CS 526

Advanced
Compiler
Construction

http://misailo.cs.Illinois.edu/courses/cs526

ABSTRACT INTERPRETATION

The slides adapted from Martin Vechev

So Far…

Dataflow Analysis

Lattice Theory & Abstract Domains

Abstraction, MOP, MFP Solutions

- What is the relationship between concrete program state and the analysis?
- How to build analyses in a rigorous manner?

MEANING OF DATAFLOW ANALYSIS (REVIEW)

Meaning of Dataflow Results

Concept of program state s for control-flow graphs

- Program point n where execution located (n is node that will execute next)
- Values of variables in program

Each execution generates a trajectory of states:

- $s_0;s_1;\ldots;s_k$, where each $s_i \in ST$
- s_{i+1} generated from s_i by executing basic block to
  - Update variable values
  - Obtain new program point n

Relating States to Analysis Result

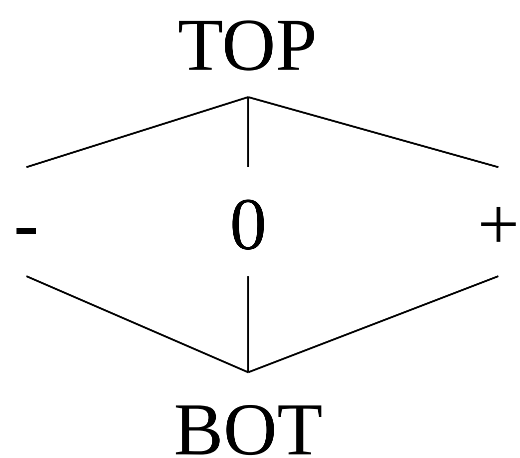
- Meaning of analysis results is given by an abstraction function $AF: ST \rightarrow P$

- Correctness condition: require that for all states s
  $$AF(s) \leq in_n$$
  where n is the next statement to execute in state s

Sign Analysis Example

Sign analysis - compute sign of each variable v

Base Lattice: P = flat lattice on {-,0,+}



Actual lattice records a value for each variable

- Example element: [a→+, b→0, c→-]

Interpretation of Lattice Values

If value of v in lattice is:

- BOT: no information about sign of v
- -: variable v is negative
- 0: variable v is 0
- +: variable v is positive
- TOP: v may be positive or negative

What is **abstraction function AF**?

- $AF([x_1,\ldots,x_n]) = [sign(x_1), \ldots, sign(x_n)]$
- Where $sign(x) = 0$ if $x = 0$, + if $x > 0$, - if $x < 0$

Operation ⊗ on Lattice

⊗	BOT	-	0	+	TOP
BOT	BOT	BOT	0	BOT	BOT
-	BOT	+	0	-	TOP
0	0	0	0	0	0
+	BOT	-	0	+	TOP
TOP	BOT	TOP	0	TOP	TOP

Transfer Functions

If n of the form v = c

- $f_n(x) = x[v \rightarrow +]$ if c is positive
- $f_n(x) = x[v \rightarrow 0]$ if c is 0
- $f_n(x) = x[v \rightarrow -]$ if c is negative

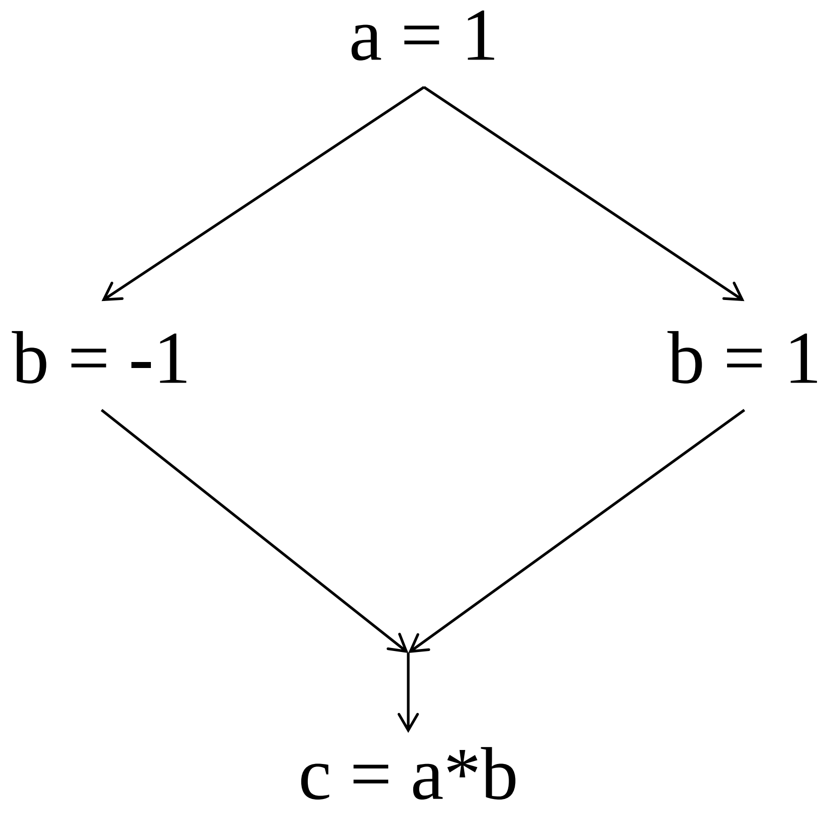
If n of the form $v_1 = v_2 * v_3$

- $f_n(x) = x[v_1 \rightarrow x[v_2] \otimes x[v_3]]$

I = TOP

(uninitialized variables may have any sign)

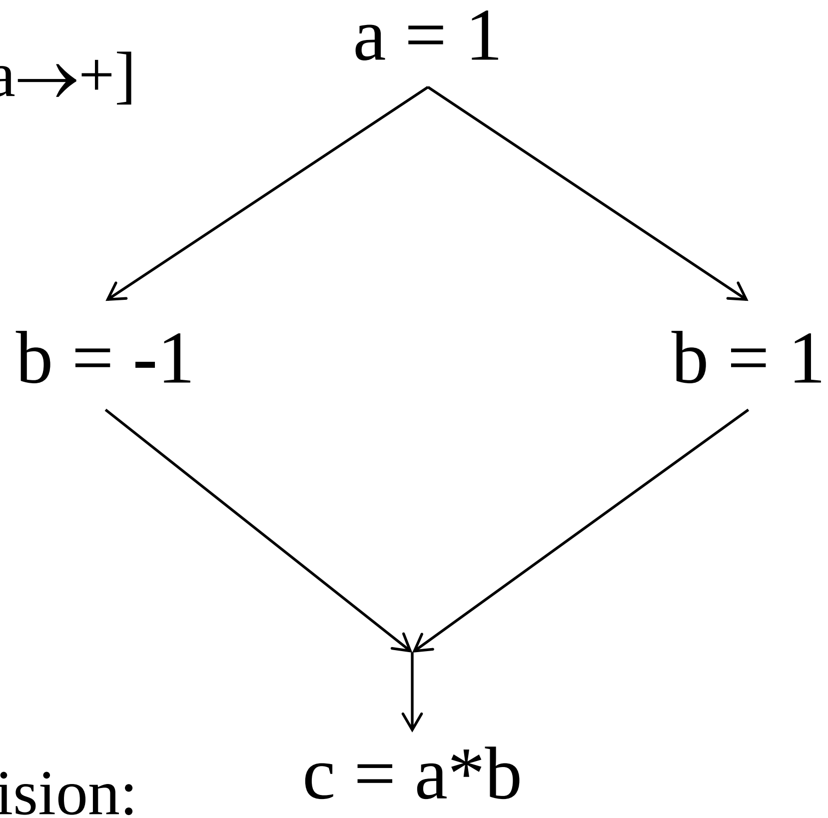
Example



Imprecision In Example

Abstract Imprecision:

[a→1] abstracted as [a→+]



Control Flow Imprecision:

[b→TOP] summarizes results of all executions. In any execution state s, AF(s)[b]≠TOP

General Sources of Imprecision

Abstraction Imprecision

- Concrete values (integers) abstracted as lattice values (-,0, and +)
- Lattice values less precise than execution values
- Abstraction function throws away information

Control Flow Imprecision

- One lattice value for all possible control flow paths
- Analysis result has a single lattice value to summarize results of multiple concrete executions
- Join operation $\vee$ moves up in lattice to combine values from different execution paths
- Typically if $x \leq y$, then x is more precise than y

Why To Allow Imprecision?

Make analysis tractable

Unbounded sets of values in execution

- Typically abstracted by finite set of lattice values

Execution may visit unbounded set of states

- Abstracted by computing joins of different paths

Intuition Behind Abstract Interpretation

Patrick Cousot's Description:

- http://www.di.ens.fr/~cousot/AI/IntroAbsInt.html

Also, Abstract interpretation vs Dataflow Analysis:

- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-820-fundamentals-of-program-analysis-fall-2015/lecture-notes/MIT6_820F15_L18.pdf

Abstract Interpretation

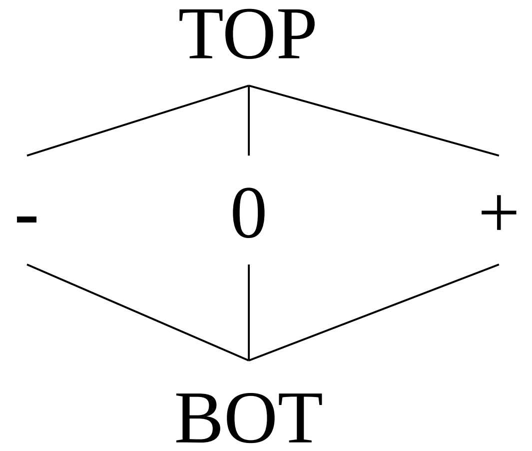
1. Define abstract domains (that represent important parts of program execution)

