Abstract Interpretation

Abstract Interpretation

- Designed to describe static analyses of imperative programs and to prove their correctness
- Since then, applied to numerous classes of programming languages and software/hardware systems
- Today, viewed as a general technique for reasoning on semantics at various abstraction levels

The general idea

- The starting point is a concrete semantics that provides the meaning of program commands into a given computational domain
- An abstract domain, which models some properties of interest of concrete computations and leaves out the remaining information
- An abstract semantics that allows us "to abstractly execute" a program on the abstract domain in order to compute the program properties modelled by the abstract domain.

Abstract Interpretation

It is a technique to formally reason on approximations

It allows to derive effective methods to compute approximations

Generally used to compute overapproximations

Seldom used to compute underapproximations

Example: out of bounds

```
function arrayOutOfBounds(int n, int x[10]) {

  a = 0

  if n >= 10 then

     n = n - 5

  else

     a = ++n


  a = max(0,a — n)

  return x[a] }
```

Let us assume $n \geq 0$

Is it a safe access? ($0 \leq a \leq 9$?)

Using exact semantics

```
function arrayOutOfBounds(int n, int x[10]) {
(0,_)(1,_)(2,_)(3,_)(4,_)(5,_)(6,_)(7,_)(8,_)(9,_)(10,_)….
  a = 0
(0,0)(1,0)(2,0)(3,0)(4,0)(5,0)(6,0)(7,0)(8,0)(9,0)(10,0)….
  if n >= 10 then
(10,0)(11,0)(12,0)(13,0)(14,0)(15,0)(16,0)(17,0)(18,0)(19,0)….
     n = n - 5
(5,0)(6,0)(7,0)(8,0)(9,0)(10,0)(11,0)(12,0)(13,0)(14,0)….
  else

     a = ++n


  a = max(0,a – n)

  return x[a] }
```

We can't track the infinite set of pairs!

use intervals !

Example: interval abstraction

```
 function arrayOutOfBounds(int n, int x[10]) {
[0,∞]
   a = 0
[0,∞][0,0]
   if n >= 10 then
    [10,∞][0,0]
      n = n - 5
    [5,∞][0,0]
   else
    [0,9][0,0]
      a = ++n
    [1,10][1,10]
[1,∞][0,10]
   a = max(0,a – n)
[1,∞][0,9]
   return x[a] }
```

Merging branches looses precision

safe! $0 \leq a \leq 9$!

Abstract Interpretation: the idea

Goal: Compute the set S of possible values at each line of code

But… this is not feasible in general

We want to find an (over)approximation $S \subseteq S^{\#}$

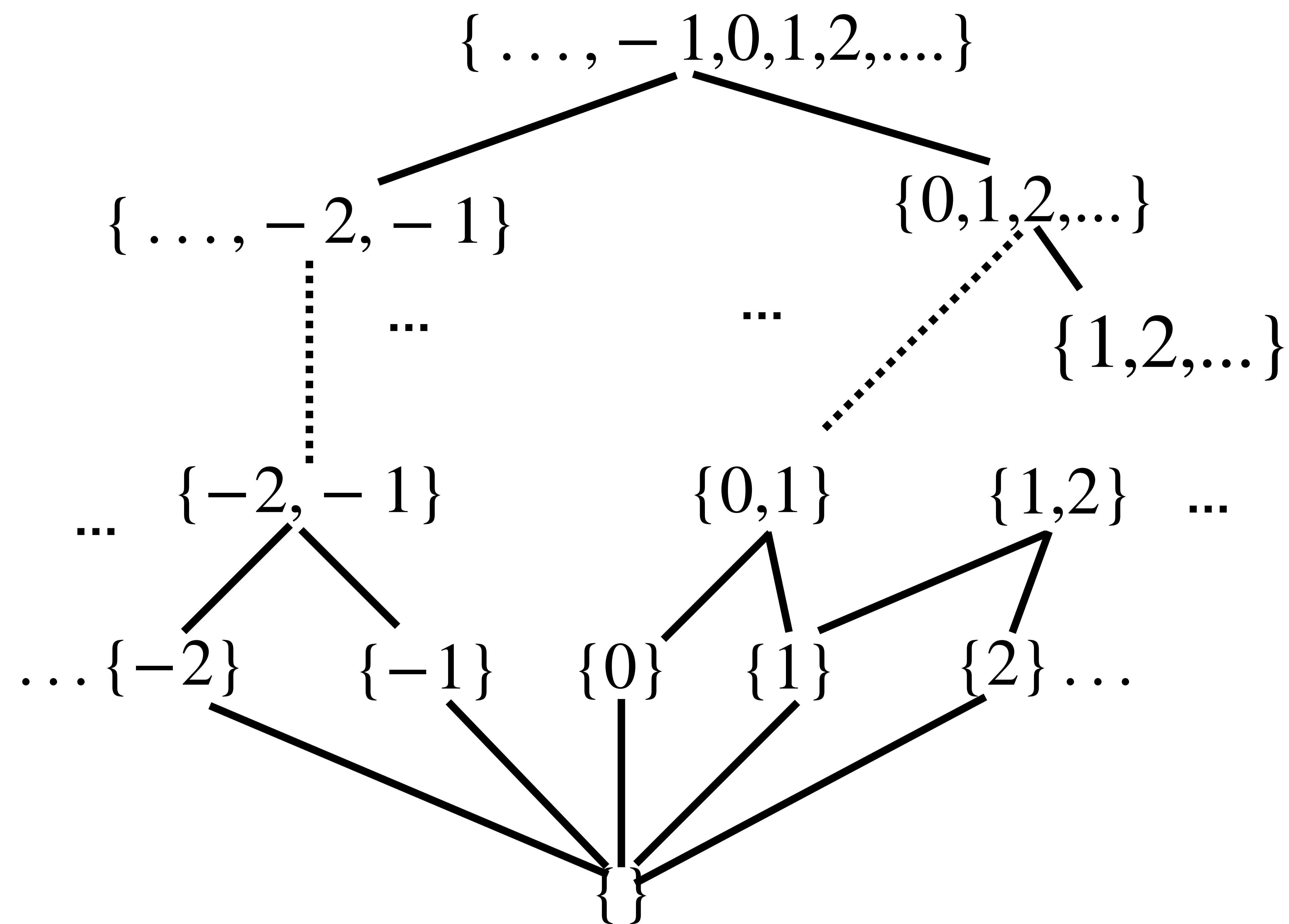
The theory of abstract interpretation allows to compute $S^{\#}$ as a set of abstract values obtained by applying abstract operations

Abstraction and concretization

Concrete domain

The set of values S that we would like to compute belongs to the concrete domain C

