mapsto \{A\}$

$p \mapsto \{A\}, q \mapsto \varnothing, x \mapsto \{B\}, z \mapsto \{A\}$, $A.f \mapsto \{B\}$

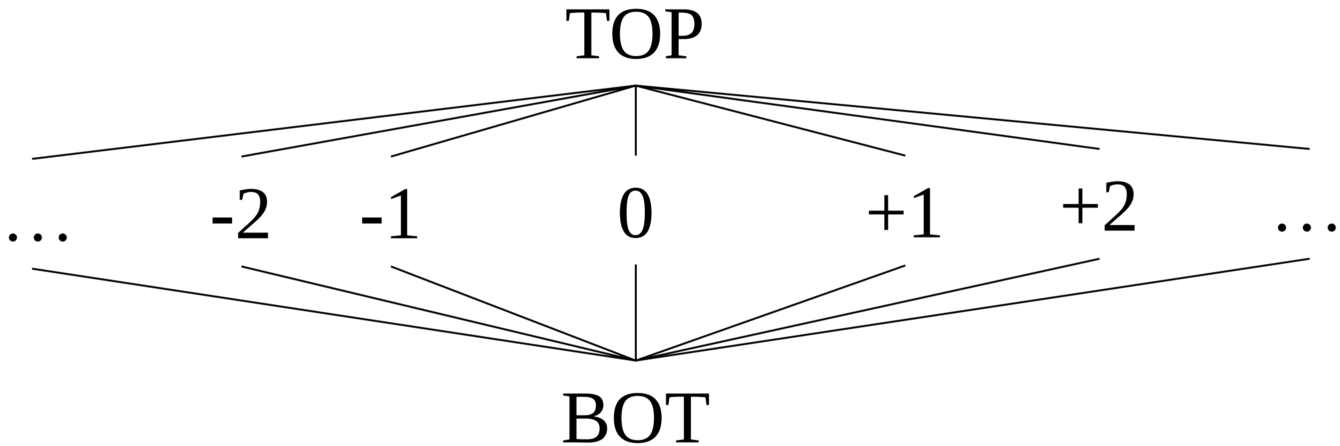
$p \mapsto \{A\}, q \mapsto \{C\}, x \mapsto \{\}, z \mapsto \{A\}$, $A.f \mapsto \{B\}$

