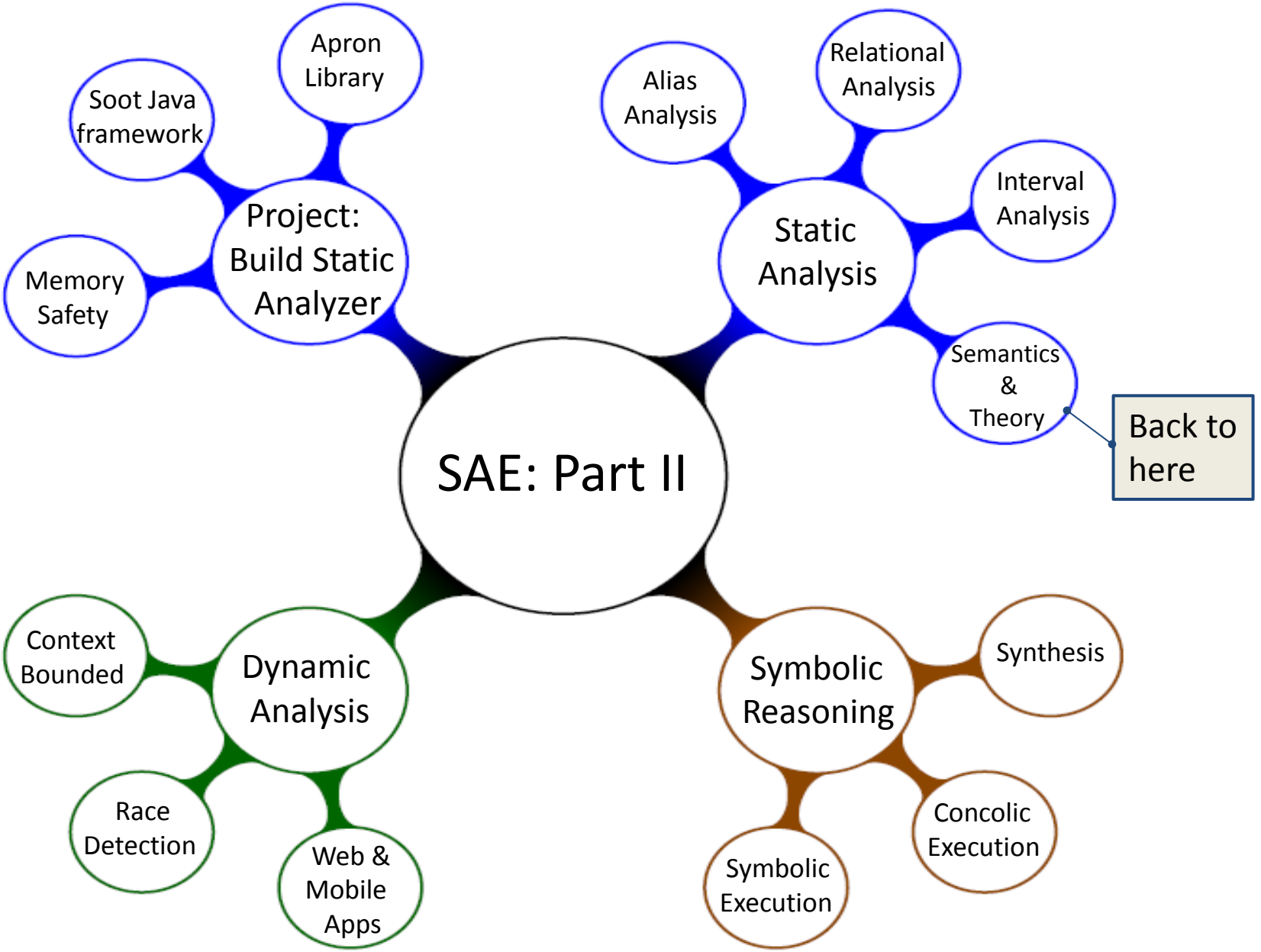
Software Architecture and Engineering: Part II

ETH Zurich, Spring 2014

Prof. Martin Vechev

SRL
SOFTWARE RELIABILITY LAB

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich


SAE: Part II
Project: Build Static Analyzer
Apron Library
Soot Java framework
Memory Safety
Static Analysis
Alias Analysis
Relational Analysis
Interval Analysis
Semantics & Theory
Back to here
Dynamic Analysis
Context Bounded
Race Detection
Web & Mobile Apps
Symbolic Reasoning
Synthesis
Concolic Execution
Symbolic Execution

Mathematical Concepts

- Structures: posets, lattices
- Functions: monotone, fixed points
- Approximating functions

Structures: Motivation

Structures are important as they define the concrete and abstract domains

Partially Ordered Sets (posets)

A partial order is a binary relation $\sqsubseteq \subseteq L \times L$ on a set L with these properties:

- Reflexive: $\forall p \in L: p \sqsubseteq p$
- Transitive: $\forall p,q,r \in L: (p \sqsubseteq q \wedge q \sqsubseteq r) \Rightarrow p \sqsubseteq r$
- Anti-symmetric: $\forall p,q \in L: (p \sqsubseteq q \wedge q \sqsubseteq p) \Rightarrow p = q$

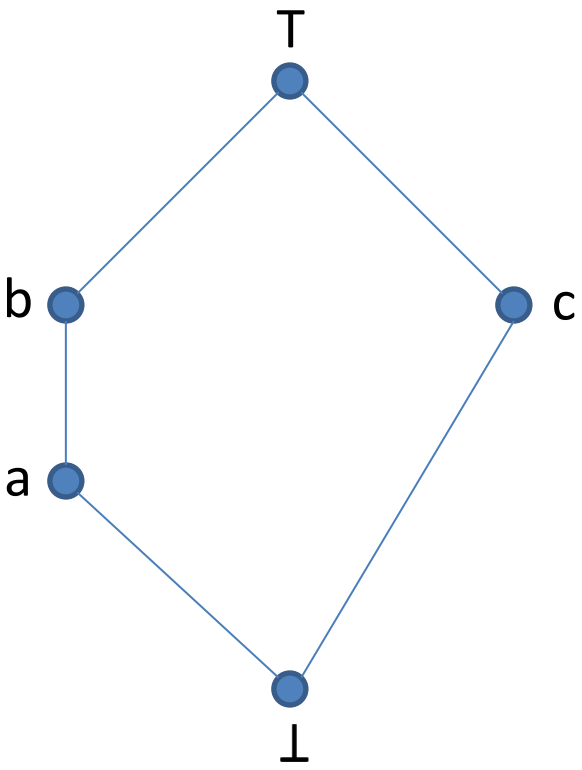
A poset $(L, \sqsubseteq)$ is a set L equipped with a partial ordering $\sqsubseteq$