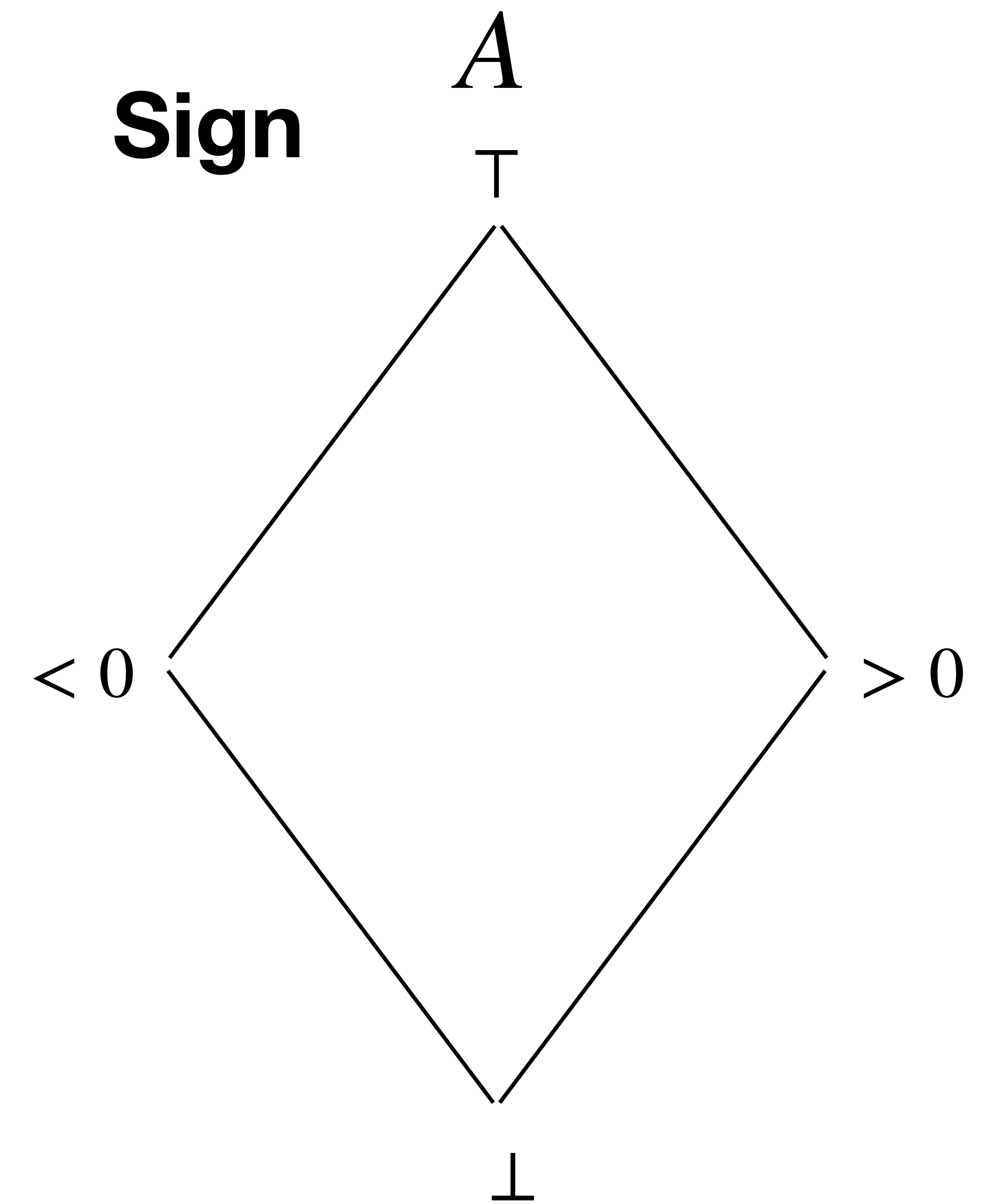
$$(\wp(\mathbb{Z}), \subseteq)$$



Abstract Domain

$(A, \sqsubseteq)$ expresses some properties of the concrete values

For example

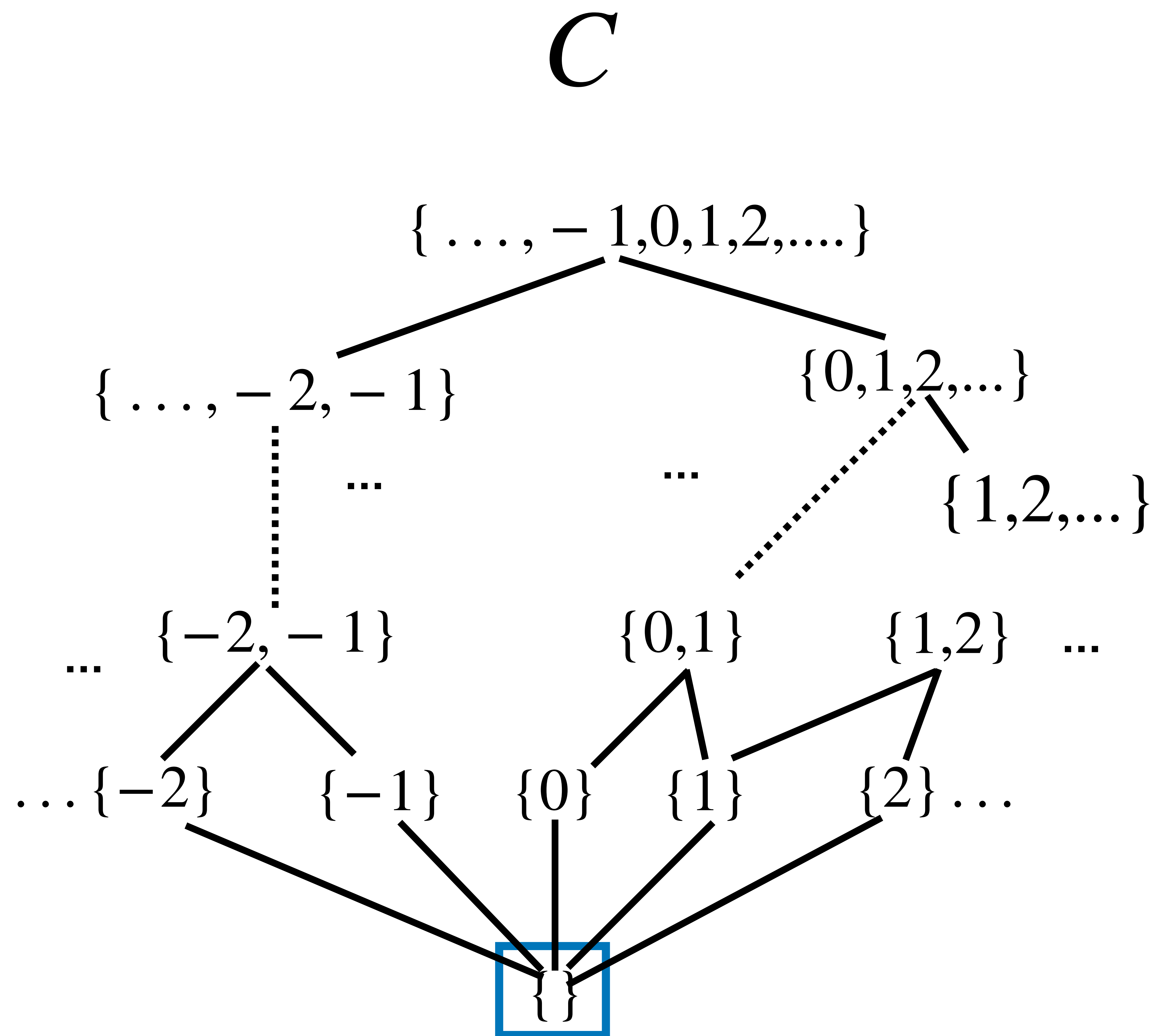


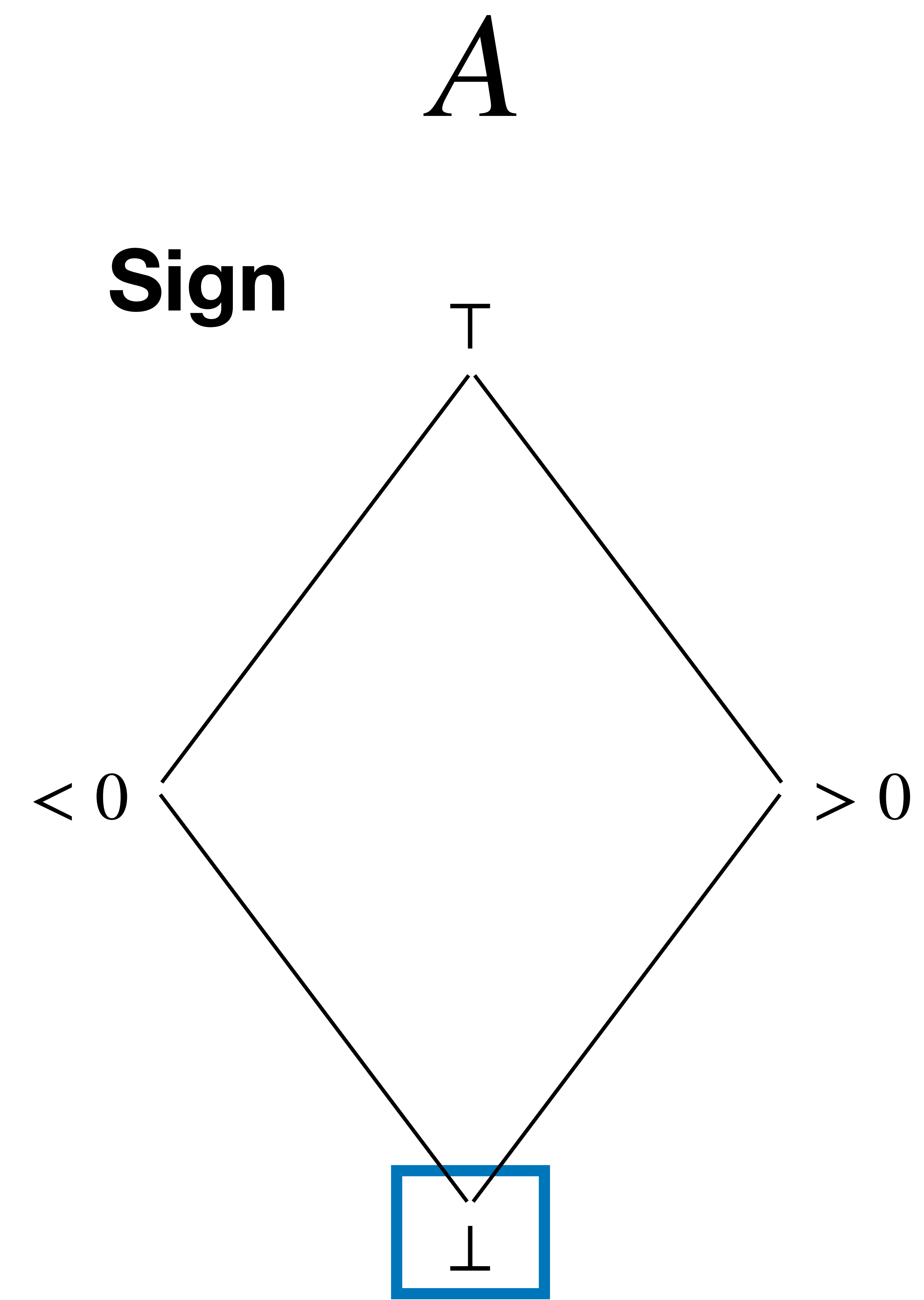
The order $\sqsubseteq$ on the abstract domain reflects the precision

Ingredients of Abstract Interpretation

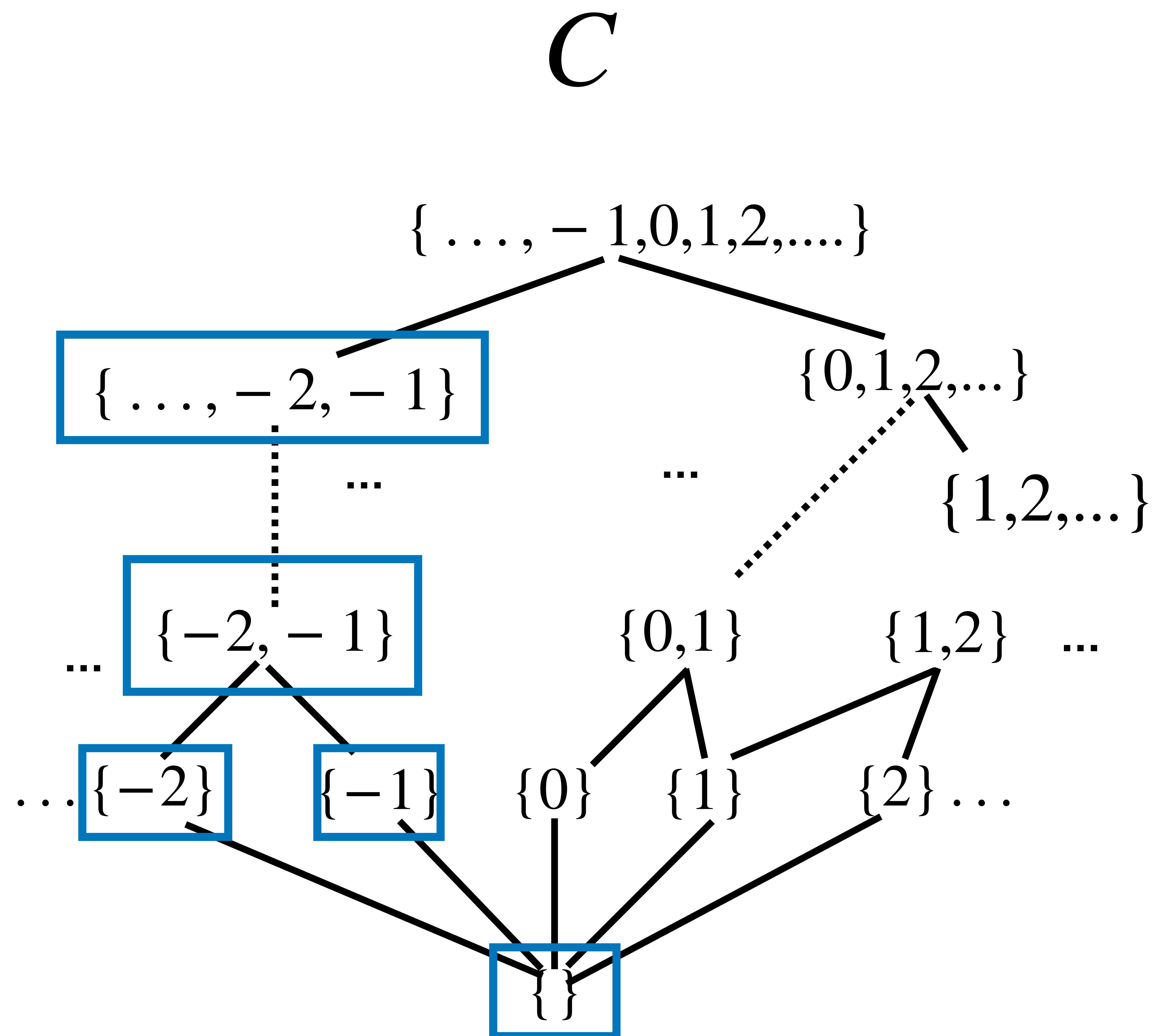
- A concrete domain C
- An abstract domain A
- An abstraction function α that connects the concrete domain to the abstract one
- A concretisation function γ that relates the abstract domain to the concrete one