RELATIONAL ABSTRACT DOMAINS

Sign Domain



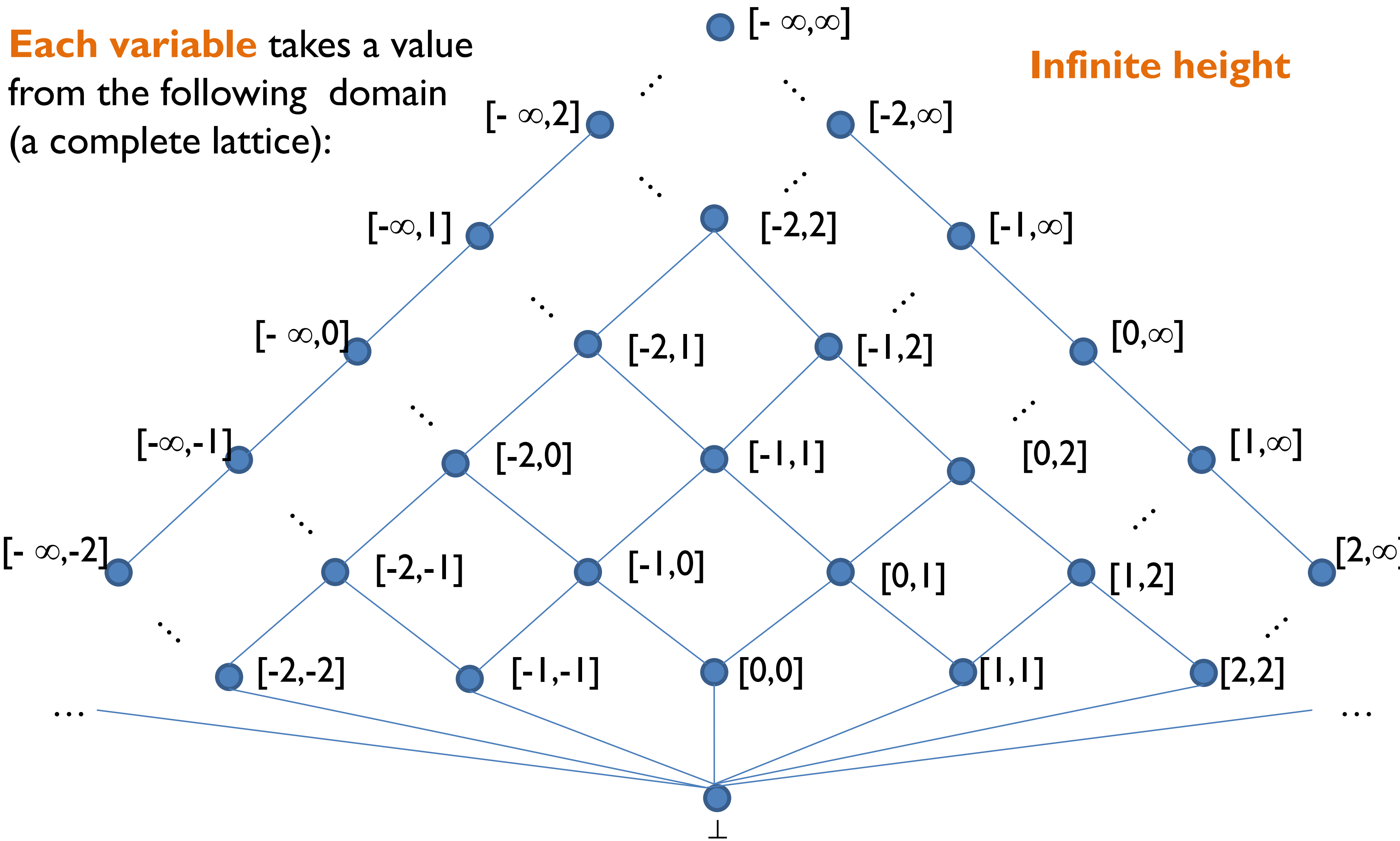
Constant Domain



Interval Domain

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

Infinite height

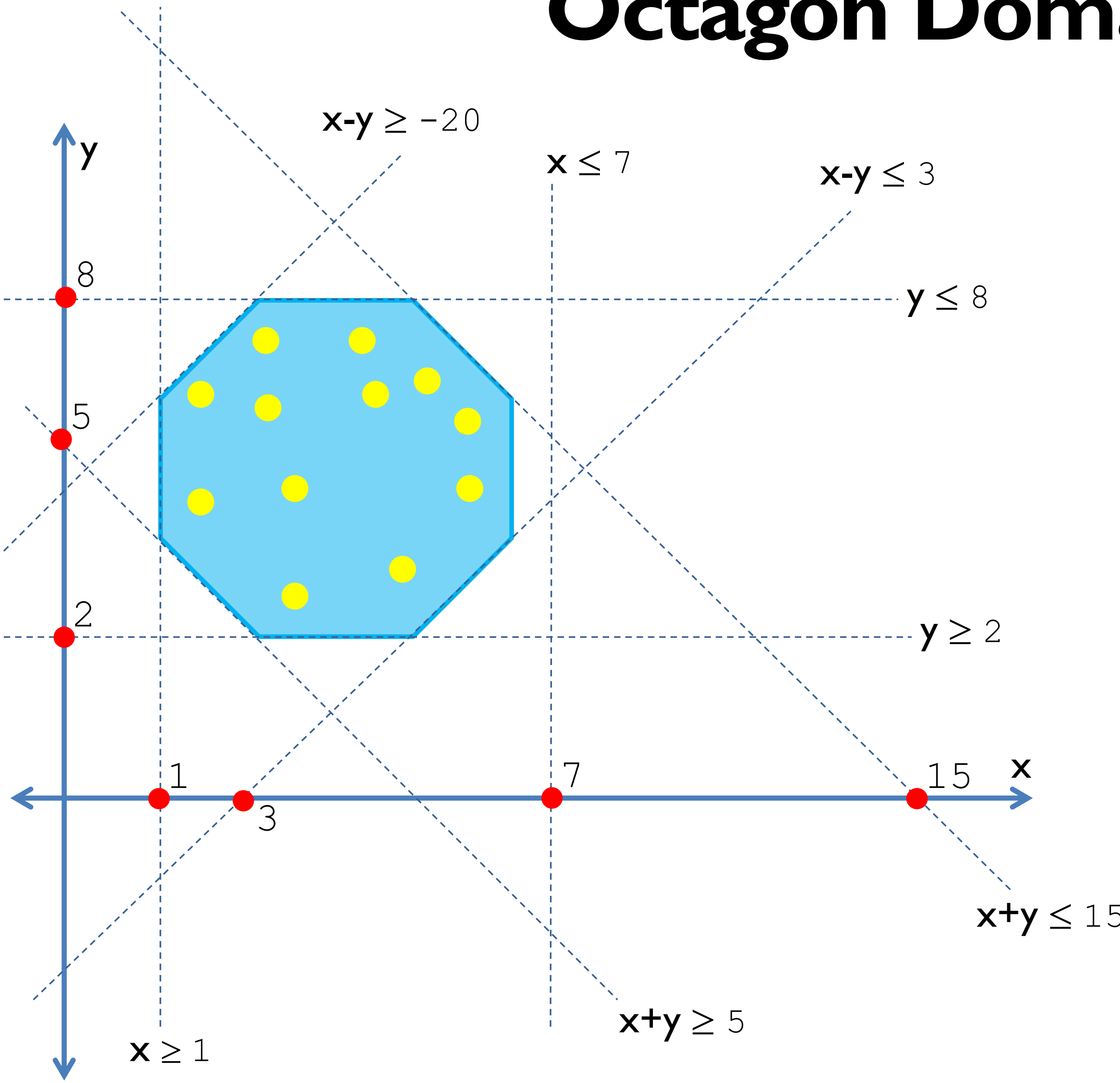


Relational Abstractions

The Interval domain is an example of a **non-relational** domain. It does not explicitly keep the relationship between variables.

Sometimes, it may be necessary to keep this relationship to be more precise. Octagon and Polyhedra domains keep the relationship. These domains are called **relational domains**.

Octagon Domain



constraints are of the following form:

$\pm x \pm y \leq c$

$\pm x \leq c$

$\pm y \leq c$

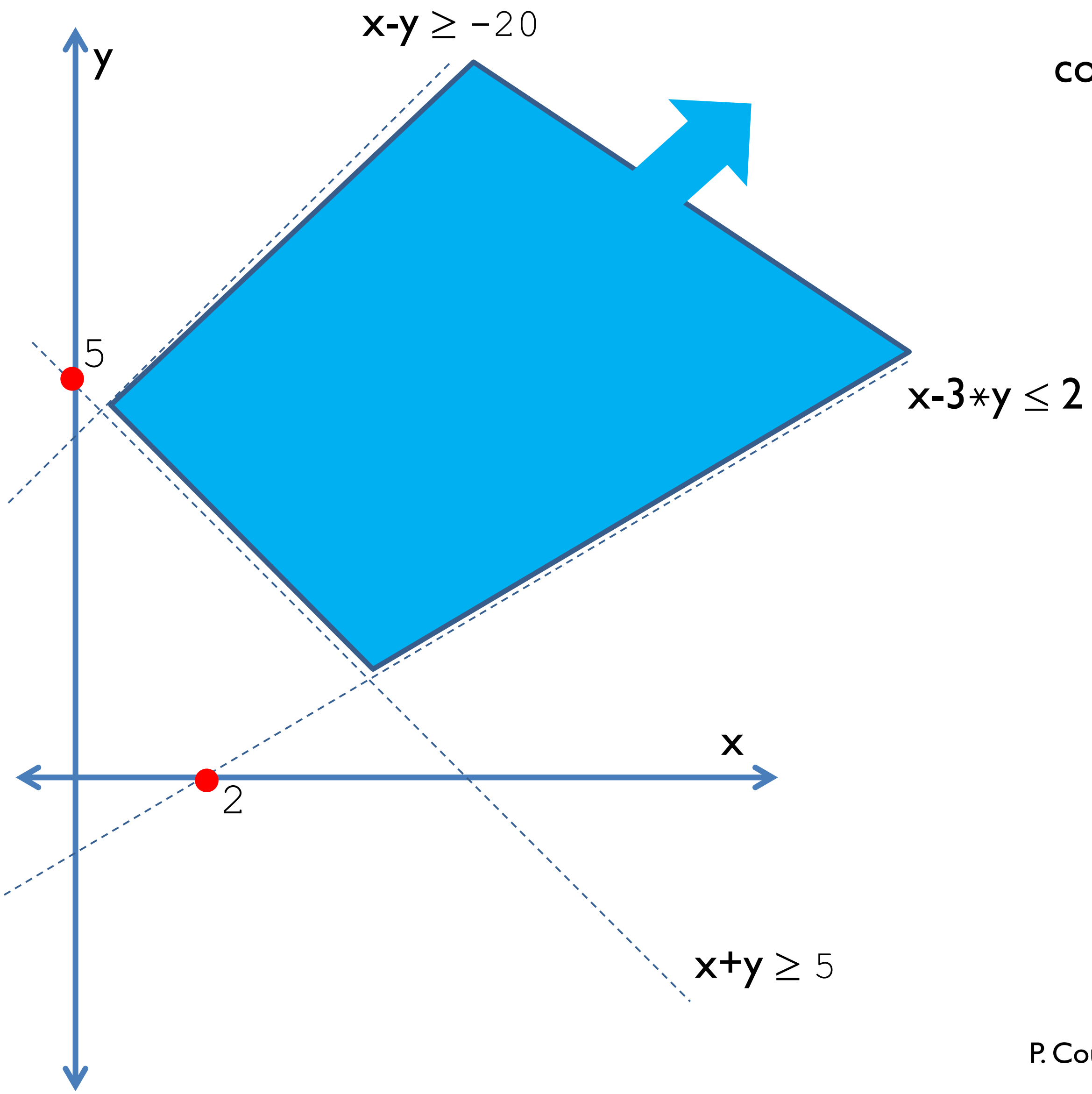
The slope is fixed

an abstract state is a map from labels to conjunction of constraints

$x - y \leq 3 \quad \wedge$
$y \leq 8 \quad \wedge$
$y \geq 2 \quad \wedge$
$x + y \leq 15 \quad \wedge$
$x + y \geq 5 \quad \wedge$
$x \geq 1 \quad \wedge$
$x - y \geq -20 \quad \wedge$
$x \leq 7$

A. Mine. The octagon abstract domain. In WCRE 2001

Polyhedra Domain



constraints are of the following form:

$$c_1 x_1 + c_2 x_2 \ldots + c_n x_n \leq c$$

the slope can vary

an abstract state is again a map from labels to conjunction of constraints:

$$x - y \geq -20 \land$$
$$x - 3 * y \leq 2 \land$$
$$x + y \geq 5$$

P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In POPL '78

Approximating a Function: Definition 2

We have the 2 functions:

$$F: C \rightarrow C$$
$$F^{\#}: A \rightarrow A$$

But what if α and γ **do not** form a **Galois Connection**?