- For example: $(\wp(L), \subseteq)$ is a poset, where $\wp$ denotes powerset

Intuition: captures implication between facts

- $p \sqsubseteq q$ intuitively means that $p \Rightarrow q$
- Later, we will say that if $p \sqsubseteq q$, then p is "more precise" than q (that is, p represents fewer concrete states than q)

Posets shown as Hasse Diagrams



given the set { a,b,c, T, ⊥ }

the Hasse diagram shows the order:

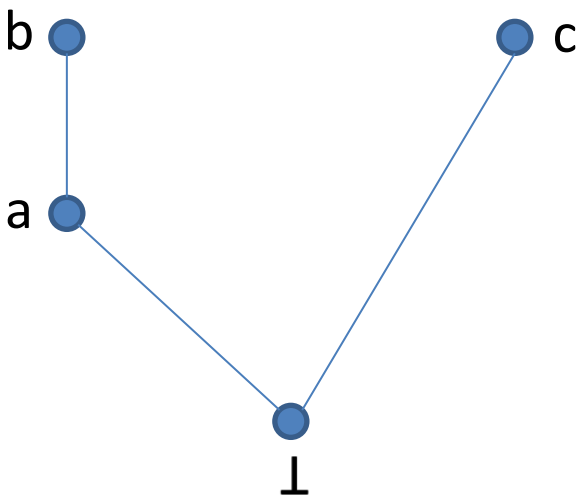
{ (⊥ ,a), (⊥ ,c), (a, b), (b, T), (c, T),
(a, a), (b, b), (c, c), (T, T), (⊥ ,⊥),
(⊥ ,b), (⊥ , T), (a, T) }

Least / Greatest in Posets

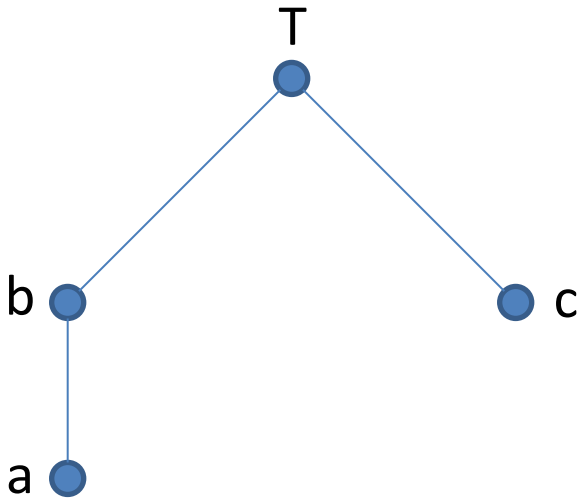
Given a poset $(L, \sqsubseteq)$, an element $\bot \in L$ is called the least element if it is smaller than all other elements of the poset: $\forall p \in L: \bot \sqsubseteq p$. The greatest element is an element T if $\forall p \in L: p \sqsubseteq T$.

The least and greatest elements may not exist, but if they do they are unique.

Least / Greatest: example



No greatest element



No least element

Example where both do not exist ?

Bounds in Posets

Given a poset $(L, \sqsubseteq)$ and $Y \subseteq L$:

$u \in L$ is an upper bound of Y if $\forall p \in Y: p \sqsubseteq u$

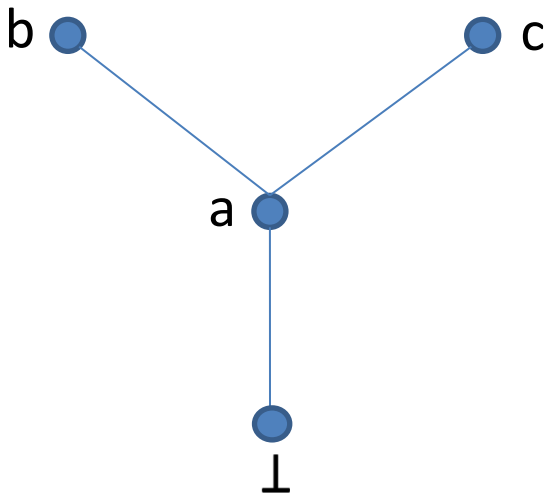
$l \in L$ is a lower bound of Y if $\forall p \in Y: p \sqsupseteq l$

note that the bounds for Y may not exist

- $\bigsqcup Y \in L$ is a least upper bound of Y if $\bigsqcup Y$ is an upper bound of Y and $\bigsqcup Y \sqsubseteq u$ whenever u is another upper bound of Y.
- $\bigsqcap Y \in L$ is greatest lower bound of Y if $\bigsqcap Y$ is a lower bound of Y and $\bigsqcap Y \sqsupseteq l$ whenever l is another lower bound of Y

– Note that $\bigsqcup Y$ and $\bigsqcap Y$ need not be in Y.
– We often write $p \sqcup q$ for $\bigsqcup\{p, q\}$ and $p \sqcap q$ for $\bigsqcap\{p, q\}$