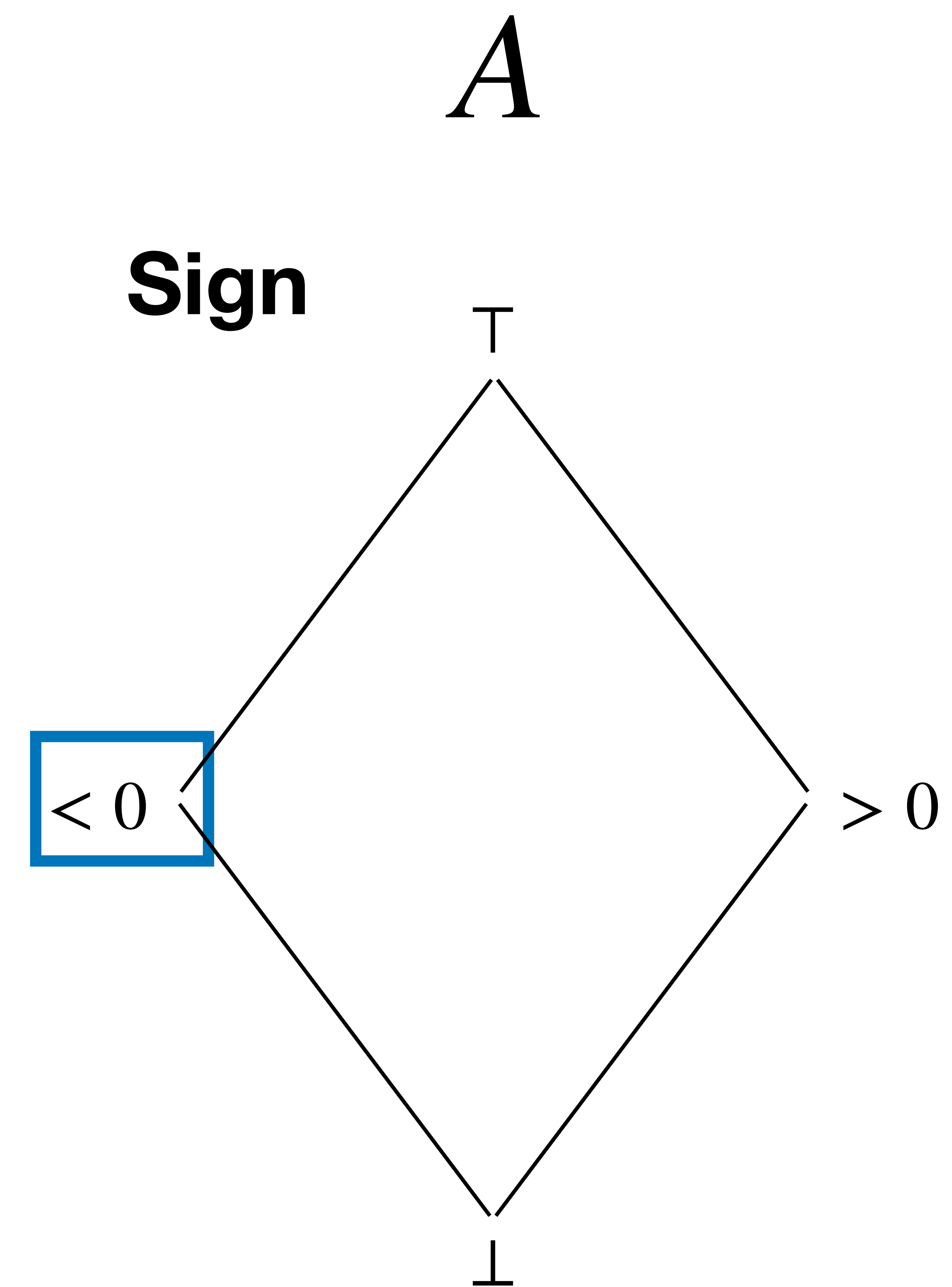
Defining approximation



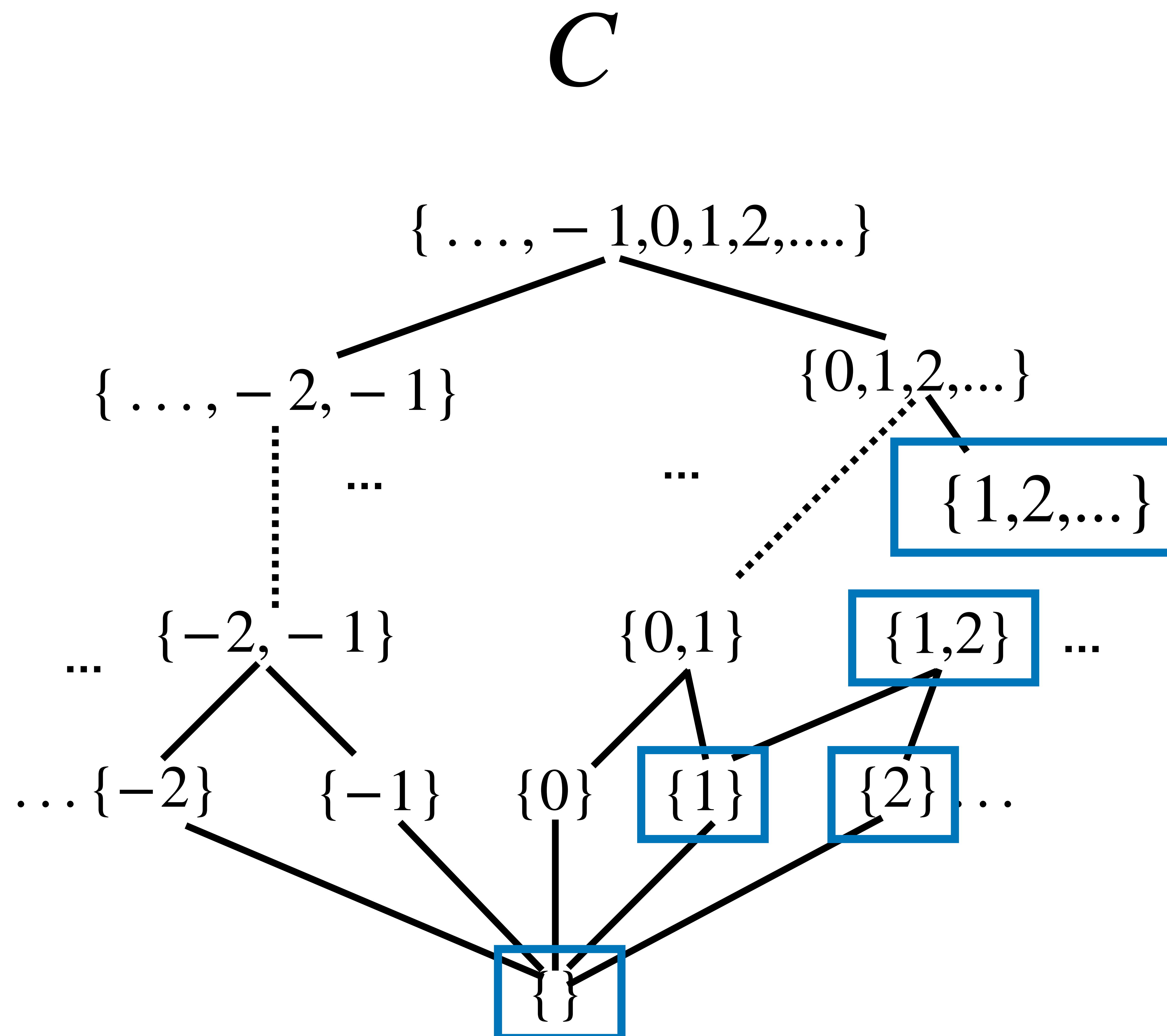


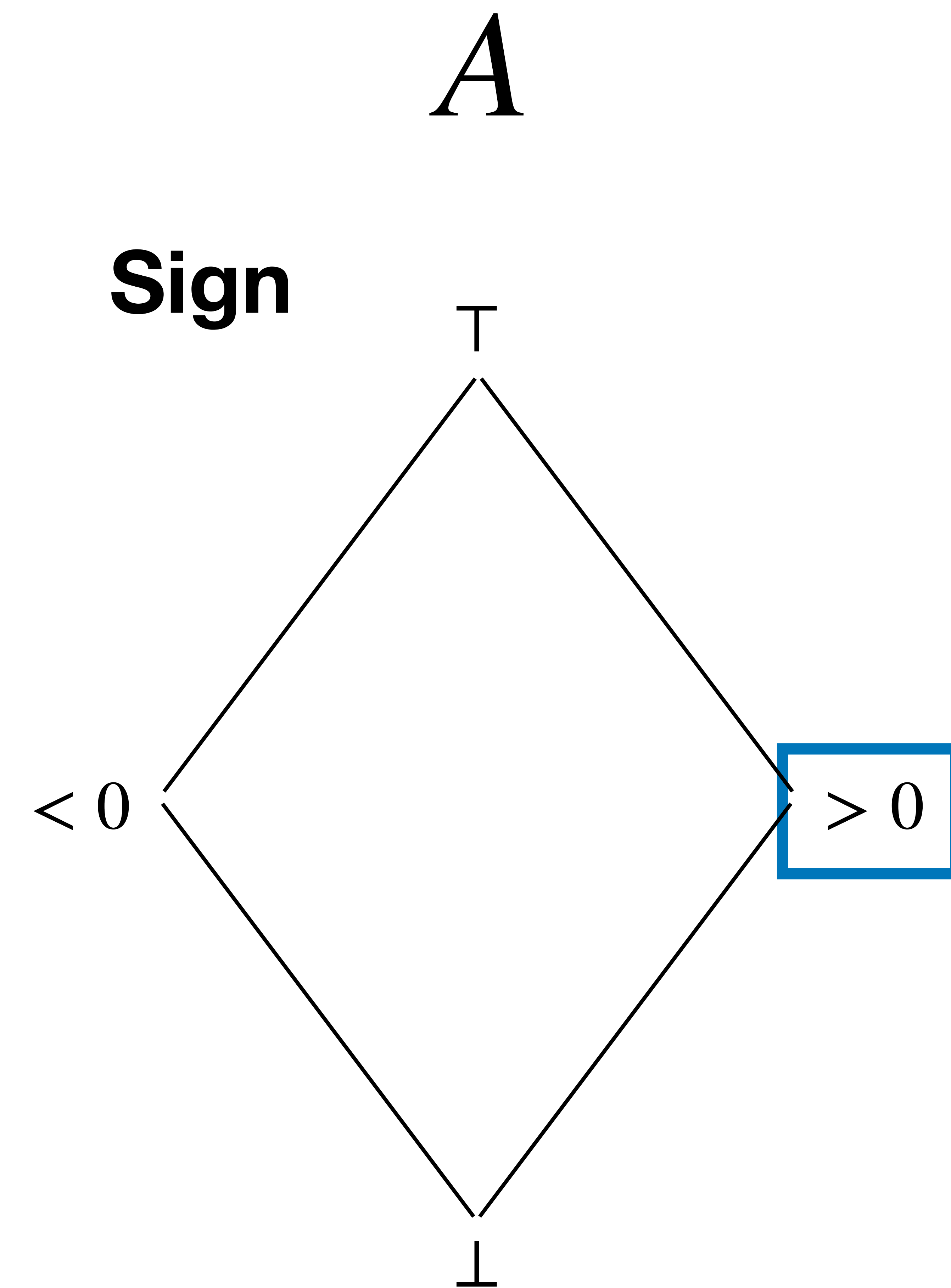
Defining approximation



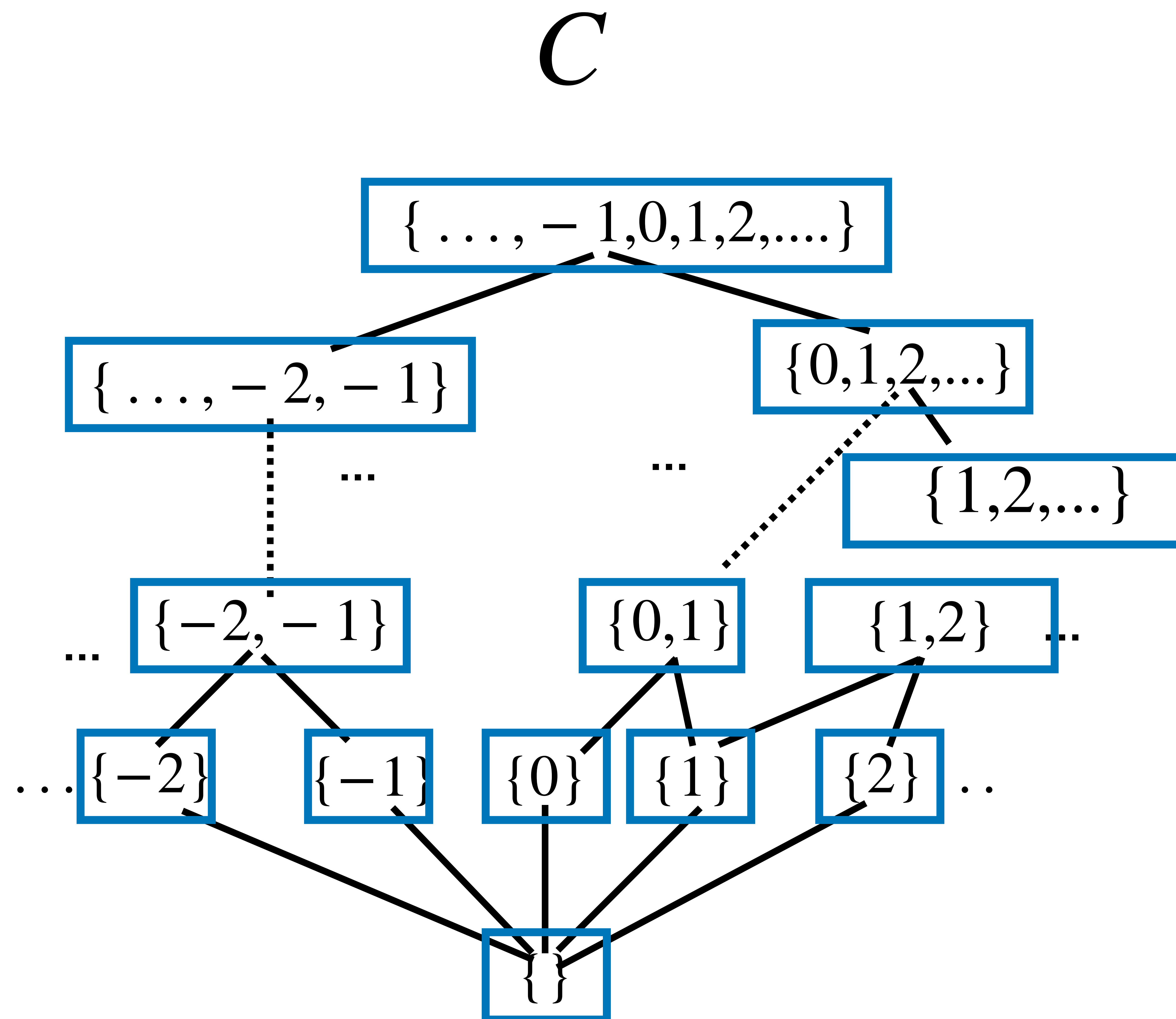


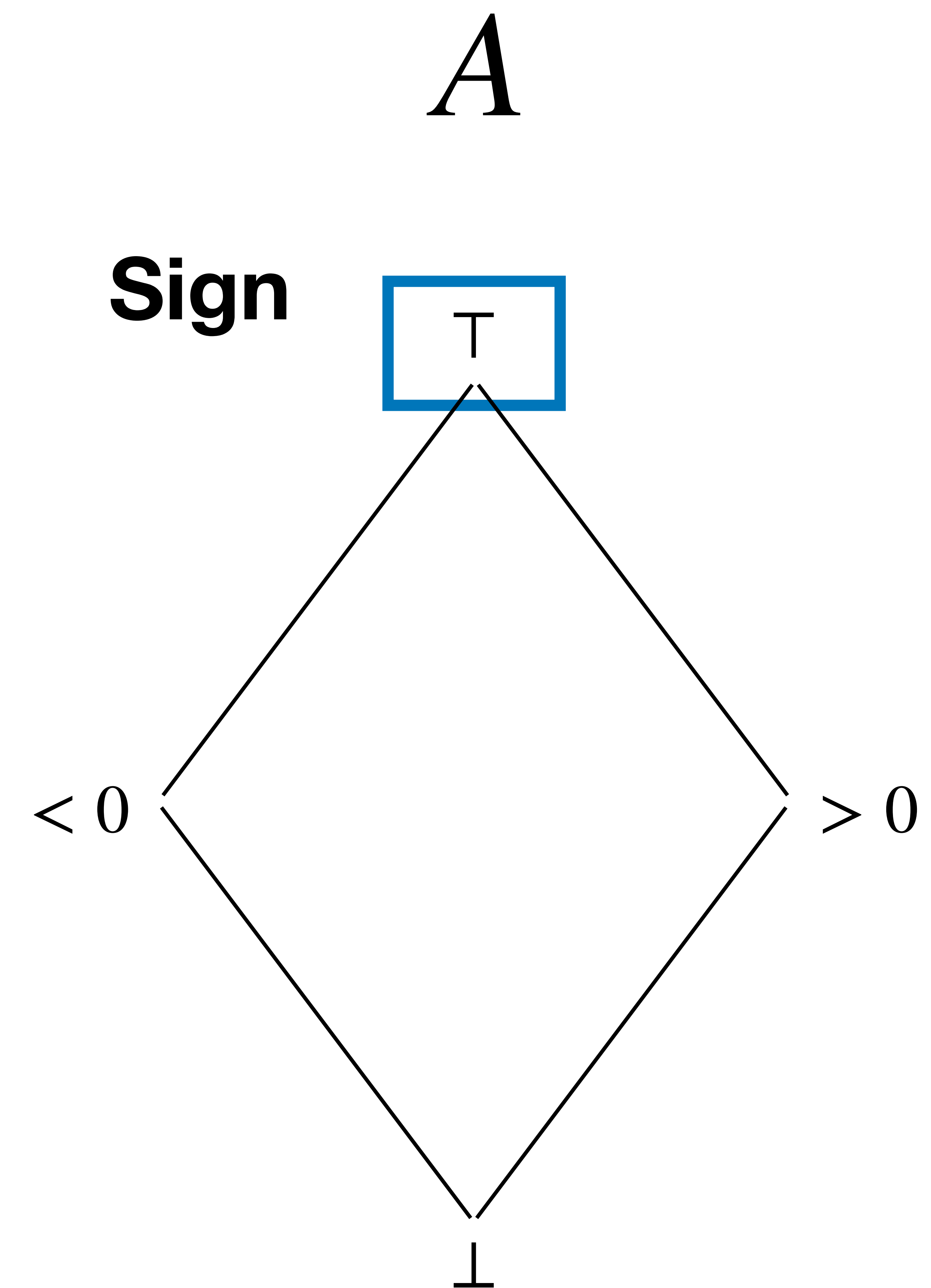
Defining approximation





Defining approximation