2. Define abstraction and concretization functions to relate the abstract domain with the program execution

3. Iterate updating the abstract state until convergence

ABSTRACT DOMAINS

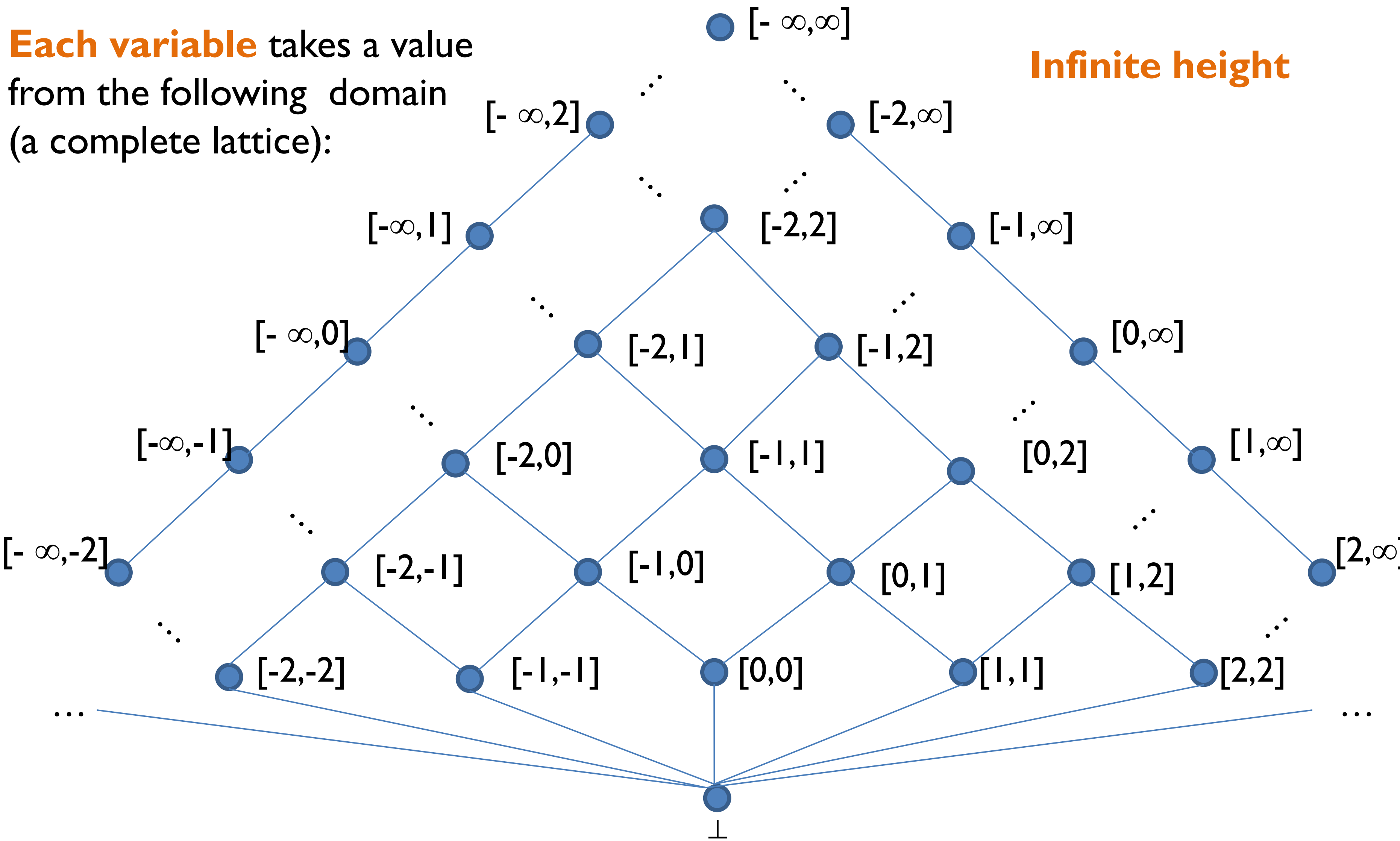
Sign Domain



Interval Domain

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

Infinite height

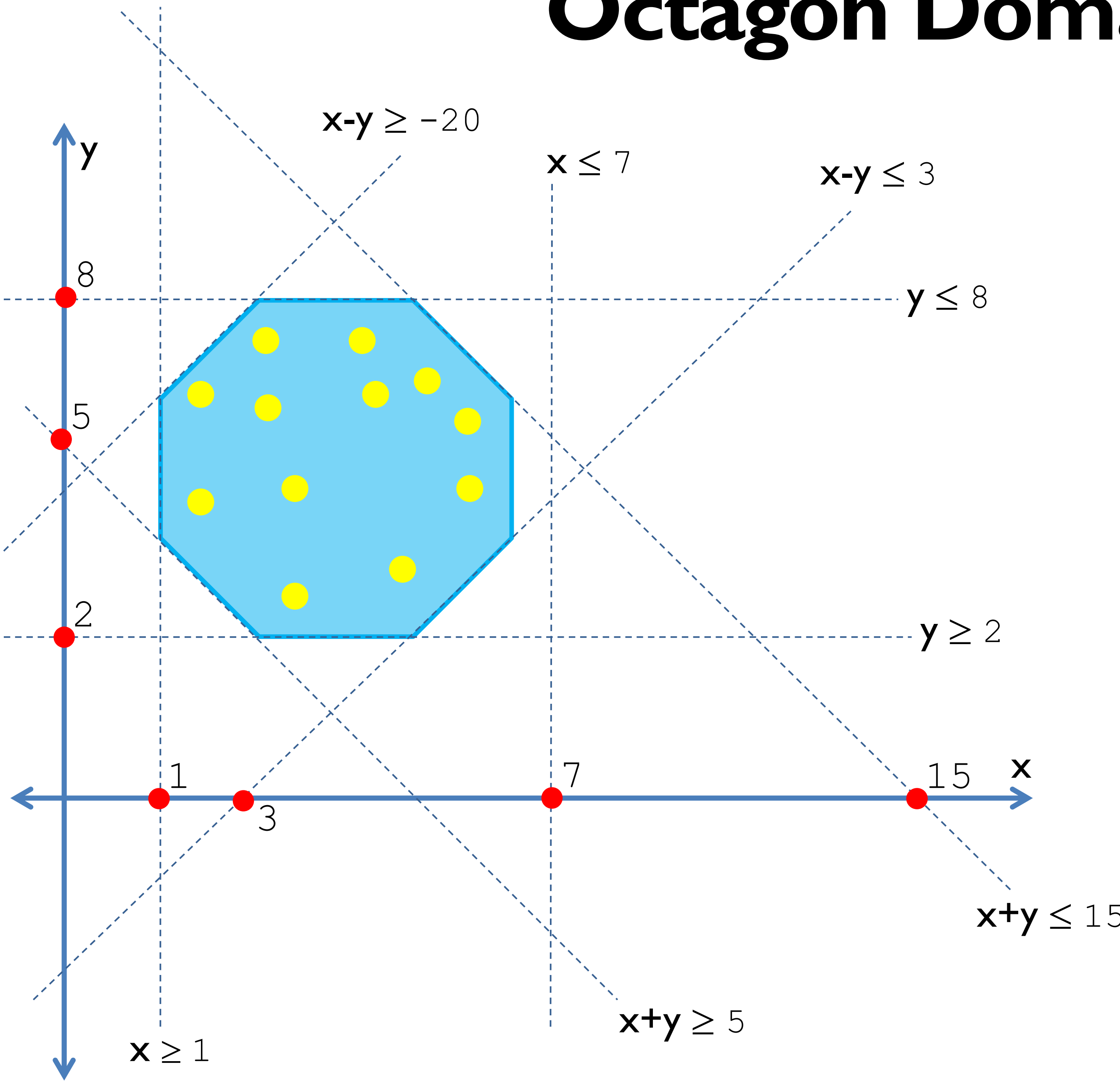


Relational Abstractions

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:

$\pm x \pm y \leq c$

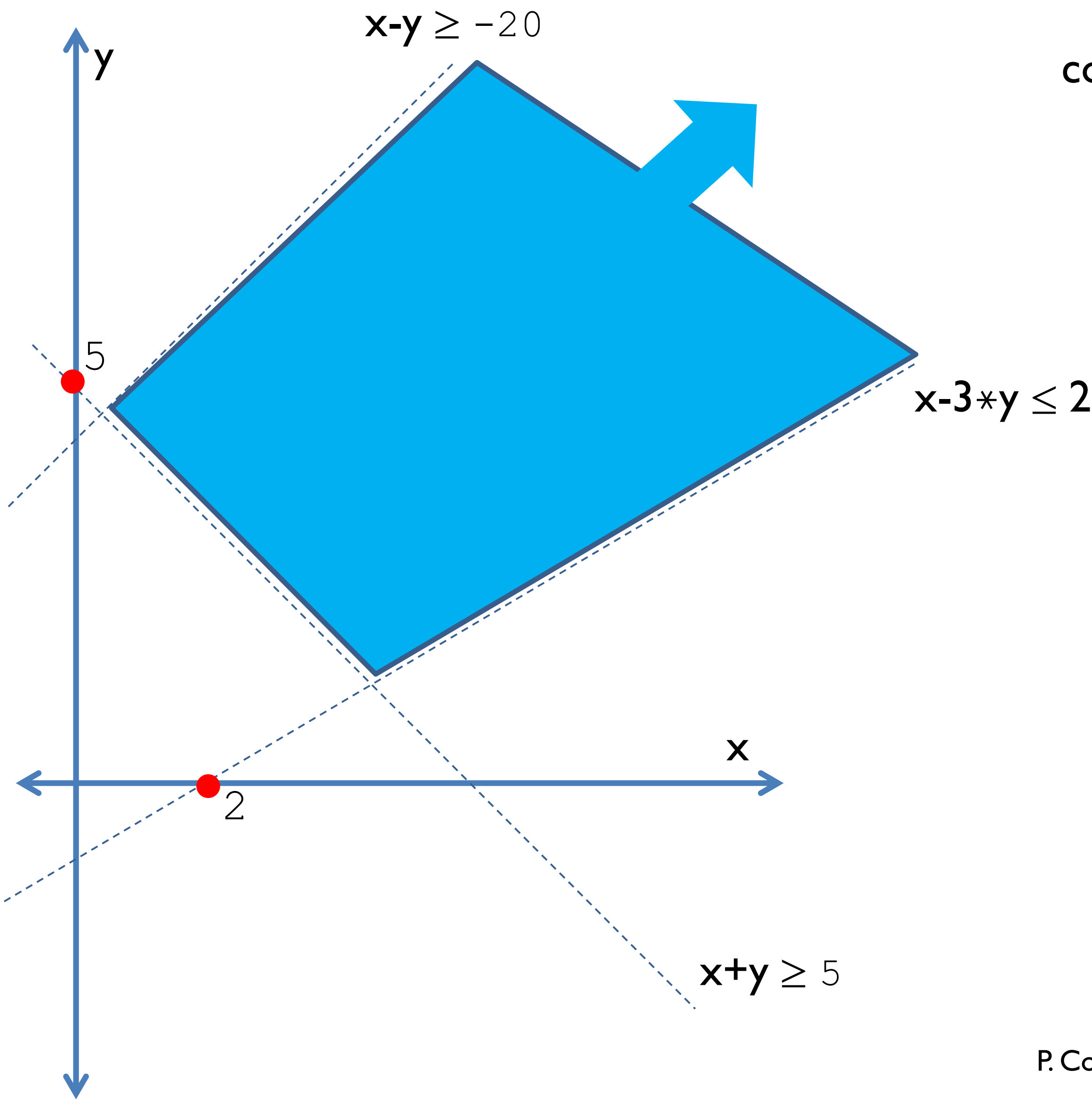
The slope is fixed

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

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

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

Polyhedra Domain



constraints are of the following form:

$c_1x_1 + c_2x_2 \ldots + c_nx_n \leq c$

the slope can vary

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

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

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

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).