Bounds: example



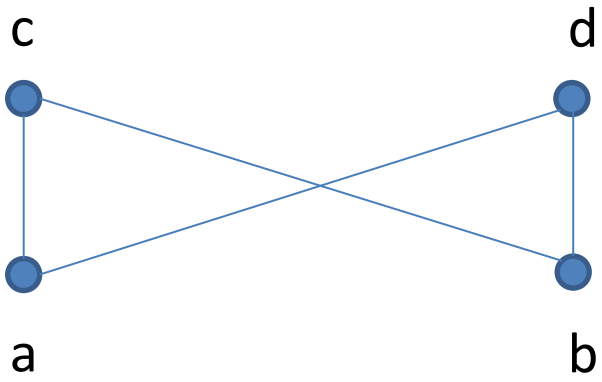
{b, c} has no upper bound

{b, c} has 2 lower bounds: a and ⊥

where ⊓ {b, c} = a

No T element

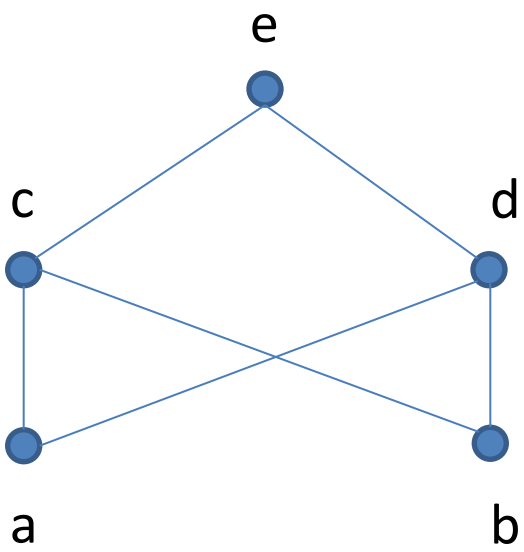
Bounds: example



is there a $\sqcup$ for ‘a’ and ‘b’ ?

is there a $\sqcap$ for ‘c’ and ‘d’ ?

Bounds: example



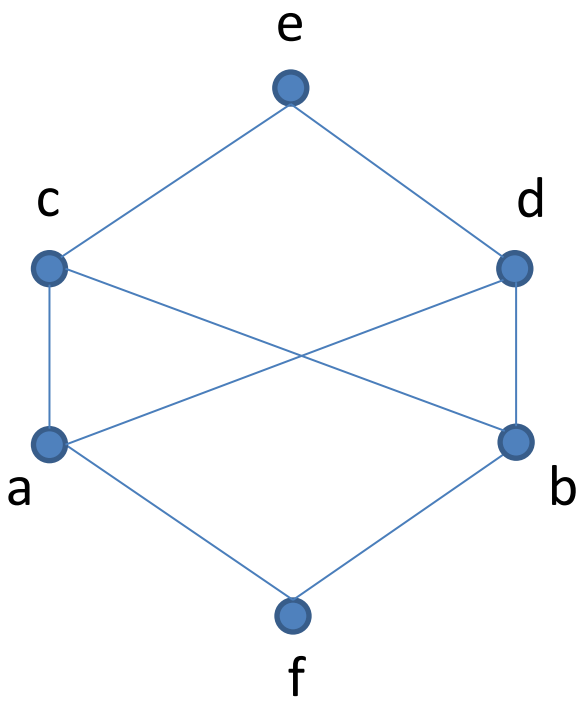
is there a $\sqcup$ for ‘a’ and ‘b’ ?

Complete Lattices

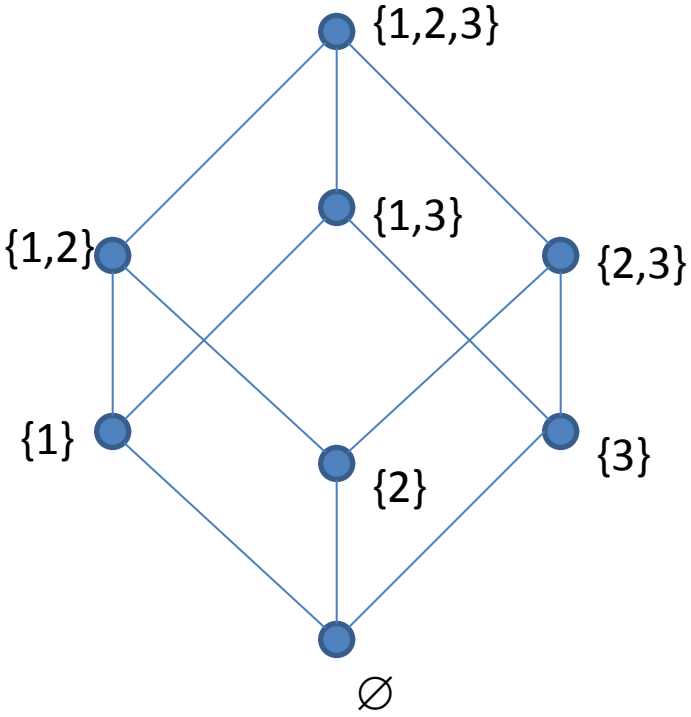
A complete lattice $(L, \sqsubseteq, \bigsqcup)$ is a poset where $\bigsqcup Y$ and $\bigsqcap Y$ exist for any $Y \subseteq L$.

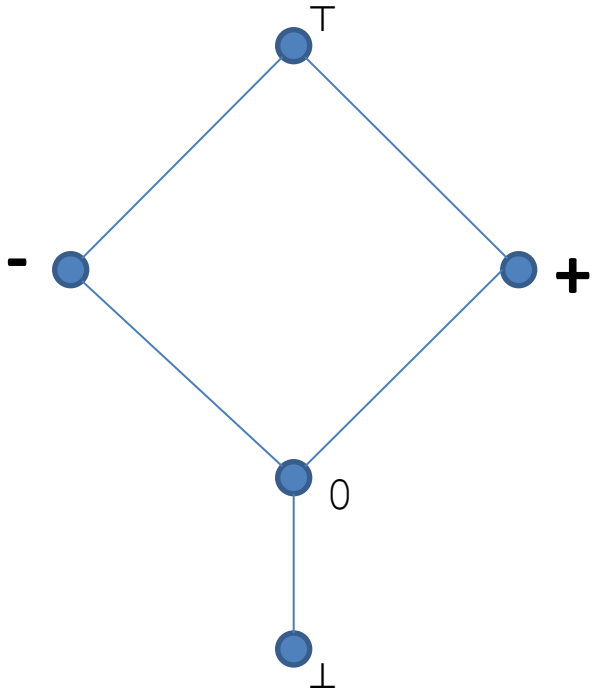
For example, for a set L, $(\wp(L), \subseteq, \cup, \cap)$ is a complete lattice.

Is this a complete lattice ?