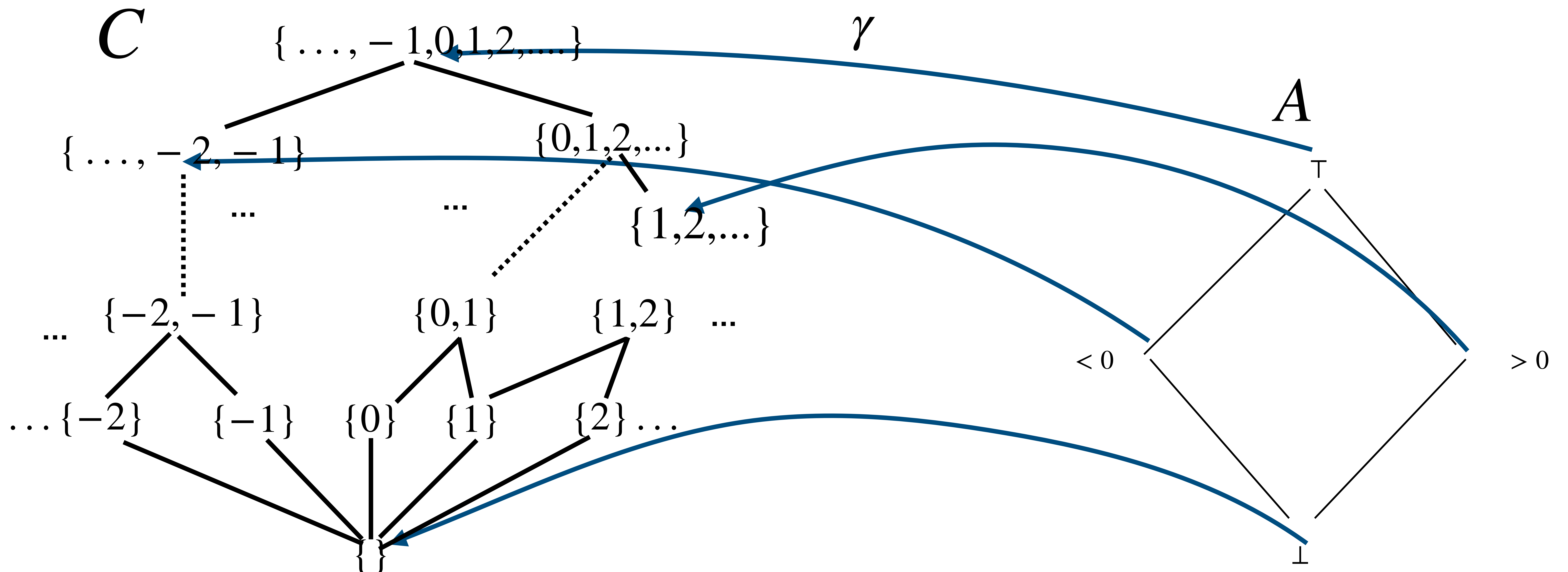




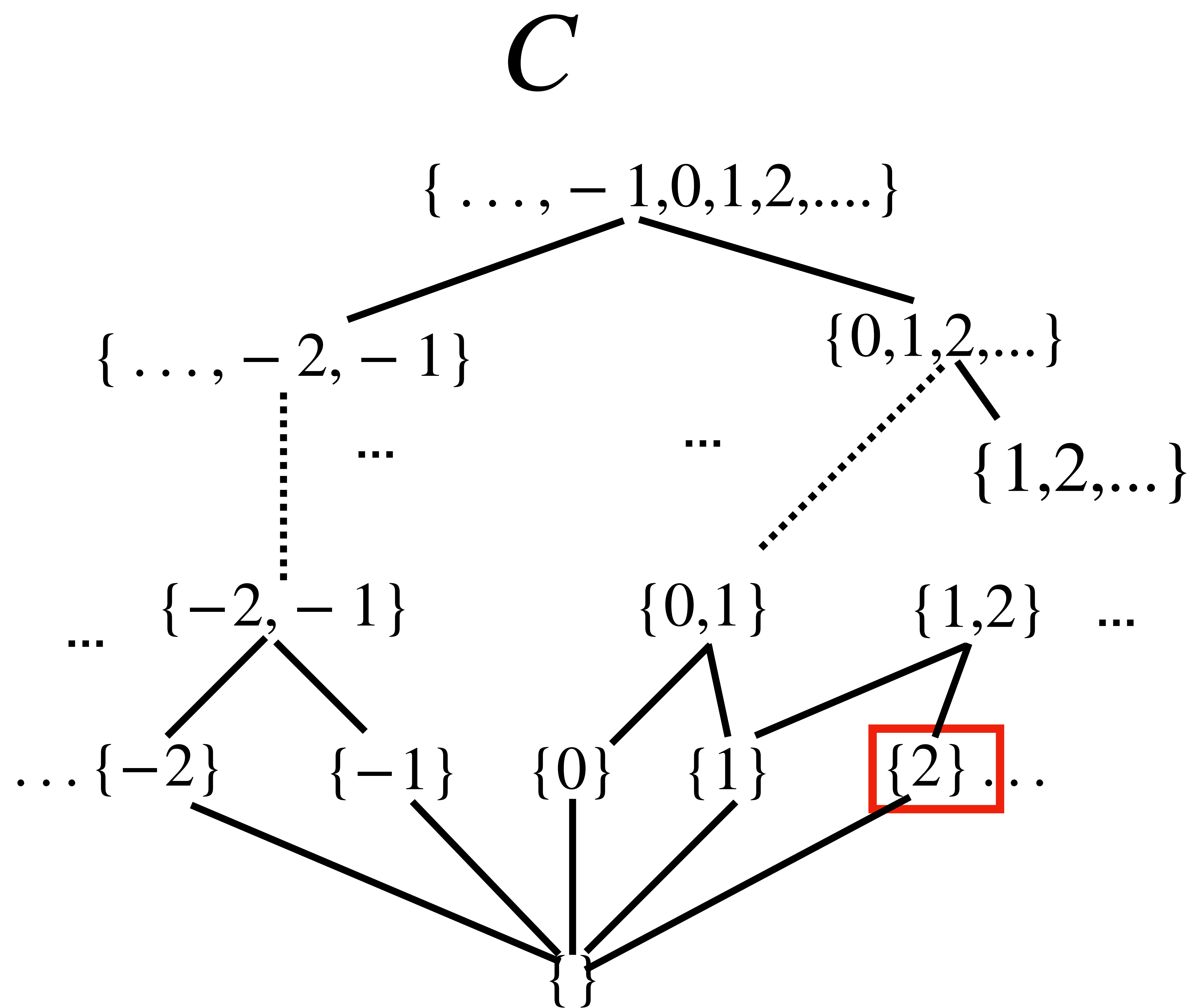
Concretization function

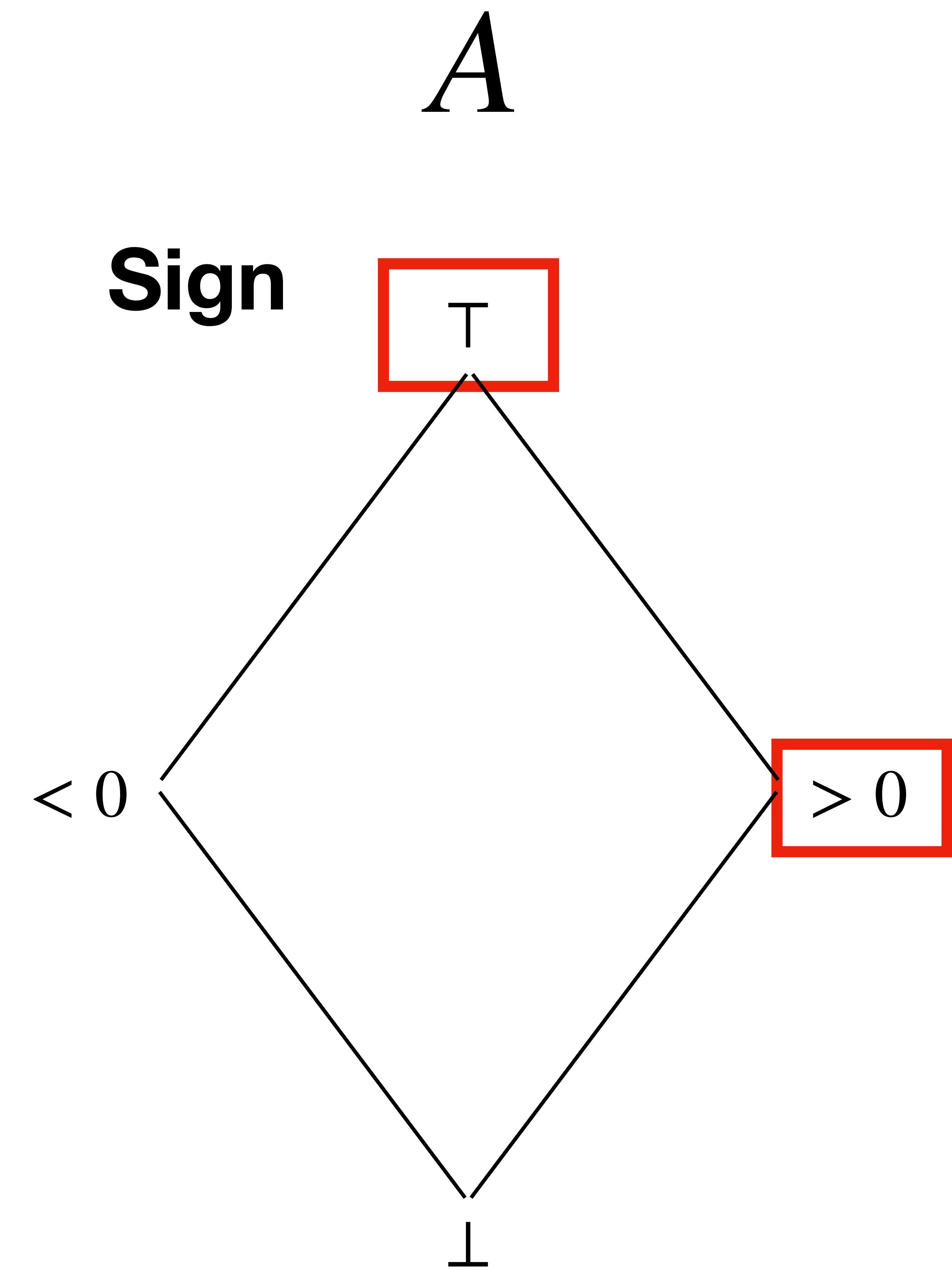
Definition

Concretization function $\gamma : A \rightarrow C$ is a monotone function
that maps abstract a into the greatest concrete c that it approximates