Previous domains were numerical…

How do we represent program states?

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**:

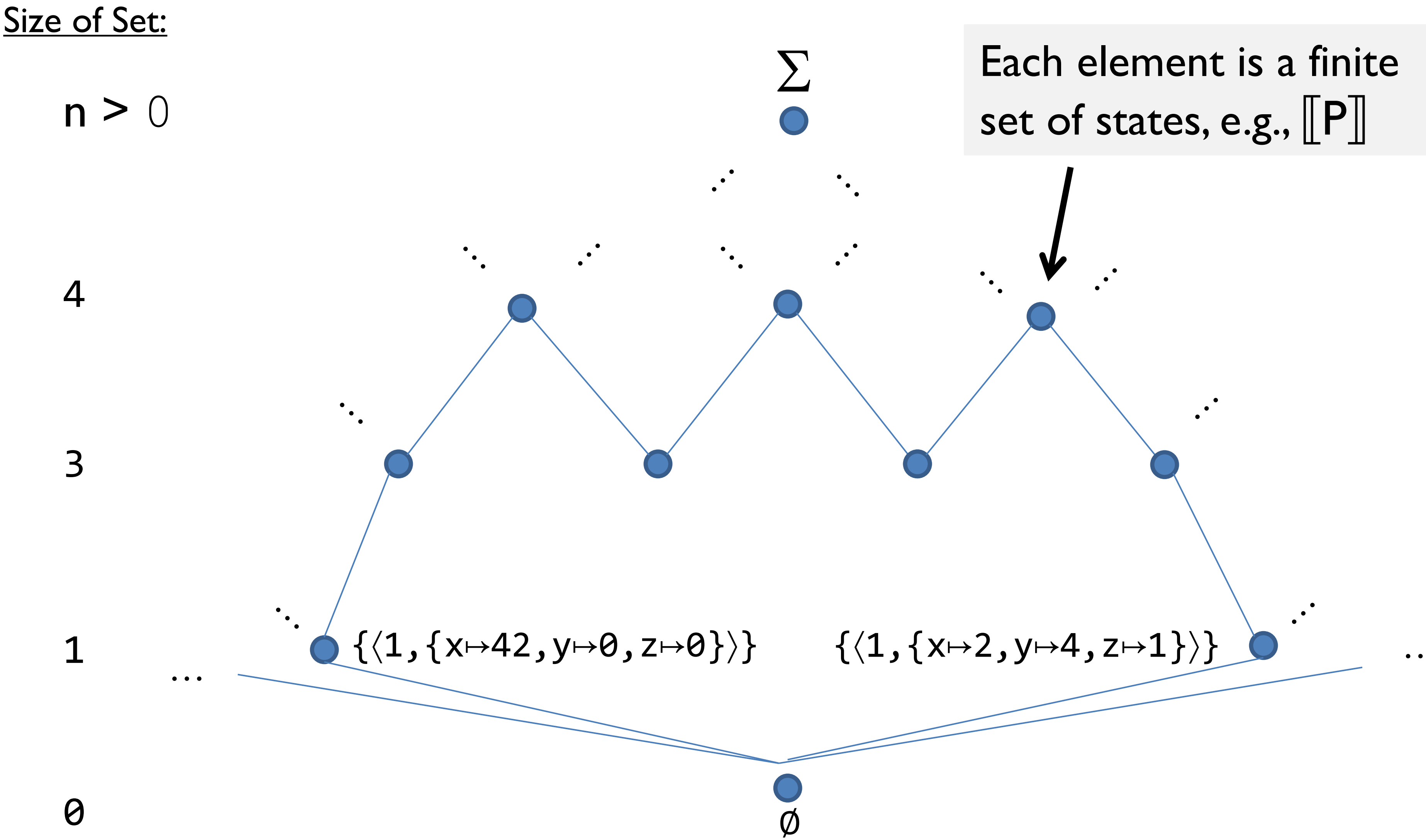
$$(\wp(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma)$$

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

Program Counter

Memory & Registers

Domain of Program States



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 → {Allocation Sites}, … }

e.g. { p → {A}, x → {A}, z → {A} }

Concretization:

{ pointer → {Objects allocated} … }

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

Details: http://www.srl.inf.ethz.ch/sae2014/Pointer.pdf

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 **at the fixed point:**

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

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

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

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

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

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

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

Details: http://www.srl.inf.ethz.ch/sae2014/Pointer.pdf

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 and for a programming language.

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

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

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:

$$F: C \rightarrow C$$

$$F^\sharp : C \rightarrow C$$

what does it mean for $F^\sharp$ to approximate F ?

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

Approximating a Function

What about when:

$$F: C \rightarrow C$$

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

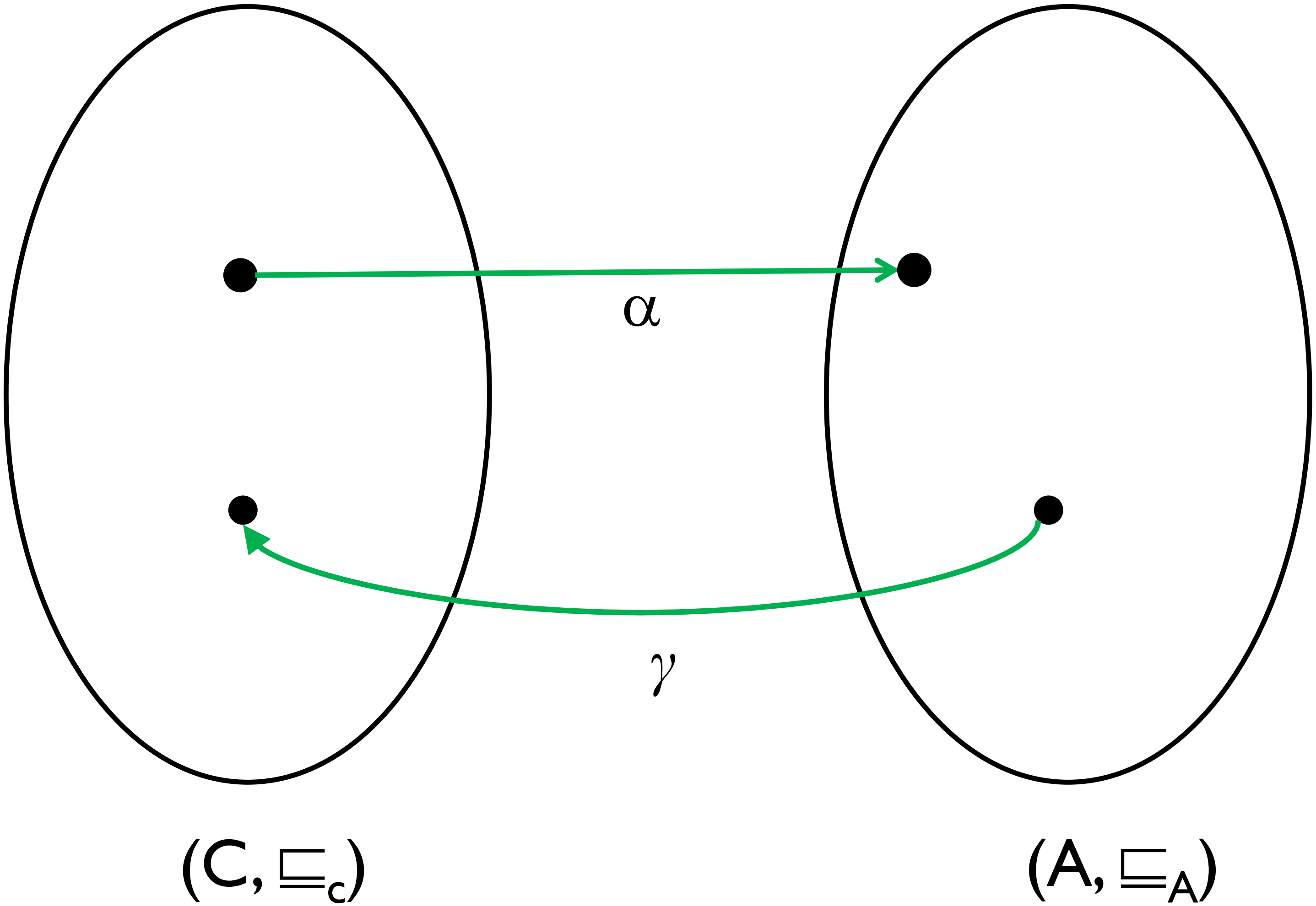
We need to connect the concrete C and the abstract A