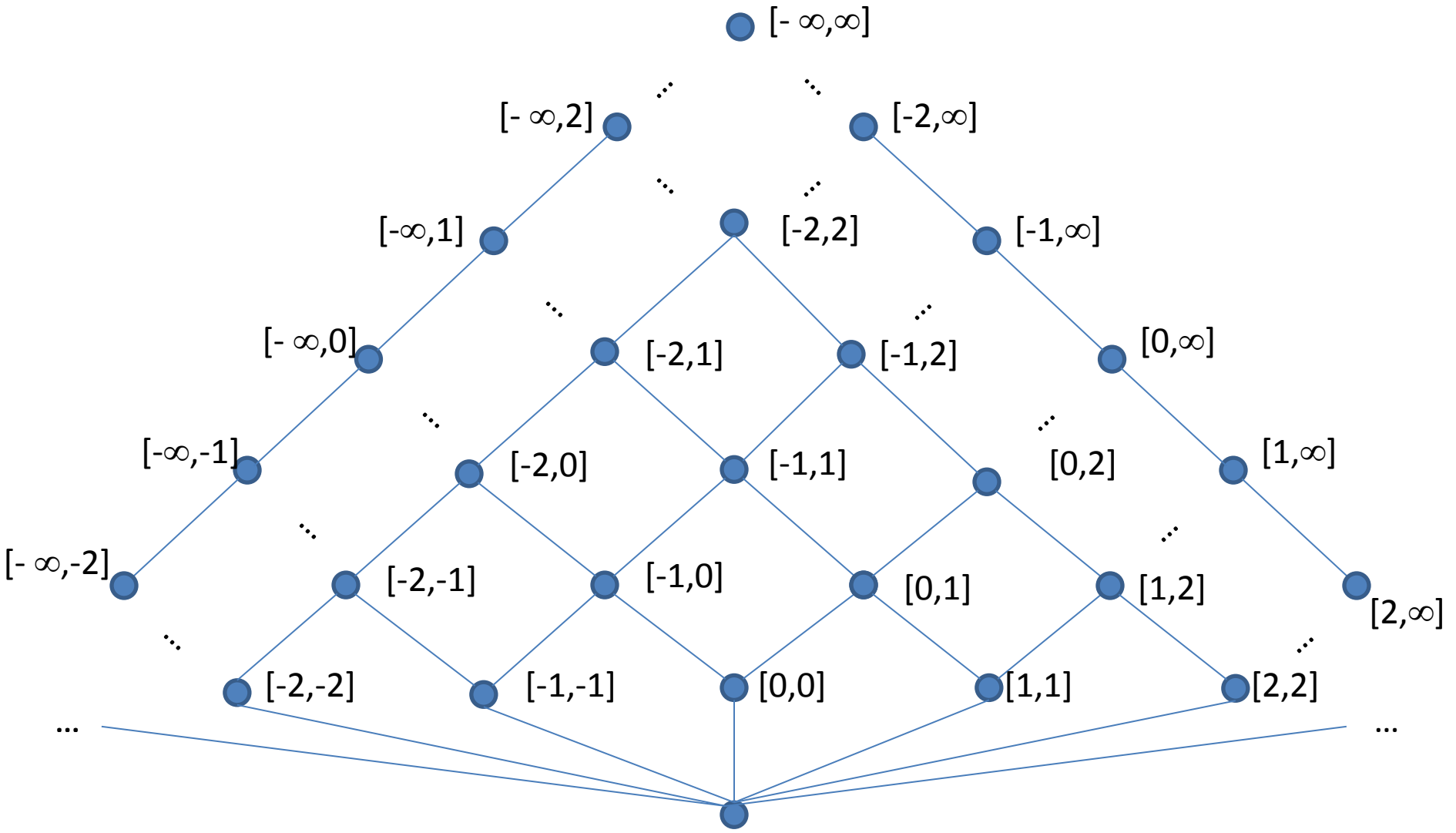


Complete Lattices: Examples





Complete Lattices: Examples



Later we will see that the set of traces $[\![P]\!]$ also belongs to a complete lattice

Concepts

- Structures: posets, lattices
- Functions: monotone, fixed points
- Approximating functions

Functions

A function $f: A \longrightarrow B$ between two posets $(A, \sqsubseteq)$ and $(B, \leq)$ is increasing (monotone): $\forall a,b \in A: a \sqsubseteq b \Rightarrow f(a) \leq f(b)$

Often, we use the special case where the function is between elements in the same poset. That is, $f: A \longrightarrow A$. Then a monotone function is: $\forall a,b \in A: a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b)$

Fixed Points

For a poset $(L, \sqsubseteq)$, function $f: L \longrightarrow L$, and element $x \in L$:

- x is a fixed point iff $f(x) = x$
- x is a post-fixedpoint iff $f(x) \sqsubseteq x$

Fix(f) denotes the set of all fixed points

Red(f) = set of all post-fixedpoints

Least Fixed Points

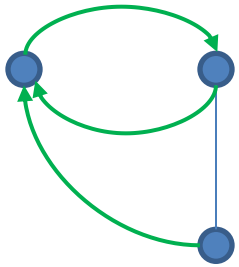
For a poset $(L,\sqsubseteq)$ and a function $f: L \longrightarrow L$, we say that

$\text{lfp}^{\sqsubseteq} f \in L$ is a least fixed point of f if:

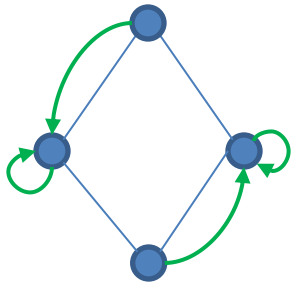
- $\text{lfp}^{\sqsubseteq} f$ is a fixed point
- It is the least fixed point: $\forall a \in L: a = f(a) \Rightarrow \text{lfp}^{\sqsubseteq} f \sqsubseteq a$

Note that the least fixed point may not exist.

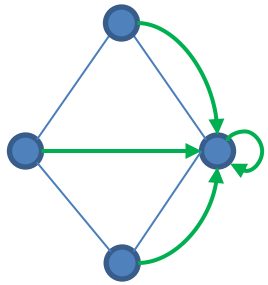
Fixed Points: Examples



- monotone function
- with no fixed point

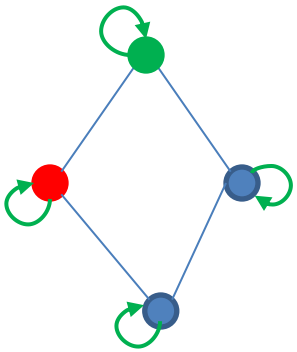


- not monotone function
- with 2 fixed points
- no least fixed point



- monotone function
- with one fixed point
- has a least fixed point

- monotone function
- with 2 fixed points
- no least fixed point



there exists a post-fixedpoint
that is less than some fixed point

- monotone function
- 4 fixed points
- least fixed point

Tarski's fixed point theorem (part of it)

If $(L,\sqsubseteq,\sqcup,\sqcap,\bot,\top)$ is a complete lattice and $f: L \longrightarrow L$ is a monotone function, then

$$\mathrm{lfp}^{\sqsubseteq} f \text{ exists, and}$$

$$\mathrm{lfp}^{\sqsubseteq} f = \sqcap \mathrm{Red}(f) \in \mathrm{Fix}(f)$$

Note: the complete lattice can be of infinite height

Tarski's theorem tells us that a fixed point exists, but does not actually suggest an algorithm for computing it.