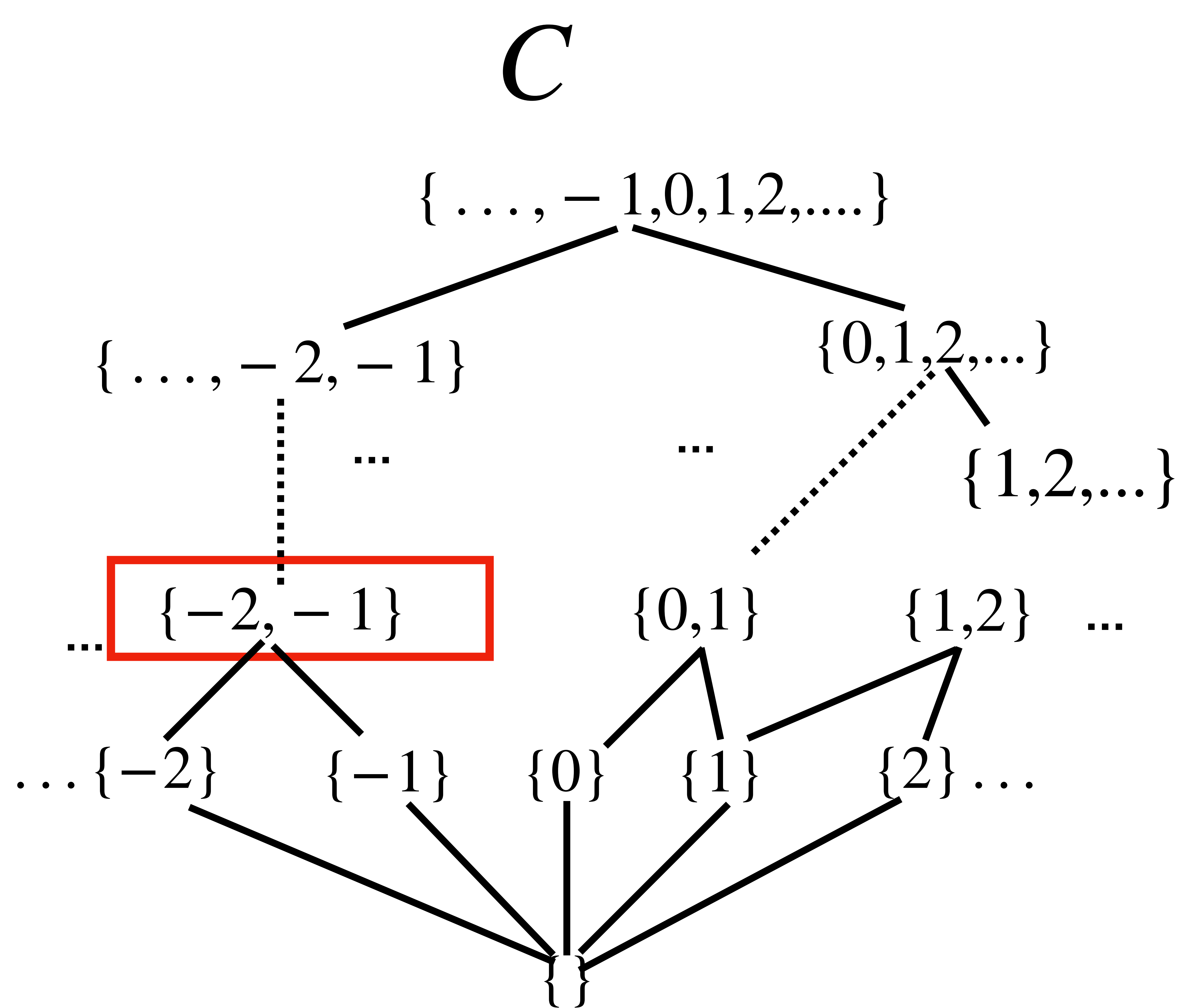


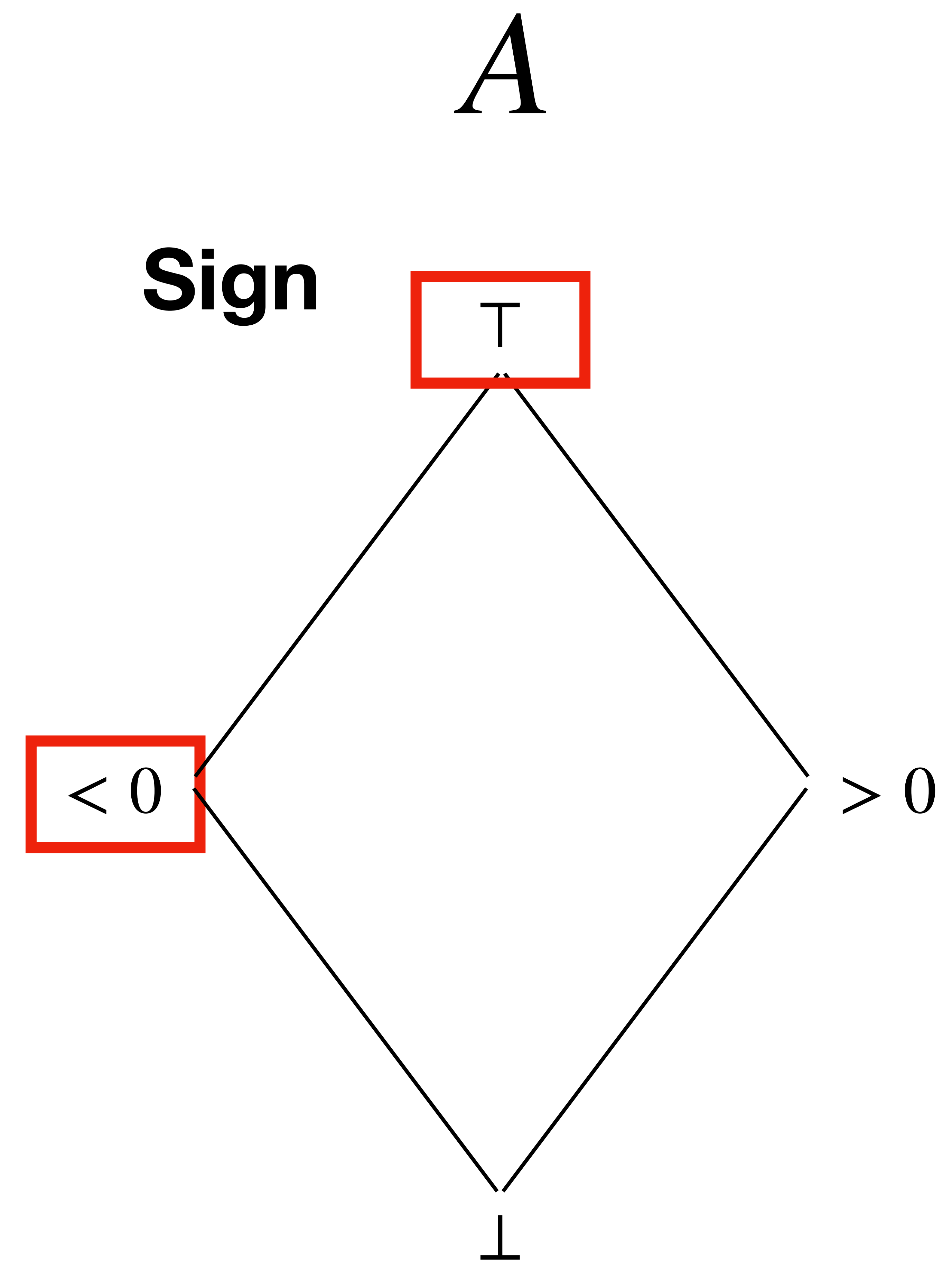
Defining approximation



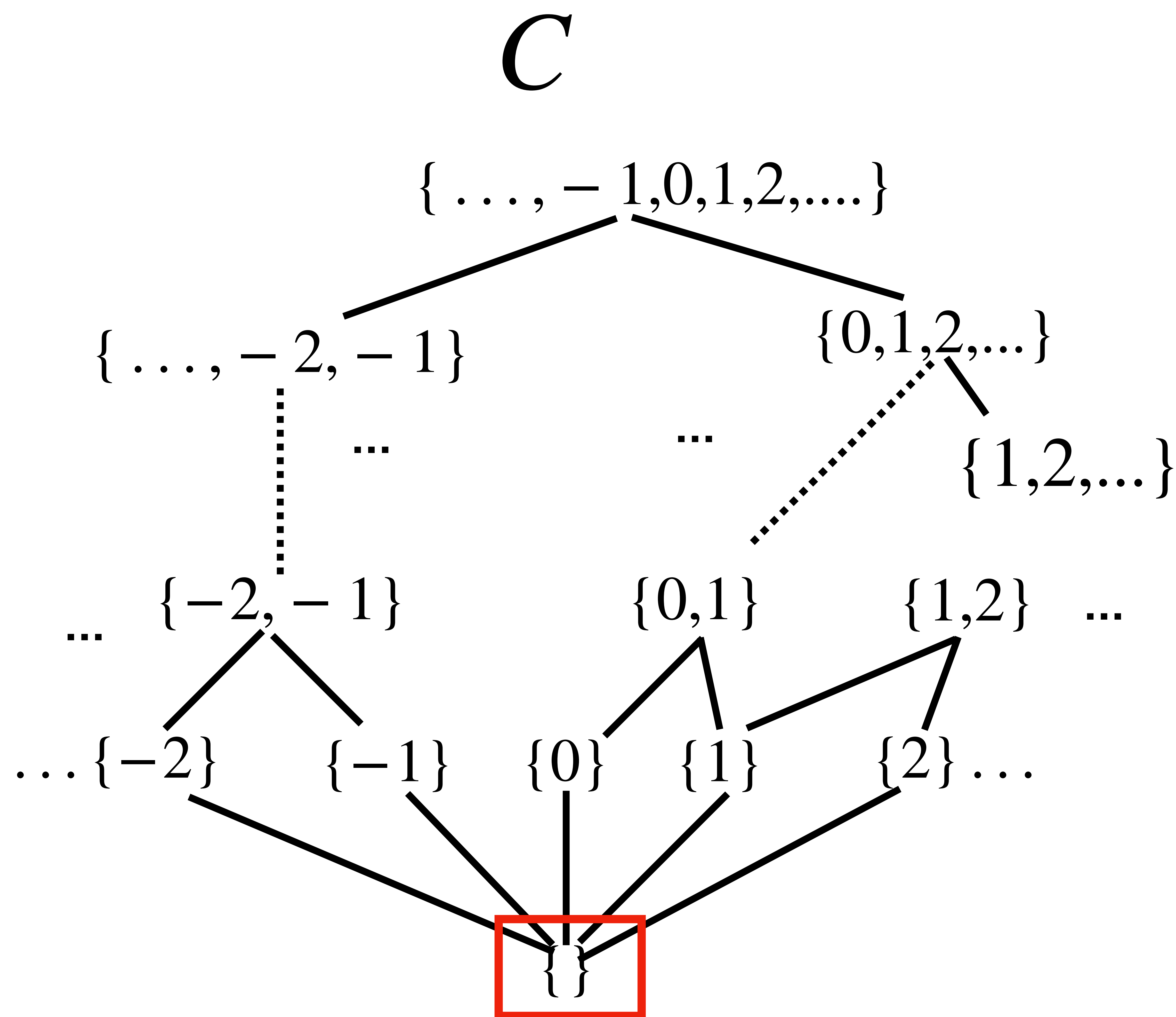


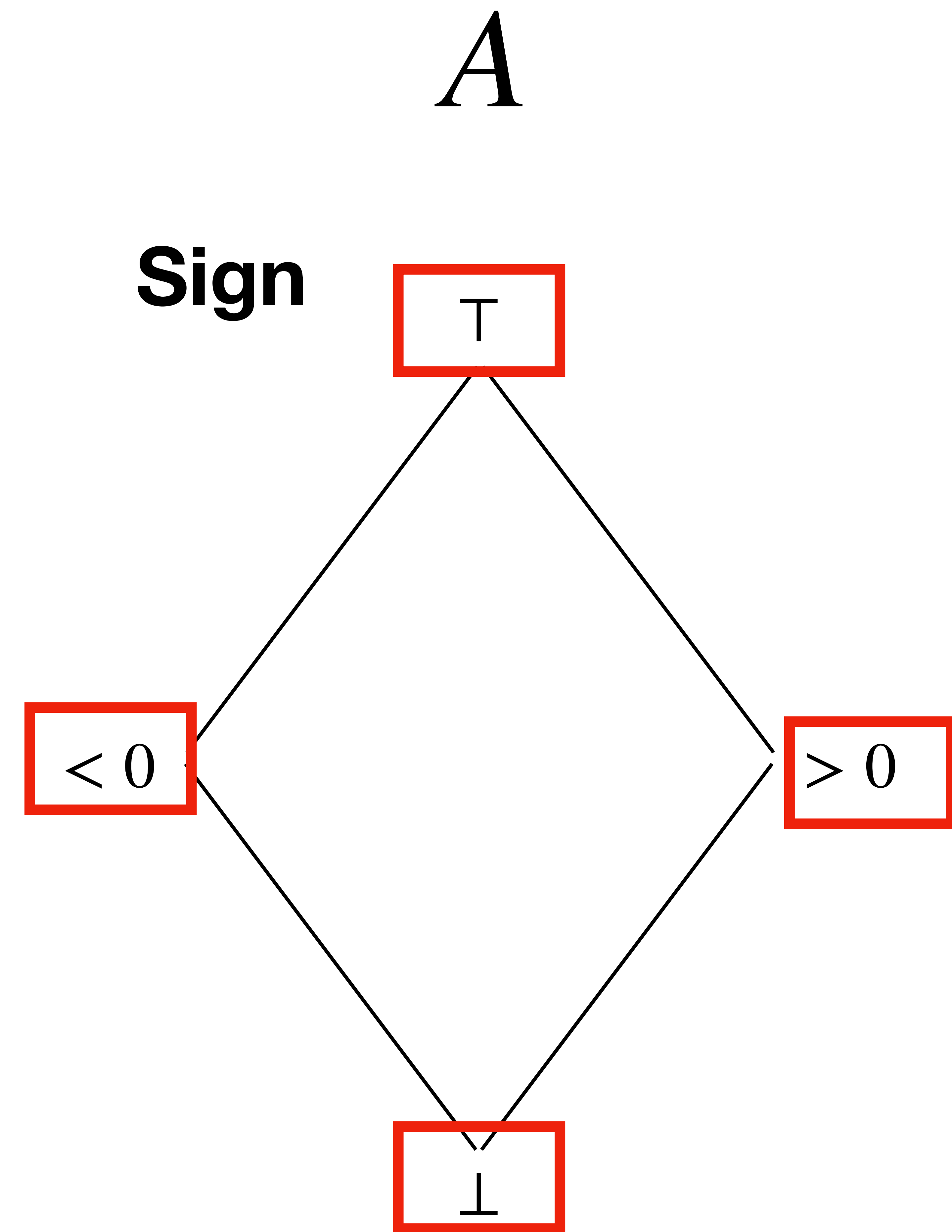
Defining approximation



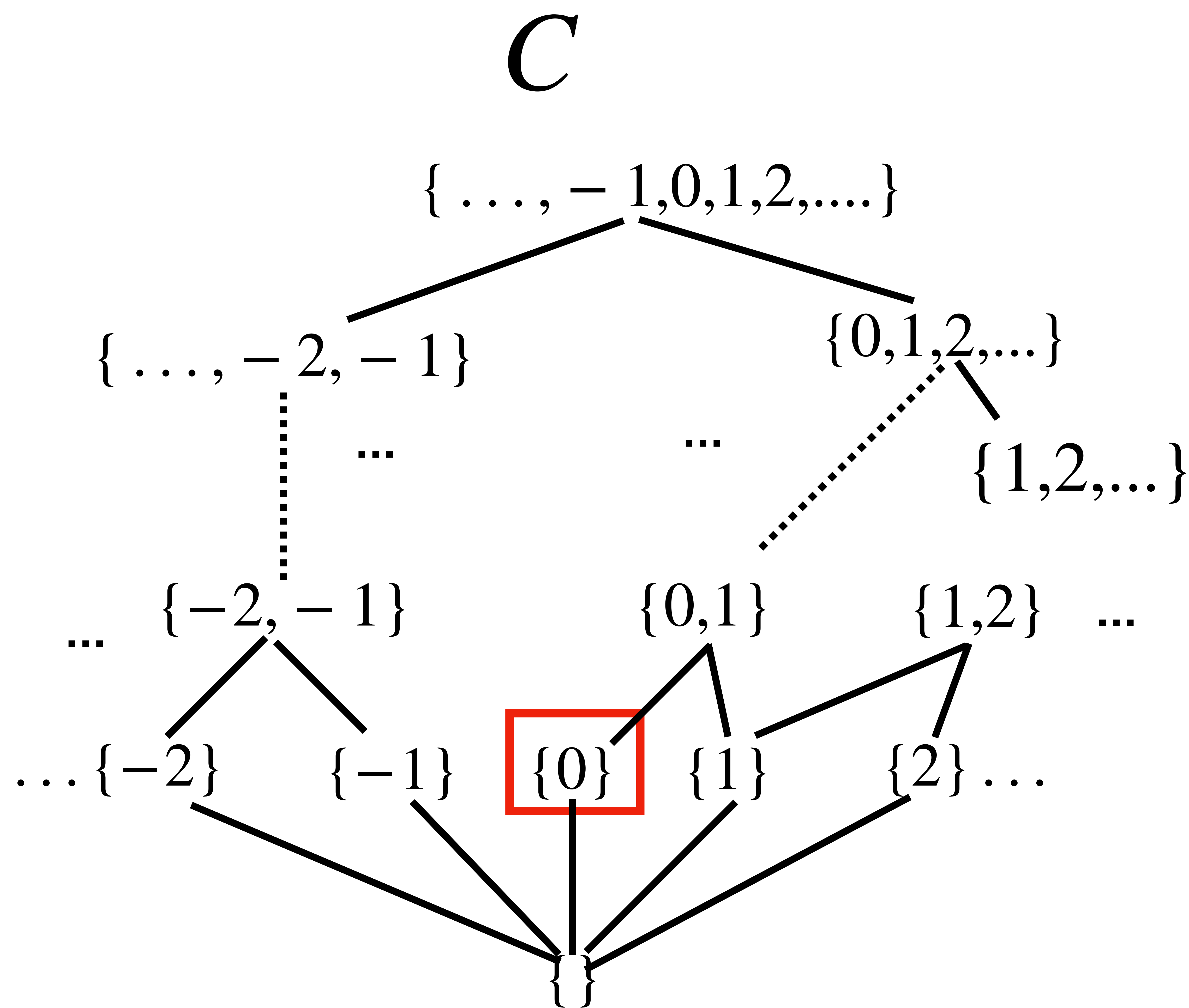


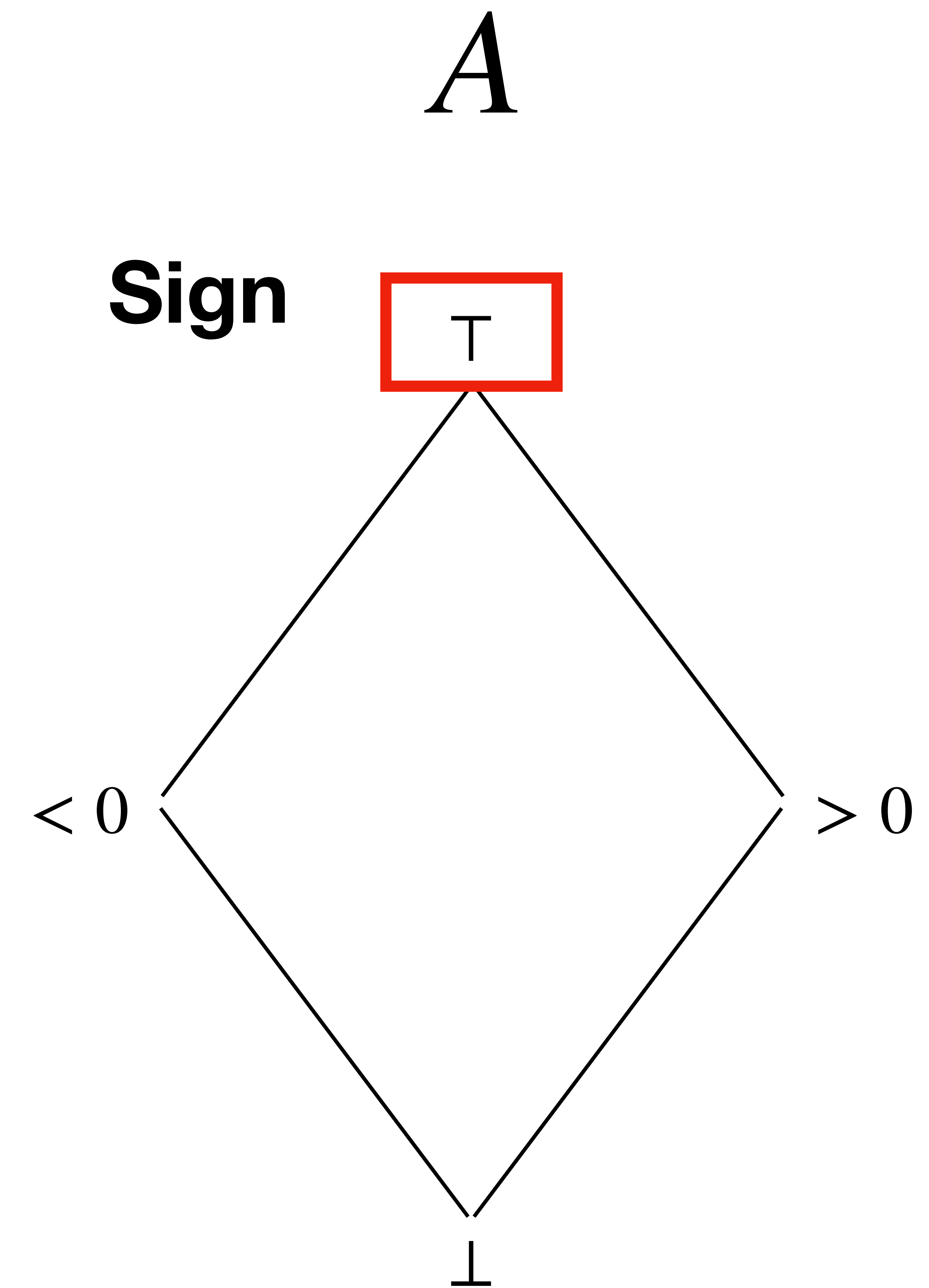
Defining approximation





Defining approximation

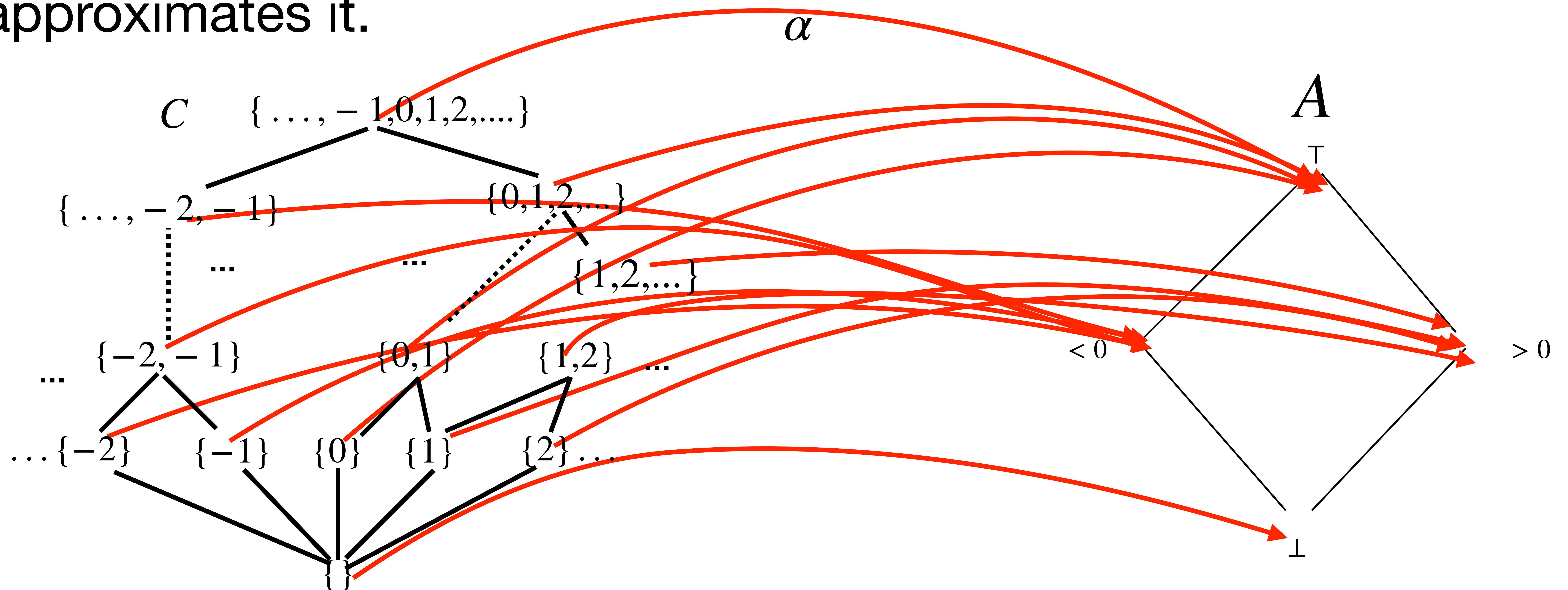