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:

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

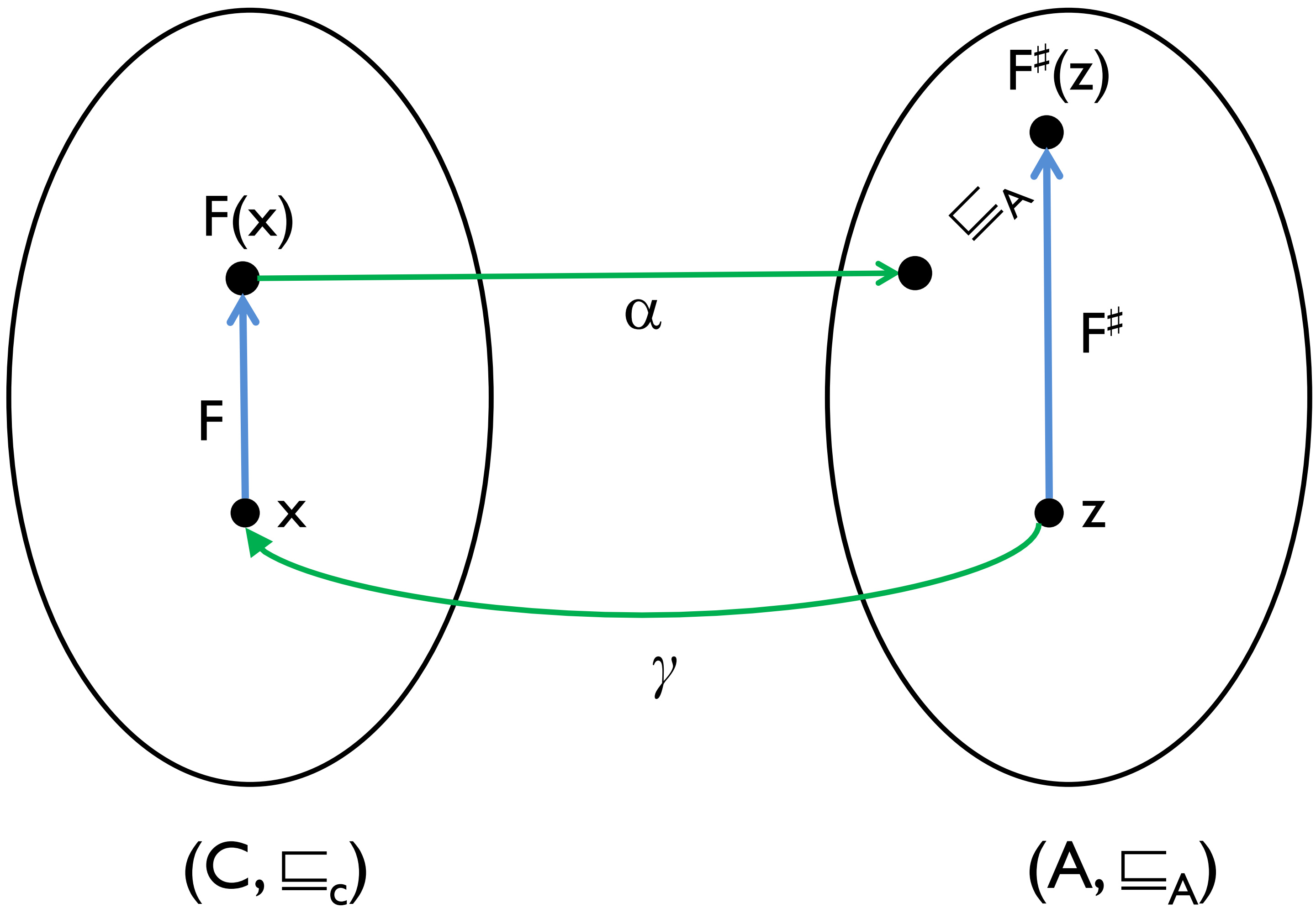
Galois Connection

For the course, **it is not important** to know 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



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 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.

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. **monotone** 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.

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 (∇)	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.