Next: we look at ways to compute a fixed point

Function Iterates

For a poset $(L, \sqsubseteq)$, a function $f: L \rightarrow L$, an element $a \in L$, the iterates of the function from a are:

$$f^0(a), f^1(a), f^2(a)\ldots$$

where $f^{n+1}(a) = f(f^n(a))$

Note that $f^0(a) = a$

In program analysis, we usually take a to be $\bot$

A useful fixed point theorem

Given a poset of finite height, a least element $\bot$, a monotone f.

Then the iterates $f^0(\bot)$, $f^1(\bot)$, $f^2(\bot)$… form an increasing sequence which eventually stabilizes from some $n \in N$, that is: $f^n(\bot) = f^{n+1}(\bot)$ and:

$$\text{lfp}^{\sqsubseteq} f = f^n(\bot)$$

This leads to a simple algorithm for computing $\text{lfp}^{\sqsubseteq} f$

Concepts

- Structures: posets, lattices
- Functions: monotone, fixed points
- Approximating functions

Over-approximate $[\![P]\!]$

Recall:

$$[\![P]\!] = \{ c_0 \cdot c_1 \cdot \ldots \cdot c_{n-1} \mid n \geq 1 \wedge c_0 \in I \wedge \forall i \in [0, n-2]: c_i \rightarrow c_{i+1} \}$$

Consider the function F:

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

$[\![P]\!]$ is a least fixed point of F:

$$F([\![P]\!]) = [\![P]\!]$$

The Art of 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 $F^\sharp(V) = V$

Approximating a Function

given functions:

$$F: C \longrightarrow C$$

$$F^{\sharp}: C \longrightarrow C$$

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:

$$F: C \longrightarrow C$$

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

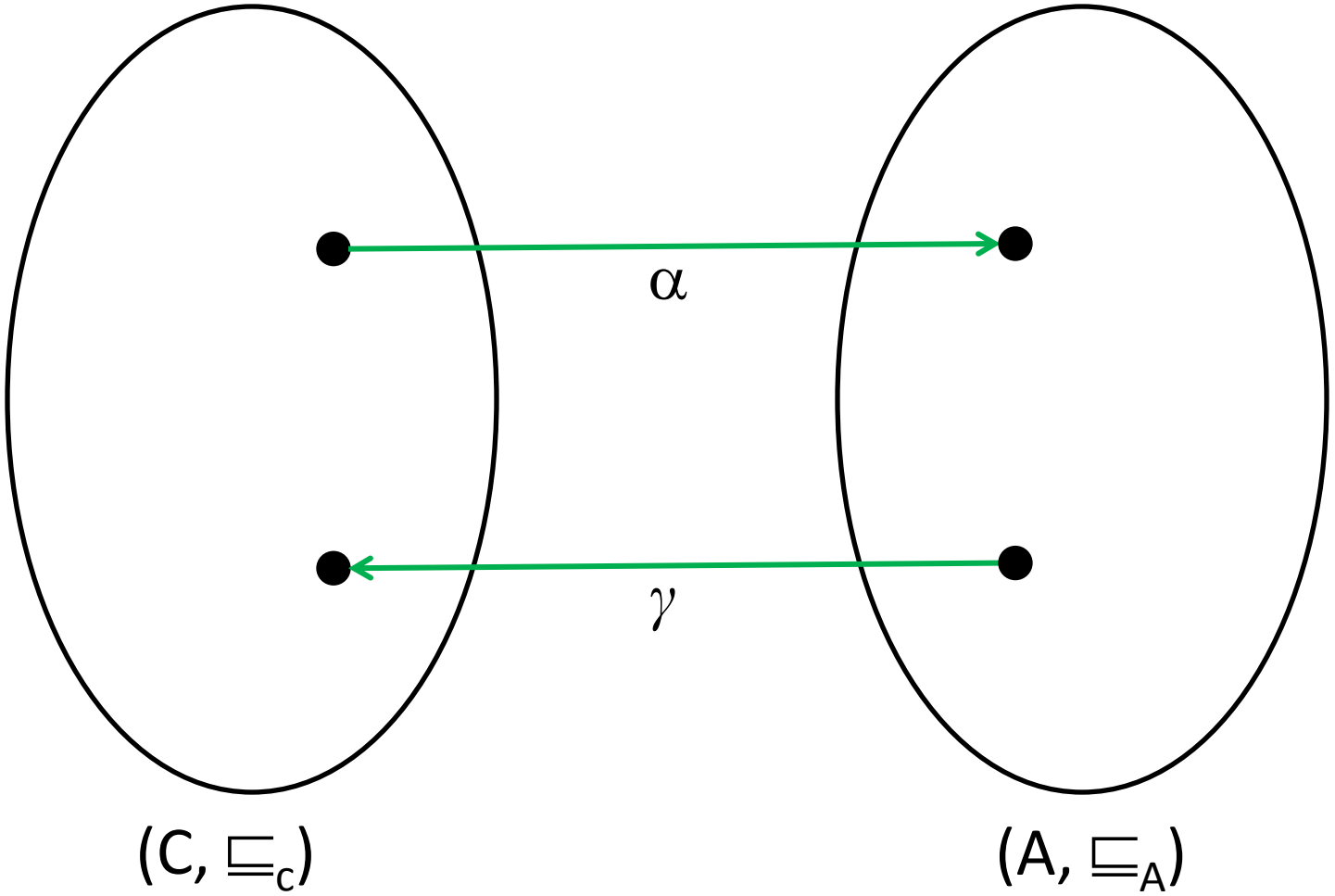
We need to connect the concrete C and the abstract A