Abstraction function

Definition

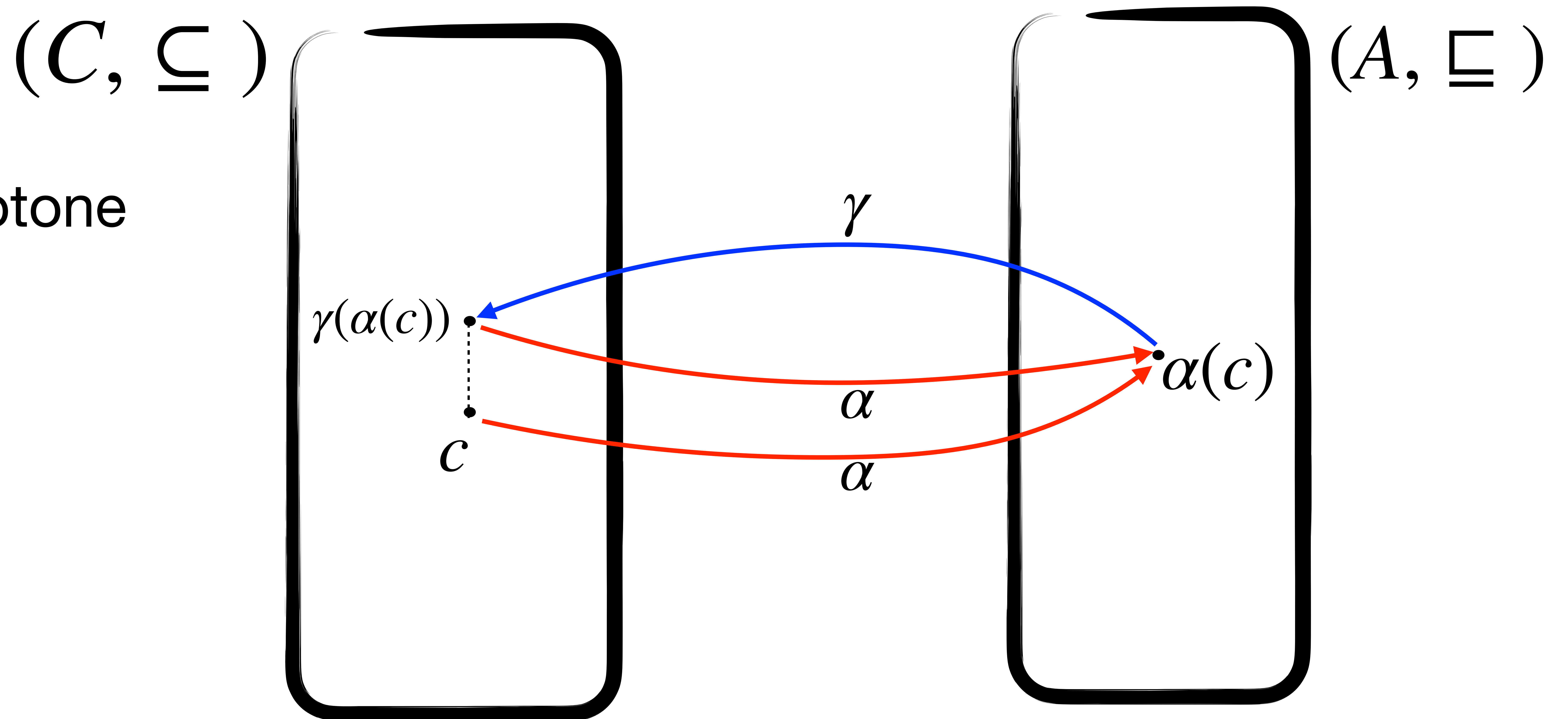
Abstraction function $\alpha : C \rightarrow A$ is a monotone function that maps concrete c into the most precise abstract element that approximates it.



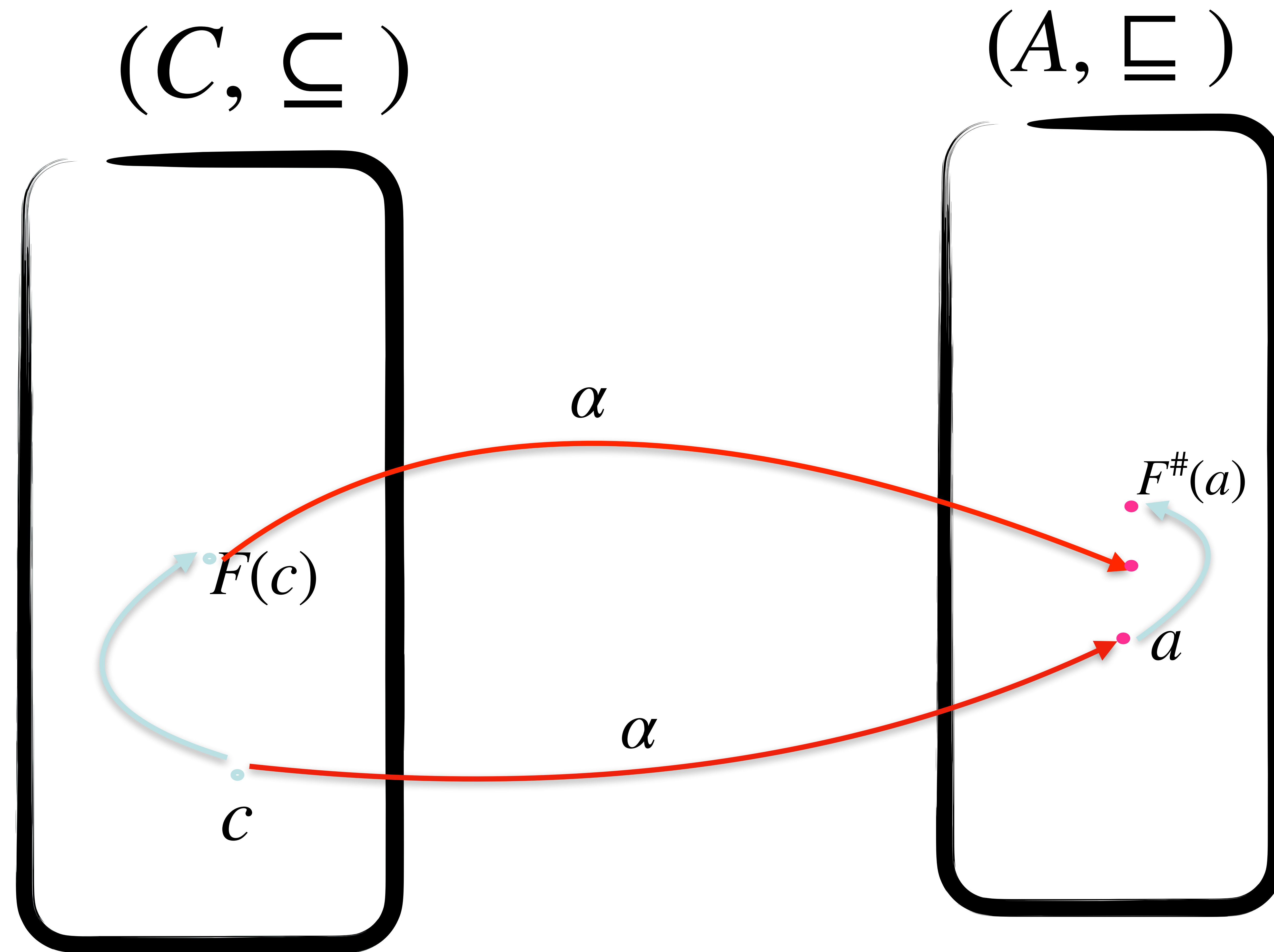
Abstract Interpretation
(AI)

Properties of Galois insertions

- α and γ are monotone
- $c \subseteq \gamma(\alpha(c))$
- $\alpha(\gamma(a)) = a$