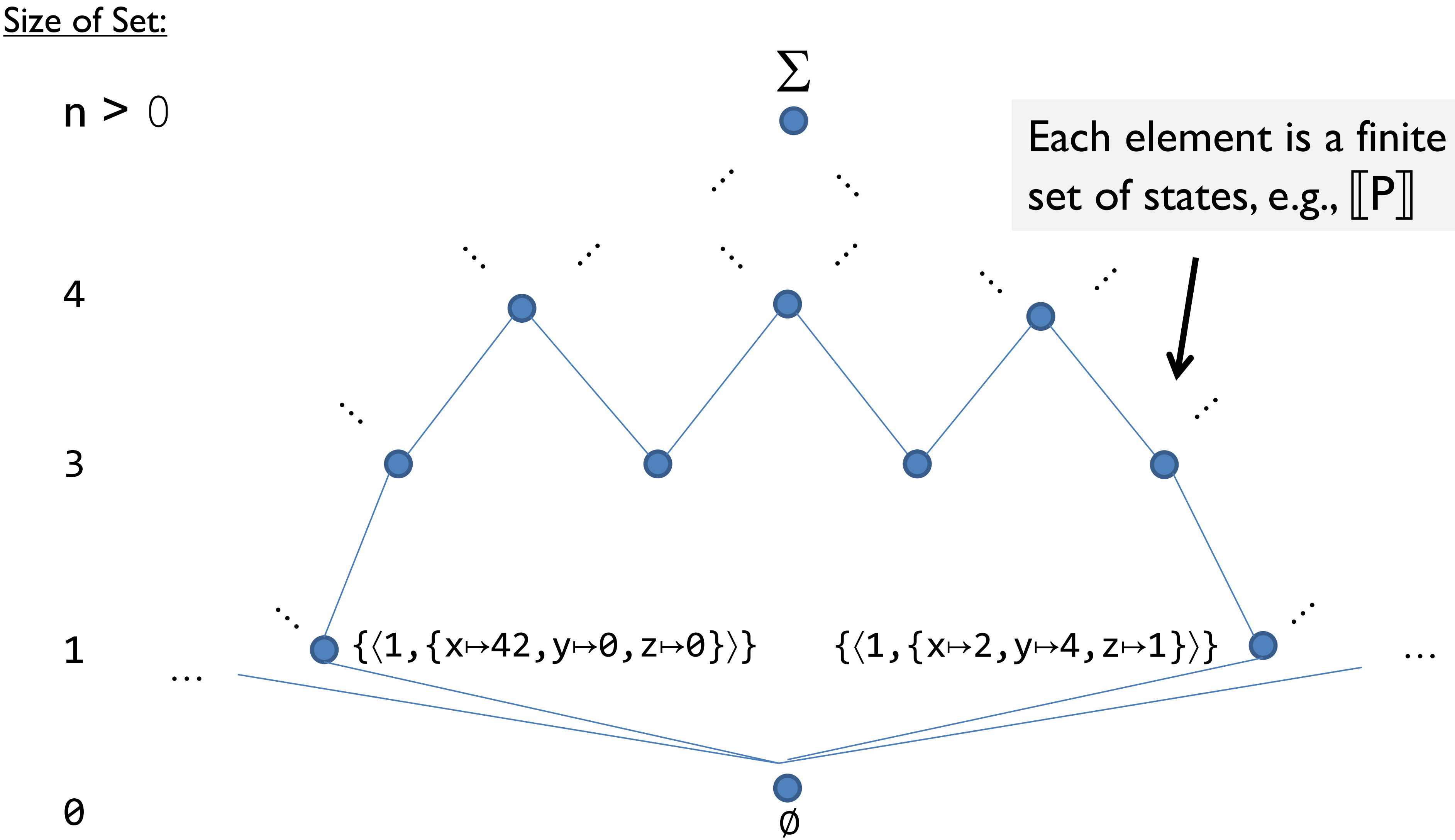
Domain of Program States

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**:

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

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

Starting Point: Domain of States



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

Let $[\![P]\!]$ be the **set of reachable states** of a program P.

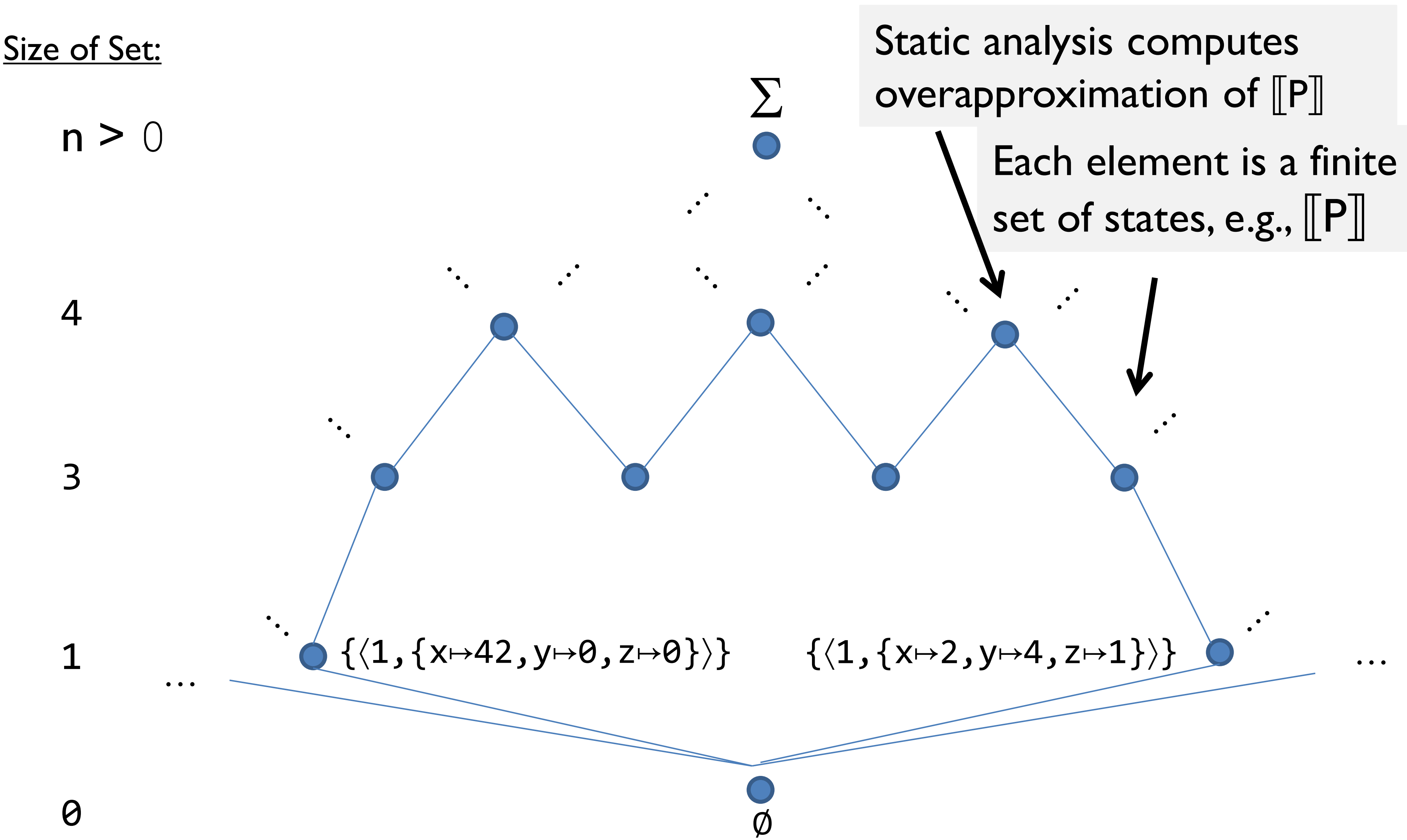
Def. Let function F be (where I is an initial set of states):