We will connect them via two functions α and γ

$\alpha : C \longrightarrow A$ is called the abstraction function

$\gamma : A \longrightarrow C$ is called the concretization function

Connecting Concrete with Abstract



Approximating a Function: Definition 1

So we have the 2 functions:

$$F: C \longrightarrow C$$

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

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

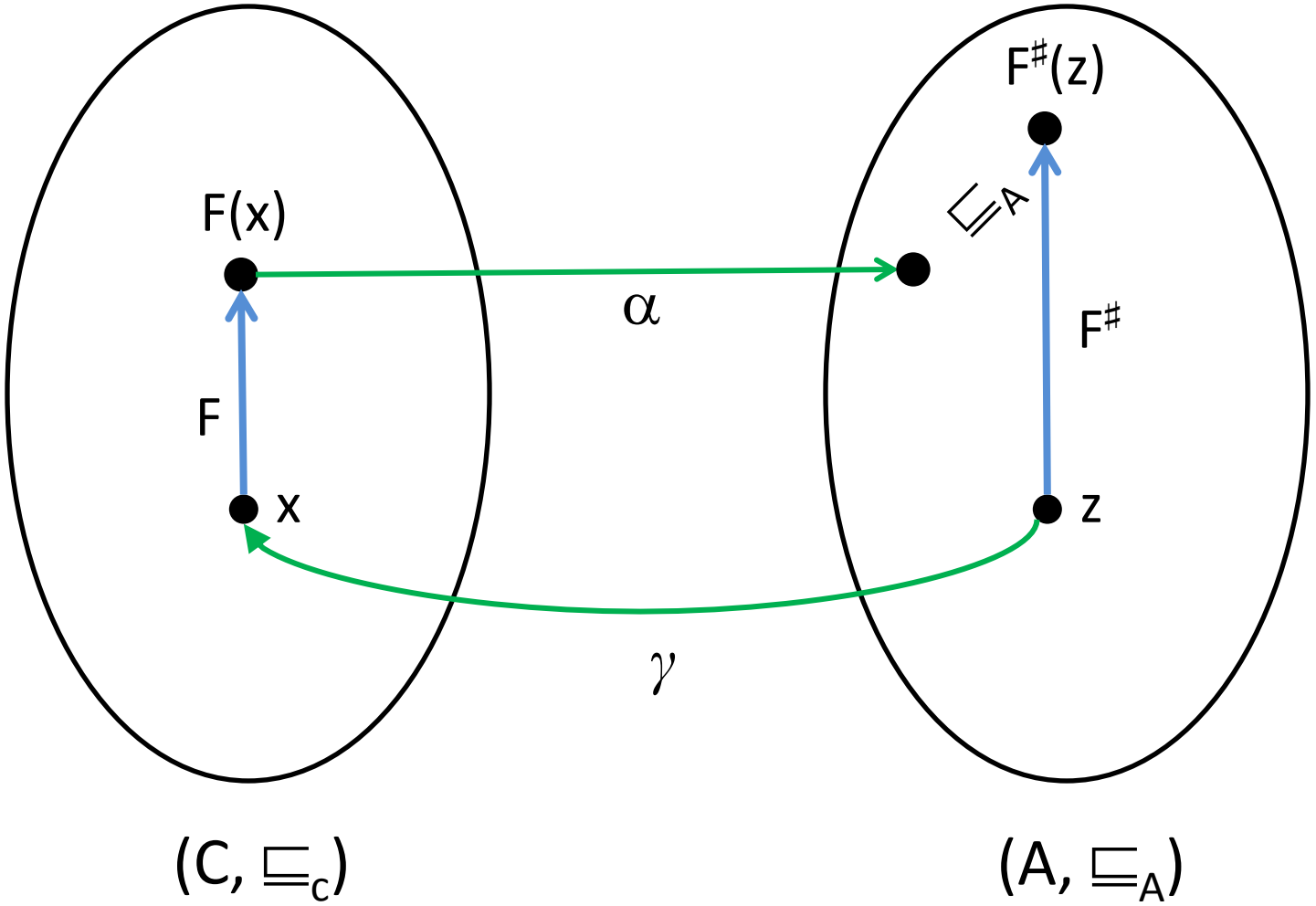
$$\forall z \in A : \alpha(F(\gamma(z))) \sqsubseteq_A F^{\sharp}(z)$$

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 what α and γ can be.

For instance, among other things, they require α to be monotone.

Visualizing Definition 1



Approximating a Function

what this equation:

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

says is 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, concretizing it, applying the concrete function and abstracting back again is less than applying the function in the abstract directly.

Least precise approximation

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

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

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

Most precise approximation

What if $F^\sharp(z) = \alpha(F(\gamma(z)))$? This is the best abstract function.

The problem is that we often cannot implement such a $F^\sharp(z)$ 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 $F: C \longrightarrow C$ and $F^\sharp : A \longrightarrow A$
2. $\alpha : C \longrightarrow A$ and $\gamma : A \longrightarrow C$ 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 !

Approximating a Function: Definition 2

So we have the 2 functions:

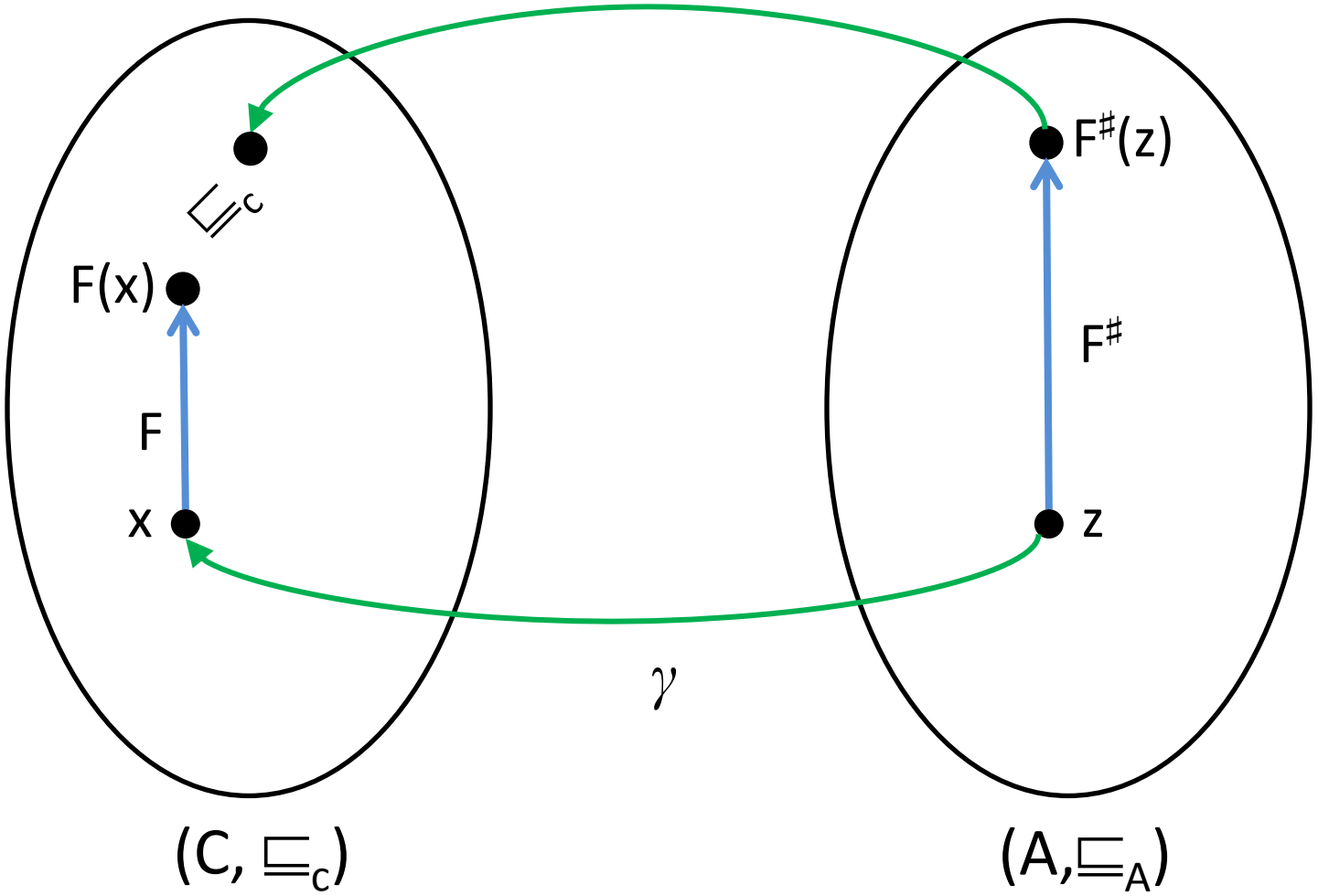
$$F: C \longrightarrow C$$

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

But what if α and γ do not form a Galois Connection ? For instance, α is not monotone. Then, we can use the following definition of approximation:

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

Visualizing Definition 2 (concretization-based)



Key Theorem II: Least Fixed Point Approximation

If we have:

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

then:

$$lfp(F) \sqsubseteq_c \gamma\,(lfp\,(F^\sharp))$$

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

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.

$F^\sharp$

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

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

$[\![\text{action}]\!] : \mathtt{A} \to \mathtt{A}$

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

$[\![\text{action}]\!]$ is often referred to as the abstract transformer.