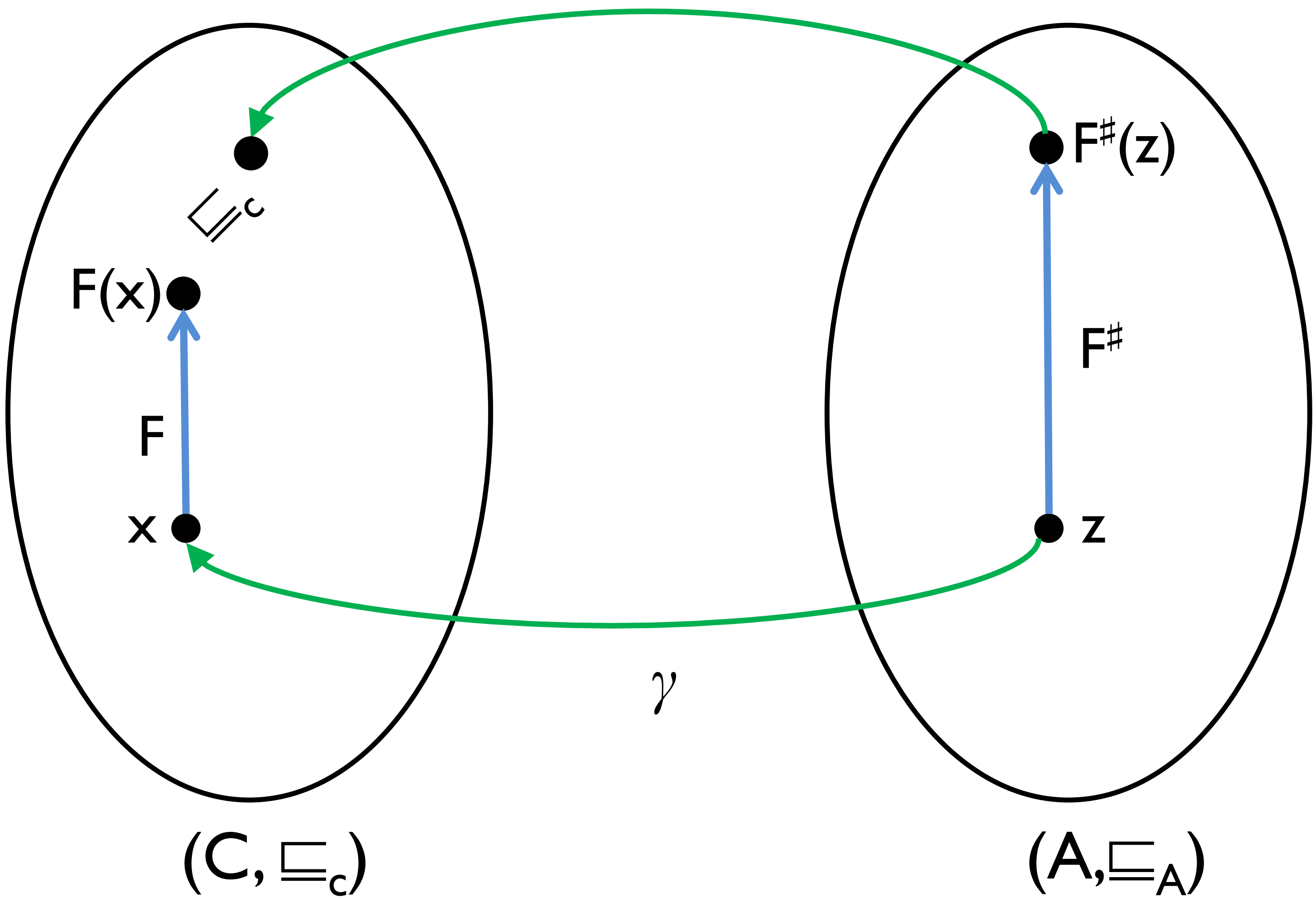
- For instance, α is not monotone.
- For instance, Polyhedral domain does not form GC.

Then, we can use the following definition of approximation:

$$\forall z \in A : F(\gamma(z)) \quad \sqsubseteq_c \quad \gamma(F^{\#}(z))$$

Visualizing Definition 2

$$\forall z \in A : F(\gamma(z)) \quad \sqsubseteq_c \quad \gamma(F^\sharp(z))$$



Key Theorem 2: Least Fixed Point Approximation

If we have:

1. **monotone** functions $F: C \rightarrow C$ and $F^{\sharp}: A \rightarrow A$
2. $\gamma: A \rightarrow C$ is monotone
3. $\forall z \in A: F(\gamma(z)) \sqsubseteq_c \gamma(F^{\sharp}(z))$ (that is, $F^{\sharp}$ approximates F)

then:

$$\mathrm{lfp}(F) \sqsubseteq_c \gamma(\mathrm{lfp}(F^{\sharp}))$$

This is important as it goes from **local** function approximation to **global** approximation. Another key theorem in program analysis.

Some Uses of Numerical Domais

- Out of bounds checks
- Division by zero
- Aliasing (A. Venet, SAS'02)
- Predicate abstraction
  (P. Cousot, Verification by abstract interpretation, 2003)
- Resource usage (J Navas et al. ICLP' 07).
- Machine Learning: Certifying Neural Networks (Singh et al POPL '19)

Additional Materials

List of classical papers and abstract domains:
http://www.di.ens.fr/~cousot/AI/

Tools and libraries for abstract interpretation

- Astree
- Fluctuat
- Frama-C

Libraries of abstract domains:

- Oct (octagon)
- NewPolka and Parma (polyhedral)
- Recent: Fast Polyhedra Abstract Domain (POPL'17)