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

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

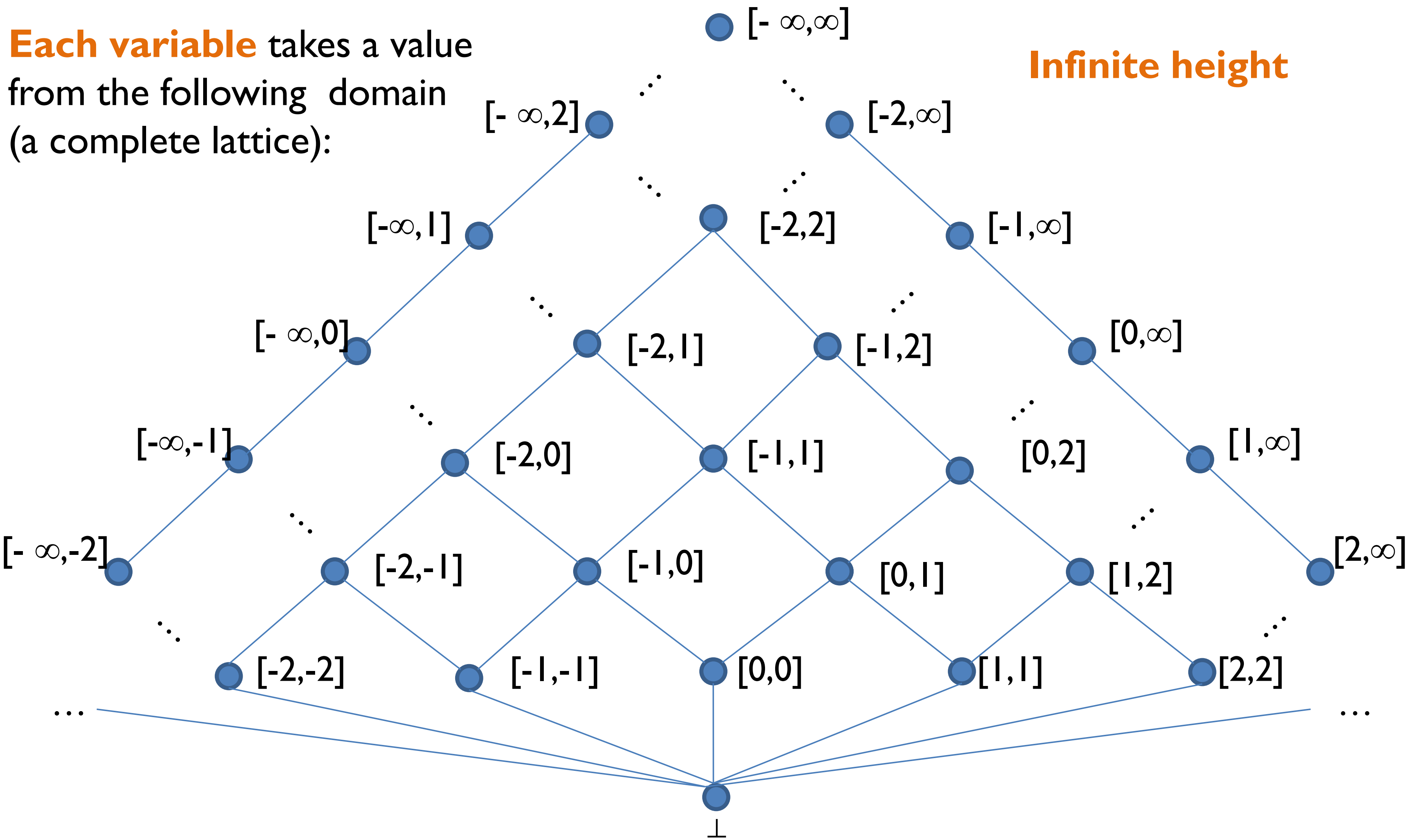
Interval Domain

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.

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 `meet(x,y)` returns $[x,y]$ if $x \leq y$ and $\bot_i$ otherwise

Intervals: Applied to Programs

The 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** $Lab \to (Var \to L^i)$

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

- $Var \to L^i$
- $Lab \to (Var \to L^i)$

Intervals: Applied to Programs

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

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

α^{i} ({ ⟨1,{x↦1,y↦9,q↦-2}⟩, ⟨1,{x↦5,y↦9,q↦-2}⟩, ⟨1,{x↦8,y↦9,q↦-2}⟩,
⟨1,{x↦1,y↦9,q↦4}⟩, ⟨1,{x↦5,y↦9,q↦4}⟩, ⟨1,{x↦8,y↦9,q↦4}⟩}
)
= 1 → (x ↦ [1,8], y ↦ [9,9], q ↦ [-2,4])

γ^{i} (1 → (x ↦ [1,8], y ↦ [9,9], q ↦ [-2,4])

= { ⟨1,{x↦1,y↦9,q↦-2}⟩, ⟨1,{x↦5,y↦9,q↦-2}⟩, ⟨1,{x↦8,y↦9,q↦-2}⟩,
⟨1,{x↦1,y↦9,q↦4}⟩, ⟨1,{x↦5,y↦9,q↦4}⟩, ⟨1,{x↦8,y↦9,q↦4}⟩,
⟨1,{x↦7,y↦9,q↦3}⟩, ⟨1,{x↦3,y↦9,q↦4}⟩, ⟨1,{x↦1,y↦9,q↦-1}⟩,
…, …, … }

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 $\alpha^i(\llbracket P \rrbracket)$

(or an over-approximation of it)

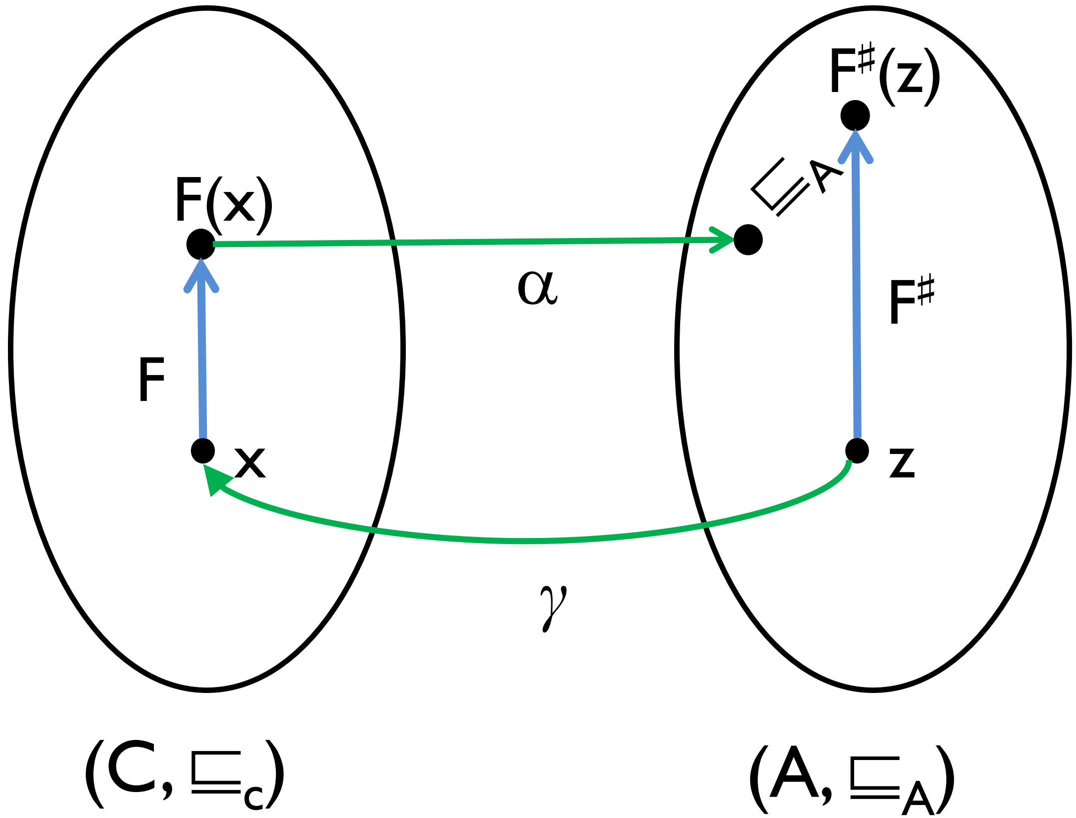
We need to approximate F

We want a function F^i where:

F^i : `(Lab → (Var → L`^{i}`))→(Lab → (Var → L`^{i}`))`

such that:

α^i (lfp F) $\sqsubseteq$ lfp F^i



Lets define F^i

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

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

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

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

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

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

```
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 $v \in L^i$

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 $\bot_i$ to any other element, we get $\bot_i$.

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

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

what about `*` ?

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

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

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

For a map m, we need to define : $[\![a_1 \ c \ a_2]\!]_i(m)$

which produces another map as a result.

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

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

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

- We simply

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 ?

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

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

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

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:
}
```

What is F^i?