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

```
foo (int i) {

1: int x := 5;
2: int y := 7;

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

Actions:

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

Multiple (two) actions reach label 3

what is action ?

An action can be:

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

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

$$\{x \mapsto 2,\ y \mapsto 0\} \xrightarrow{x:=y+x} \{x \mapsto 4,\ y \mapsto 0\}$$

$$\{x \mapsto 2,\ y \mapsto 0\} \xrightarrow{\texttt{if (x > 5) …}}$$

Defining ⟦action⟧

As we said, ⟦action⟧ captures the abstract semantics of the language for a particular abstract domain.

In later lectures we will see precise definitions for some actions in the Interval domain. Defining ⟦action⟧ for complex domains such as say Octagon can be quite tricky.

Lets just have a brief example now to what it entails even for Intervals…

Example: what is $[\![x \leq y]\!]$ for Intervals ?

Suppose we have the program:

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

What does $[\![\texttt{x} \leq \texttt{y}]\!]$ produce at label 1 ?

That is, what are $\texttt{x}$ and $\texttt{y}$ at label 1 ?

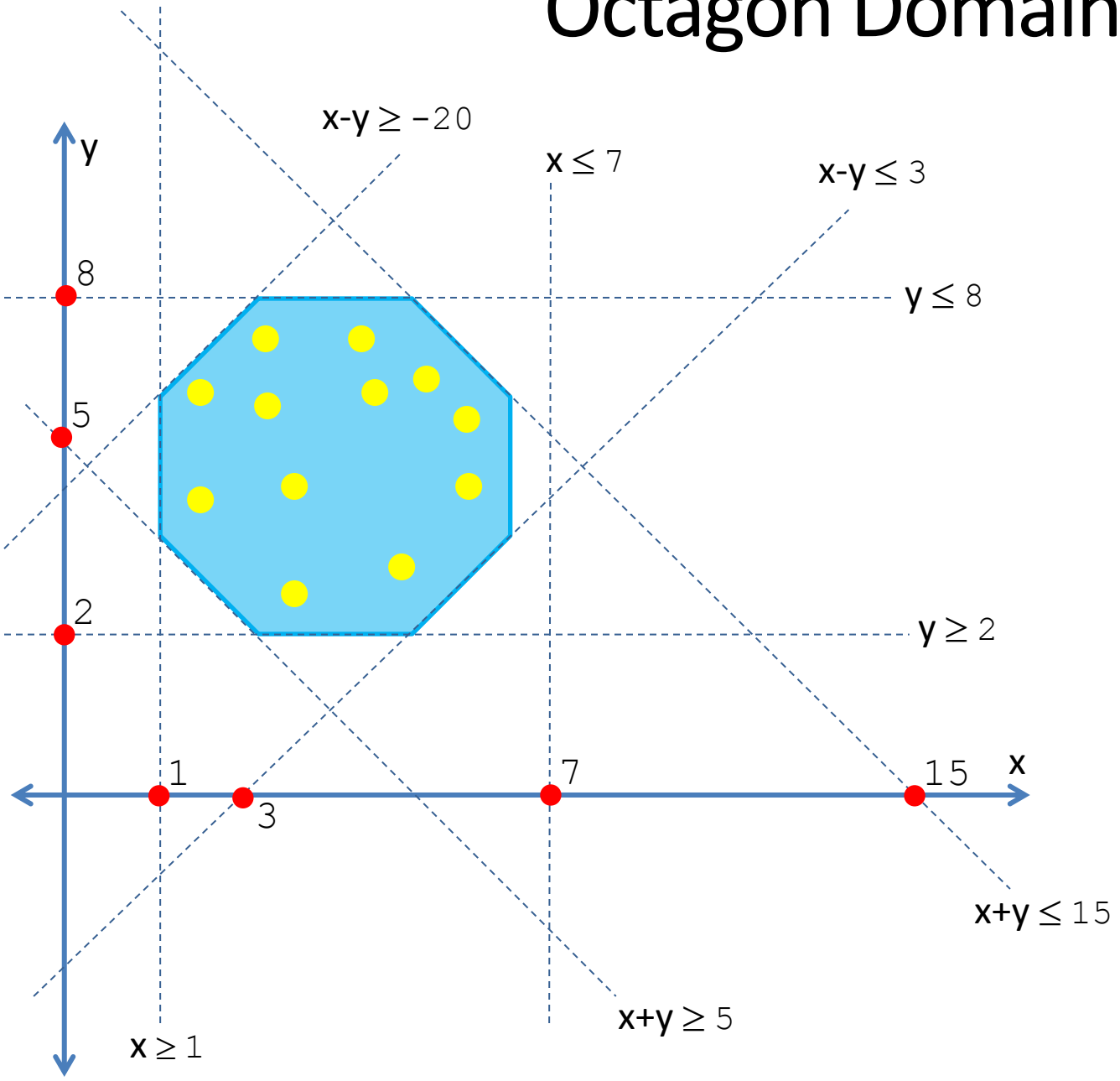
Relational Abstractions

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

In some cases however, it may be necessary to keep this relationship in order to be more precise. Next, we show two examples of abstractions (Octagon and Polyhedra) where the relationship is kept. These domains are called relational domains.

In the project, you will use the Polyhedra domain, already implemented as part of the Apron library.

Octagon Domain



constraints are of the following form:

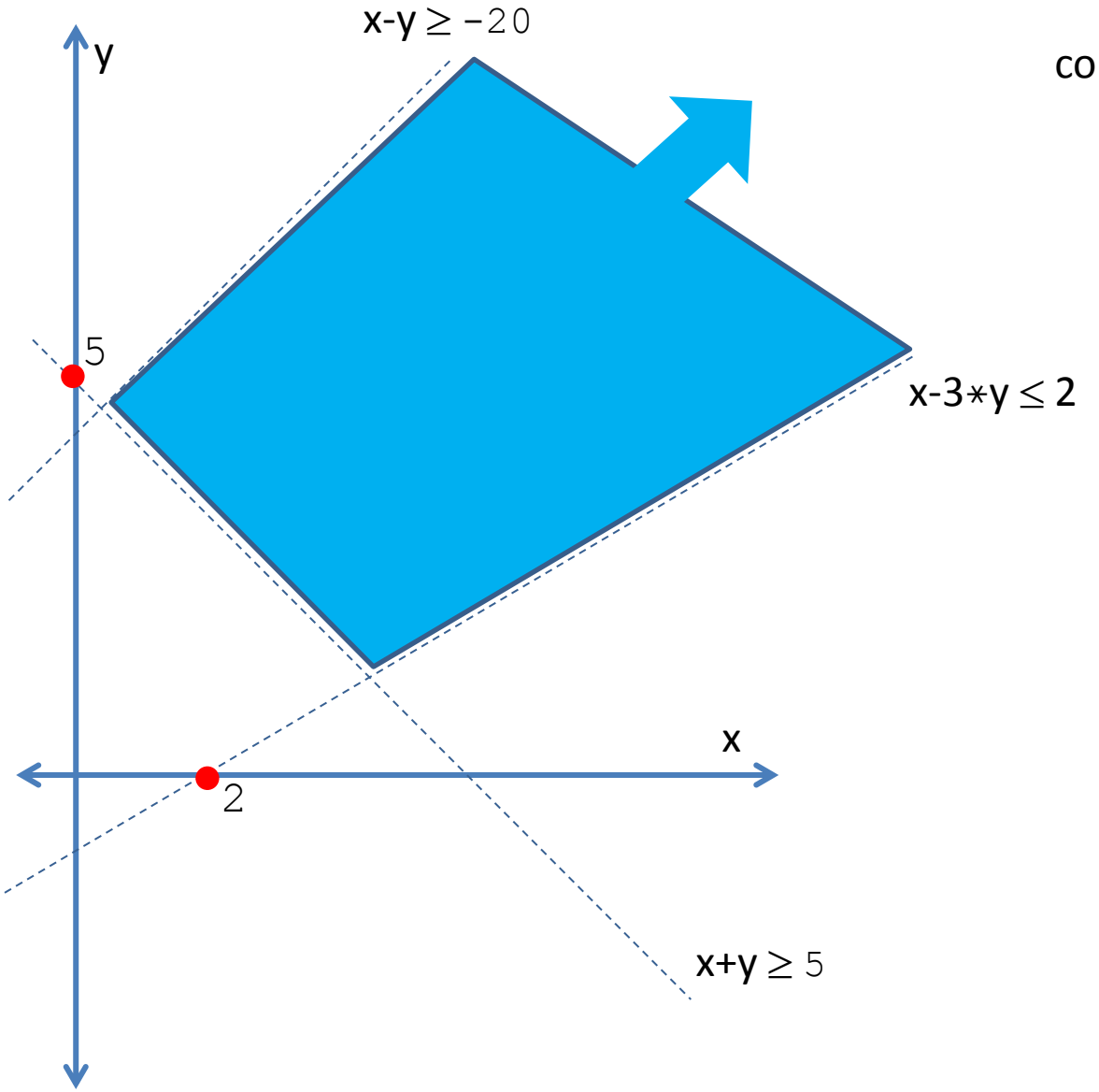
$$\pm x \pm y \leq c$$

The slope is fixed

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

$$\begin{array}{ll} x - y \leq 3 & \wedge \\ y \leq 8 & \wedge \\ y \geq 2 & \wedge \\ x + y \leq 15 & \wedge \\ x + y \geq 5 & \wedge \\ x \geq 1 & \wedge \\ x - y \geq -20 & \wedge \\ x \leq 7 & \end{array}$$

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:

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

Project

Discussion about the project