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**

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:
}
```

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

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

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

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

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

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

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

Fixed point of F^i

Let us compute the least fixed point of F^i

Iterate 0

```
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:
}
```

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

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

```
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

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

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

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

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

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

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 $\rightarrow$ [$-\infty$,∞], y $\rightarrow$ [$-\infty$,∞], i $\rightarrow$ [$-\infty$,∞]

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

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

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

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

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

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

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

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

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

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

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 ⟶ [-∞,∞], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

2: x ⟶ [5,5], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

3: x ⟶ [5,5], y ⟶ [7,8], i ⟶ [-∞,∞]

4: x ⟶ [5,5], y ⟶ [7,8], i ⟶ [0,∞]

5: x ⟶ [5,5], y ⟶ [8,8], i ⟶ [0,∞]

6: x ⟶ [5,5], y ⟶ [8,8], i ⟶ [-1,∞]

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

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

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

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

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

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 → [-∞,∞], y → [-∞,∞], i → [-∞,∞]

2: x → [5,5], y → [-∞,∞], i → [-∞,∞]

3: x → [5,5], y → [7,8], i → [-∞,∞]

4: x → [5,5], y → [7,8], i → [0,∞]

5: x → [5,5], y → [8,9], i → [0,∞]

6: x → [5,5], y → [8,9], i → [-1,∞]

7: x → [5,5], y → [7,8], i → [-∞, -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: $x \longrightarrow [-\infty,\infty]$, $y \longrightarrow [-\infty,\infty]$, $i \longrightarrow [-\infty,\infty]$

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

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

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

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

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

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

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 $\rightarrow$ [-∞,∞], y $\rightarrow$ [-∞,∞], i $\rightarrow$ [-∞,∞]

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

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

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

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

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

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

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

```
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 ⟶ [-∞,∞], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

2: x ⟶ [5,5], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

3: x ⟶ [5,5], y ⟶ [7,9], i ⟶ [-∞,∞]

4: x ⟶ [5,5], y ⟶ [7,9], i ⟶ [0,∞]

5: x ⟶ [5,5], y ⟶ [8,10], i ⟶ [0,∞]

6: x ⟶ [5,5], y ⟶ [8,10], i ⟶ [-1,∞]

7: x ⟶ [5,5], y ⟶ [7,9], i ⟶ [-∞, -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 ⟶ [-∞,∞], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

2: x ⟶ [5,5], y ⟶ [-∞,∞], i ⟶ [-∞,∞]

3: x ⟶ [5,5], y ⟶ [7,10], i ⟶ [-∞,∞]

4: x ⟶ [5,5], y ⟶ [7,10], i ⟶ [0,∞]

5: x ⟶ [5,5], y ⟶ [8,10], i ⟶ [0,∞]

6: x ⟶ [5,5], y ⟶ [8,10], i ⟶ [-1,∞]

7: x ⟶ [5,5], y ⟶ [7,10], i ⟶ [-∞, -1]

The issue is that the iterates:

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

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:

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

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 $x^0 \sqsubseteq x^1 \sqsubseteq x^2 \sqsubseteq \ldots \sqsubseteq x^n \sqsubseteq \ldots$ 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:

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

will stabilizes 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: $lfp^{\sqsubseteq} F \sqsubseteq y^n$

Widening for Interval Domain

Let us define a widening operator for the intervals

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

[a, b] ∇_i [c, d] = [e, f] where:

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

if d > b, then f = ∞, 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

$it^0 = \bot$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

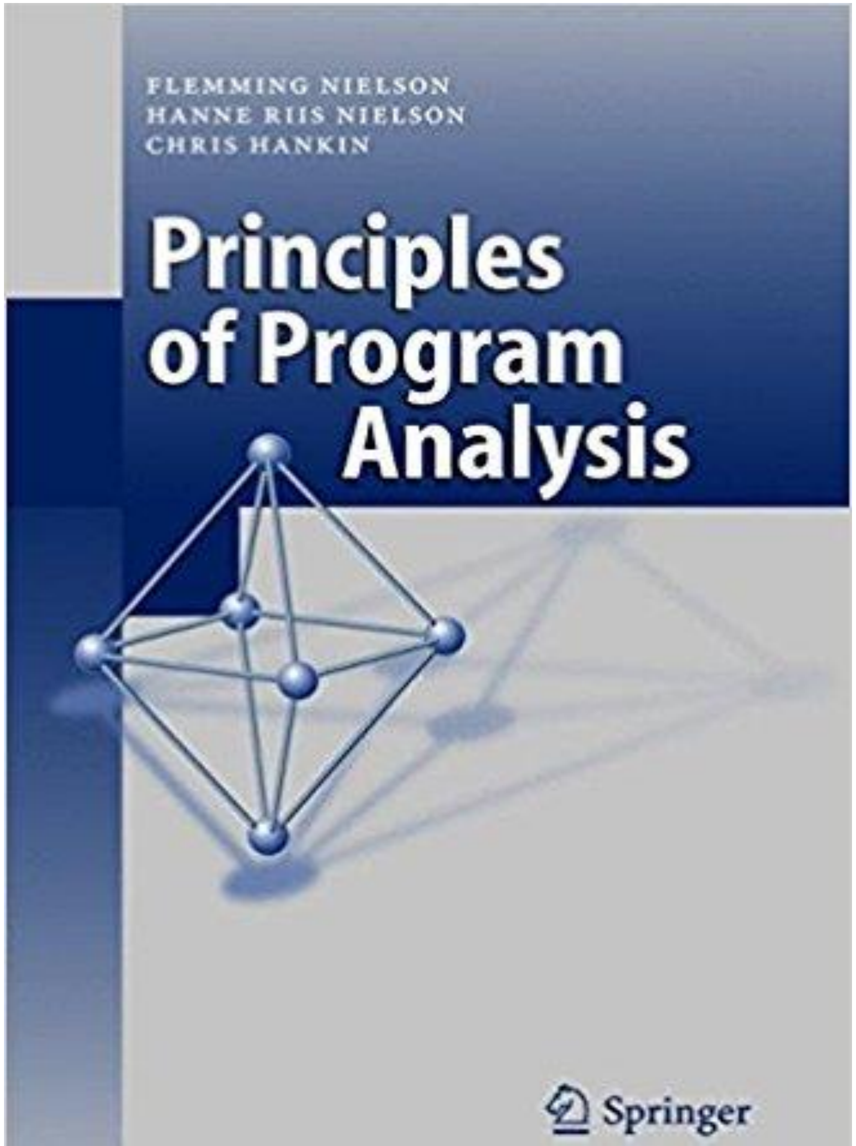
Chaotic (Asynchronous) Iteration

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

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

Additional Reading



Covers many aspects of static analysis:

- Full formalization of dataflow analysis
- Lattices
- Detailed introduction to abstract interpretation

Good news:

- Available from Springer website (from academic network)

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)