Correct approximation of functions

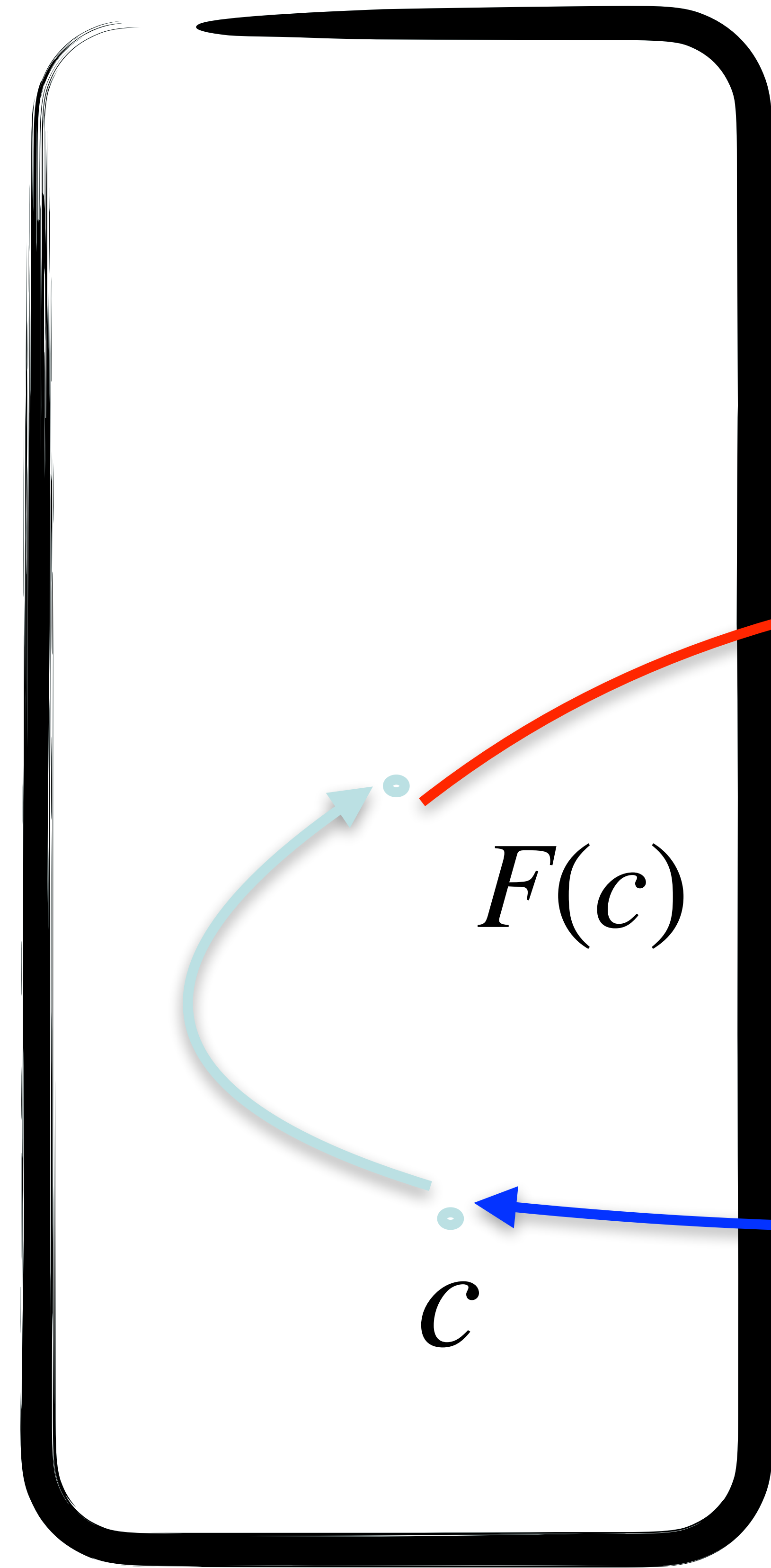


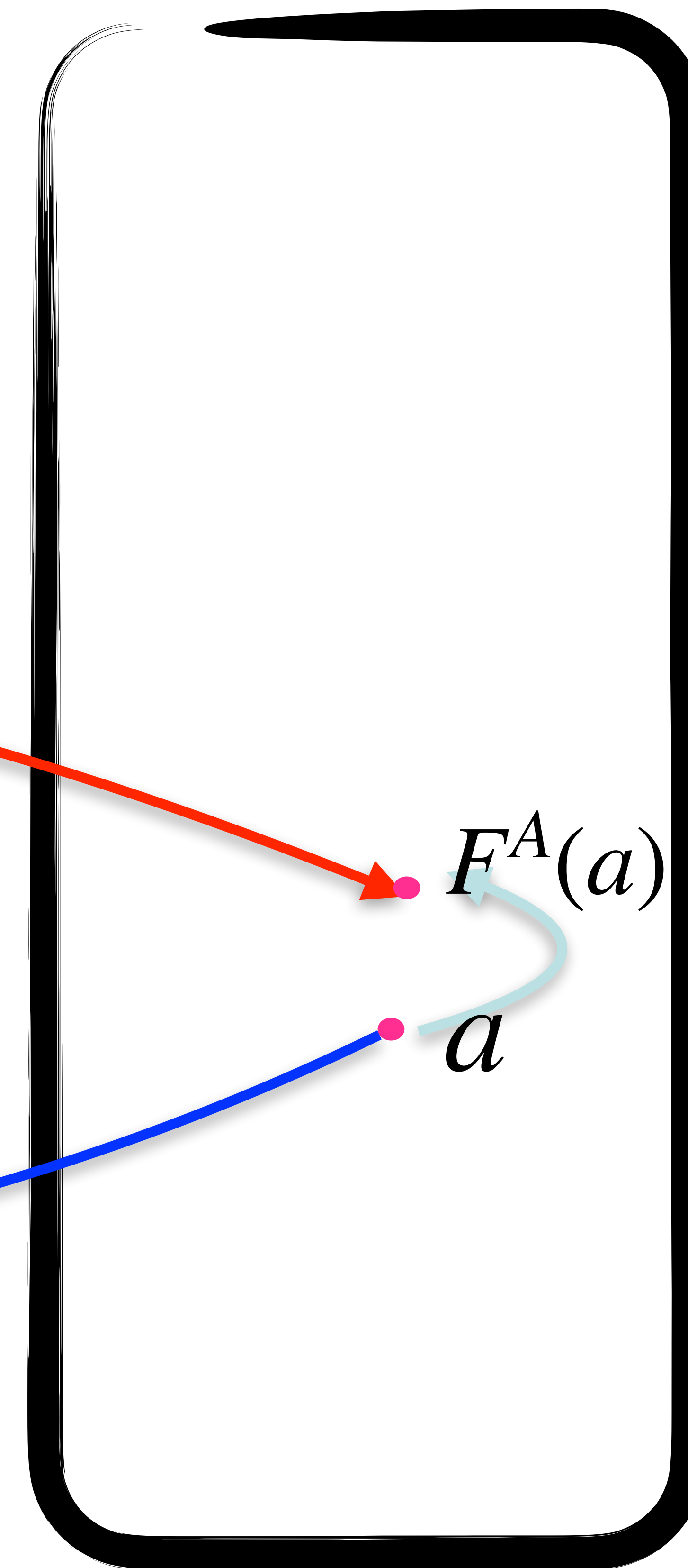
$F^{\#}\alpha \sqsupseteq \alpha F$

Best correct approximation (bca)

$(C, \subseteq)$

$(A, \sqsubseteq)$





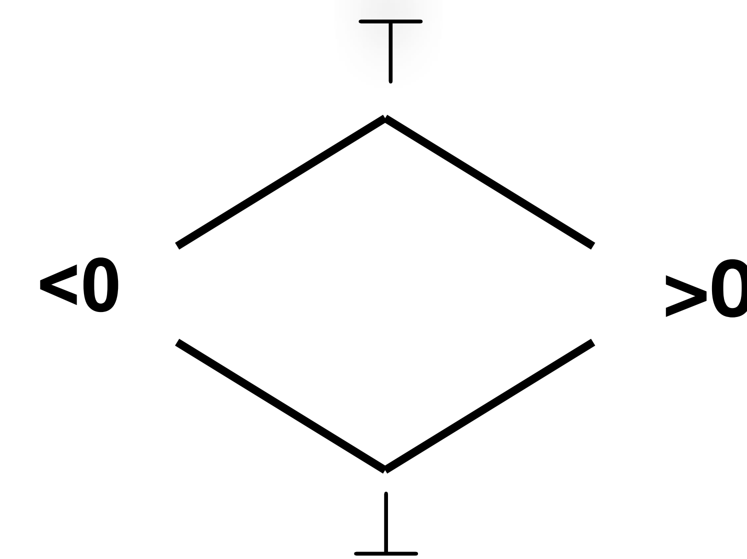
$F^A \triangleq \alpha F \gamma$

Abstract operations: +

$$(\wp(\mathbb{Z}), \subseteq)$$

$$+ : \wp(Z) \rightarrow \wp(Z)$$

$$\{3,5\} + \{-2,4\} = \{1,7,3,9\}$$



$+^\#$	⊥	<0	>0	T
⊥	⊥	⊥	⊥	⊥
<0	⊥	<0	T	T
>0	⊥	T	>0	T
T	⊥	T	T	T

$\{1,2,...\}$ $\{\ldots,-2,-1\}$ $\top$

> 0 < 0 > 0

$$\{3\} + \{-2\} = \{1\}$$

We lost precision

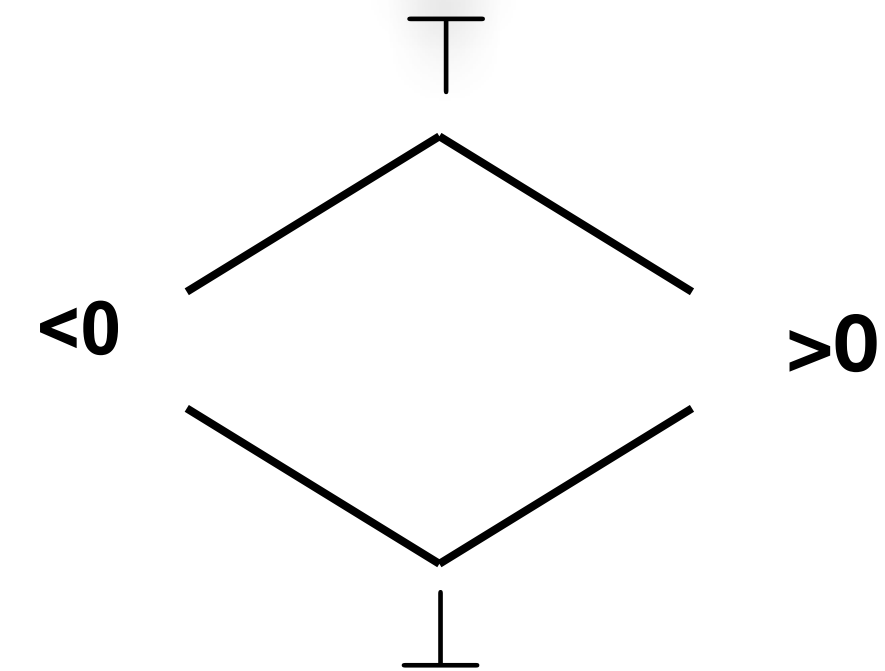
> 0 > 0 > 0

$\{1,2,...\}$ $\{1,2,...\}$ $\{2,3,...\}$

$$\{3\} + \{2\} = \{5\}$$

Precise result!

Abstract operations: $\times$



$(\wp(\mathbb{Z}), \subseteq)$

$\times : \wp(Z) \rightarrow \wp(Z)$

$\{3,5\} \times \{-2,4\} = \{-6, 12, -10, 20\}$

>0 <0 <0

$\{1,2,...\}$ $\{\ldots,-2,-1\}$ $\{\ldots,-2,-1\}$

$\{3\} \times \{-2\} = \{-6\}$

Precise result!

>0 >0 >0

$\{1,2,...\}$ $\{1,2,...\}$ $\{1,2,...\}$

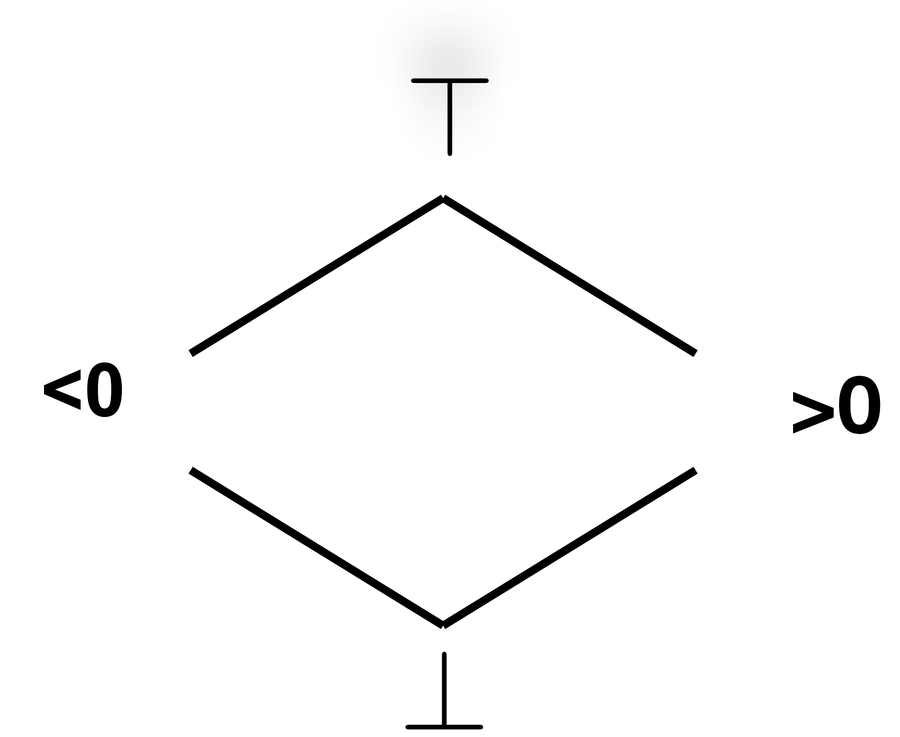
$\{3\} \times \{2\} = \{6\}$

Precise result!

$\times^{\#}$	⊥	<0	>0	⊤
⊥	⊥	⊥	⊥	⊥
<0	⊥	>0	<0	⊤
>0	⊥	<0	>0	⊤
⊤	⊥	⊤	⊤	⊤

Correctness

The abstract operations $+^{\#}$ and $\times^{\#}$ are correct on the domain Sign:



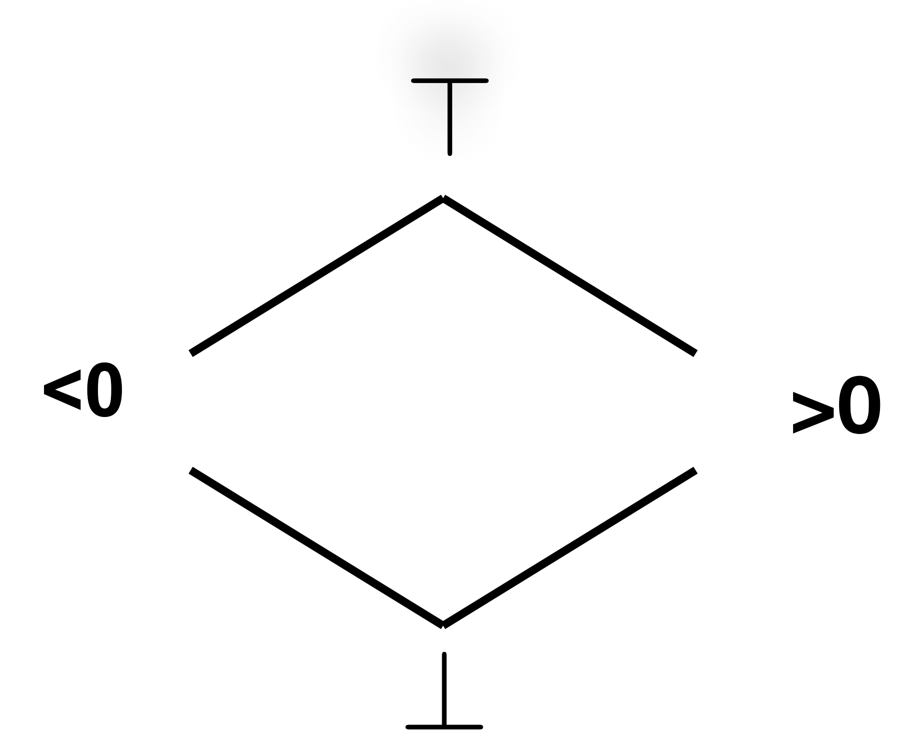
$$\forall n, m \in C \,.\, \alpha(n) +^{\#} \alpha(m) \sqsupseteq \alpha(n+m)$$

Remember that an abstract operation $F^{\#}$ is correct on an abstract domain A whenever it returns a correct approximation of the result of the concrete computation:

$$F^{\#} \sqsupseteq \alpha F \gamma = F^{A}$$

Completeness

The abstract operation $\times^\#$ has a very nice property on the domain Sign:



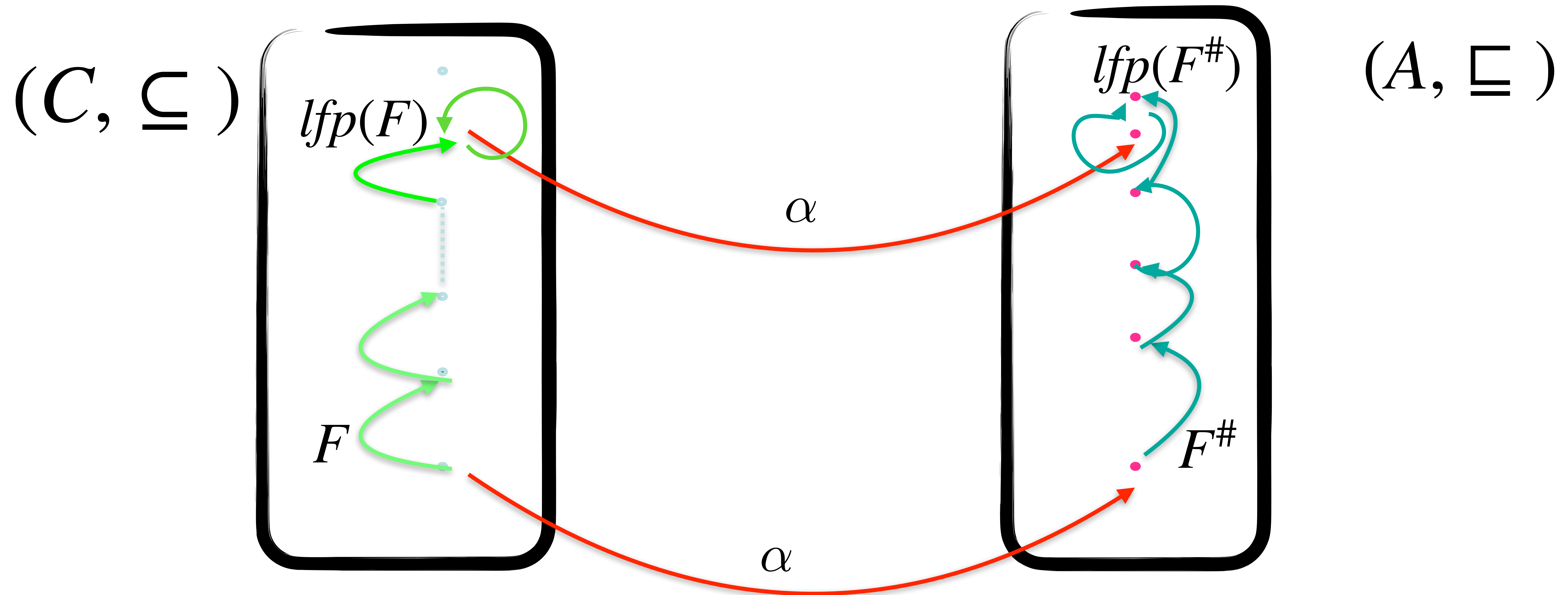
$$\forall n, m \in C \,.\, \alpha(n) \times^\# \alpha(m) \overset{\not\sqsupset}{=} \alpha(n \times m)$$

An abstract operation $F^\#$ is complete on an abstract domain A whenever it returns the best abstraction of the result of the concrete computation:

$$F^\# \alpha = \alpha F$$

Fixpoint computation approximation

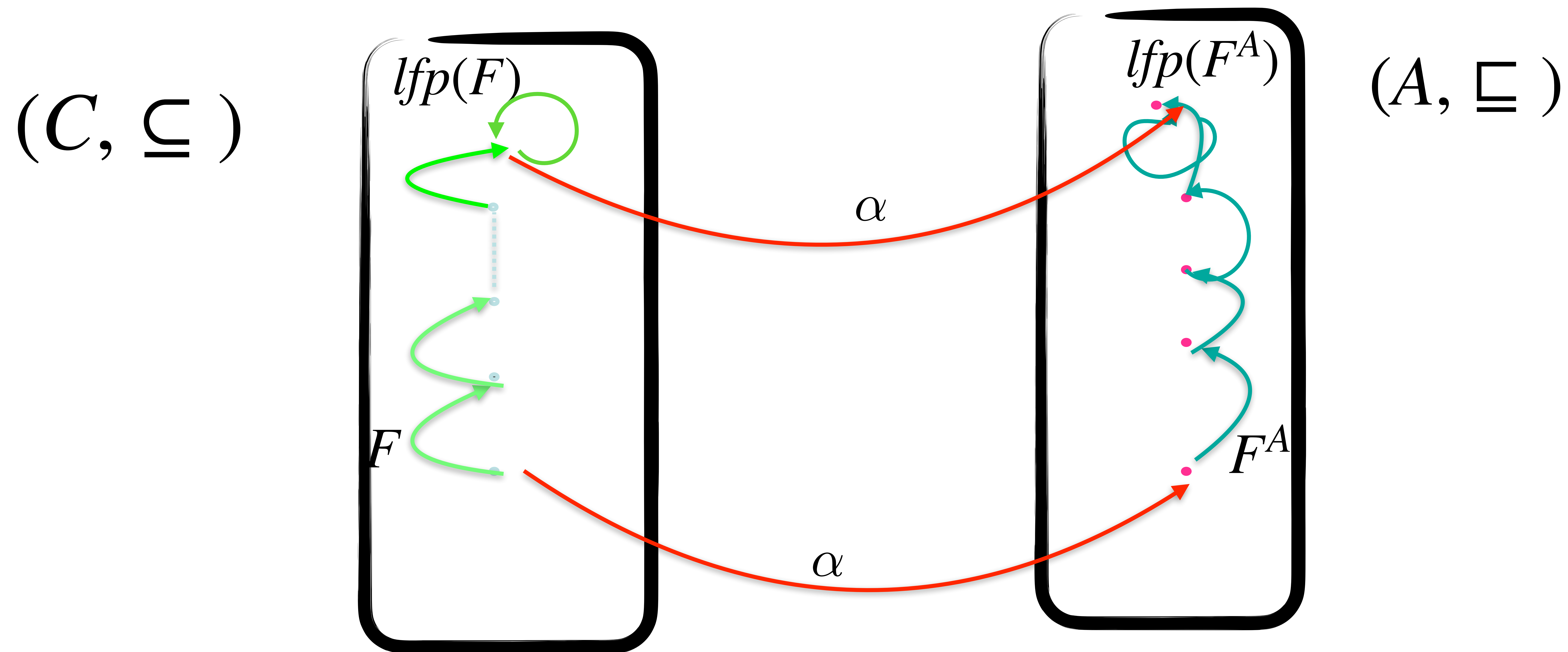
If F monotone and $F^{\#}$ correct



$lfp(F^{\#})$ is a correct over approximation of $lfp(F)$

Fixpoint computation approximation

If F monotone and F^A is complete



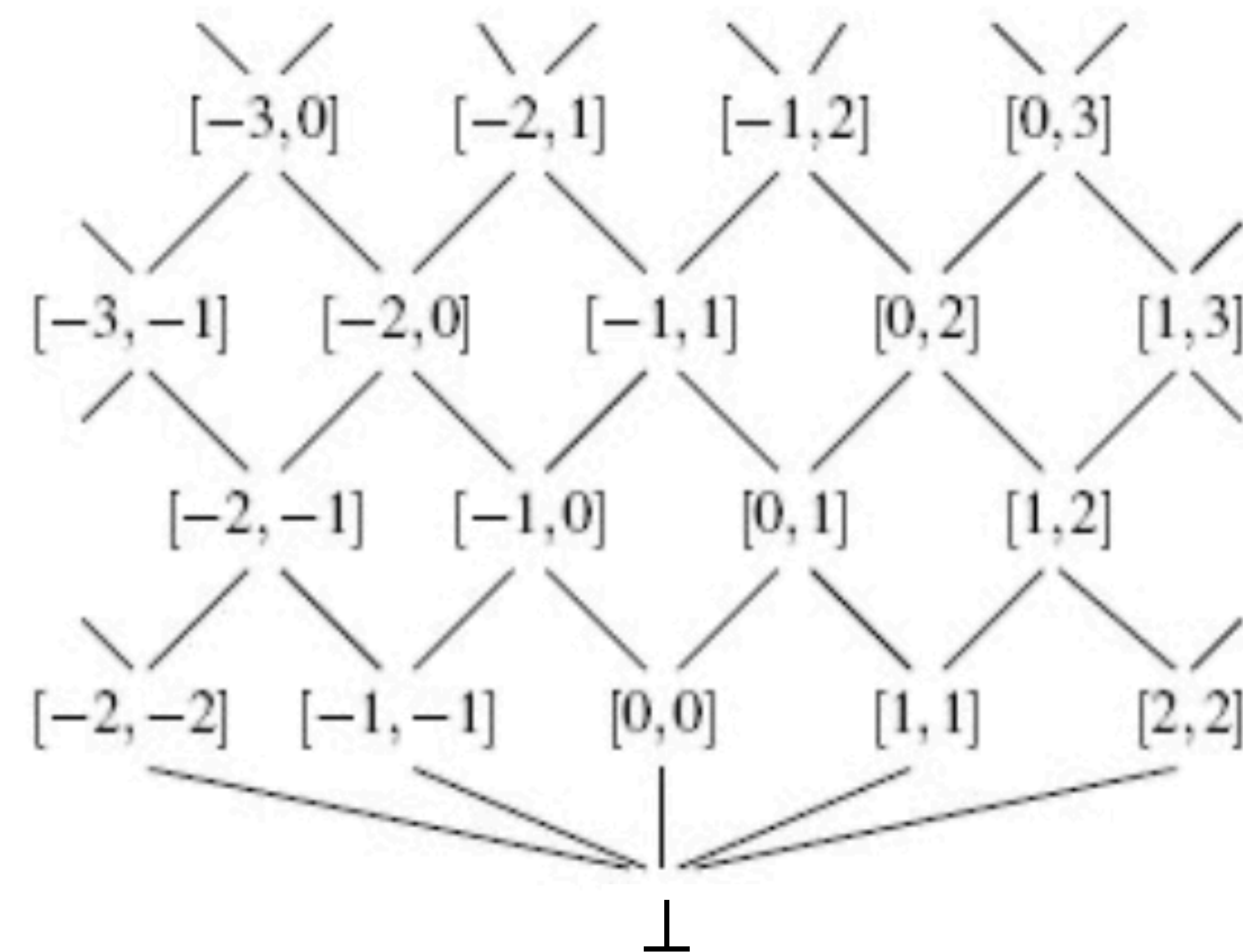
Abstract domains

Intervals

Elements of A:

- $\bot$ the empty set of values
- $(n_0, n_1),\ n_0 \in (\mathbb{Z} \cup \{-\infty\}),\ \ n_1 \in (\mathbb{Z} \cup \{+\infty\}), n_0 \leq n_1$

$\sqsubseteq$ is the interval inclusion



$$\begin{aligned}
\gamma(\bot) &= \{\} \\
\gamma([n_0, n_1]) &= \{\ n \in \mathbb{Z}\ \mid n_0 \leq n \leq n_1\} \\
\gamma([-\infty, n_1]) &= \{\ n \in \mathbb{Z} \mid n \leq n_1\} \\
\gamma([n_0, +\infty]) &= \{\ n \in \mathbb{Z} \mid n_0 \leq n\} \\
\gamma([-\infty, +\infty]) &= \mathbb{Z}
\end{aligned}$$

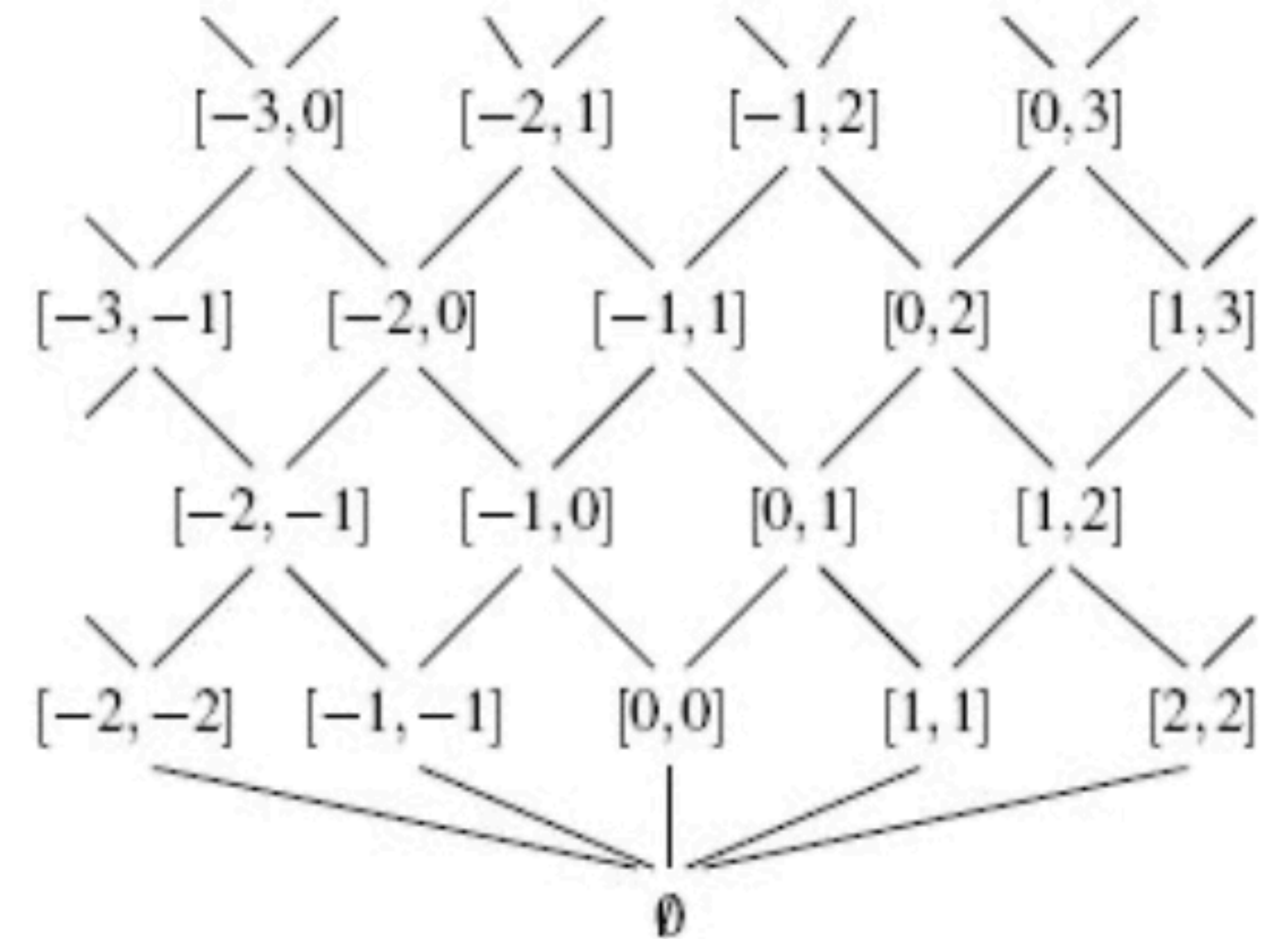
$$\begin{aligned}
&\alpha(c) = \bot \text{ if } c = \emptyset, \\
&\alpha(c) = [min(c), max(c)] \text{ if } c \neq \emptyset, min(c) \text{ and } max(c) \text{ exists} \\
&\alpha(c) = [min(c), +\infty] \text{ if } c \neq \emptyset, min(c) \text{ exists} \\
&\alpha(c) = [-\infty, max(c)] \text{ if } c \neq \emptyset, max(c) \text{ exists} \\
&\alpha(c) = [-\infty, +\infty] \text{ otherwise}
\end{aligned}$$

$+^A$ and $\times^A$ are complete on Int

$$[n,m] +^A [p,r] = [n+p, m+r]$$

$$[n,m] \times^A [p,r] = [n \times p, m \times r]$$

if all positives, otherwise pay attention

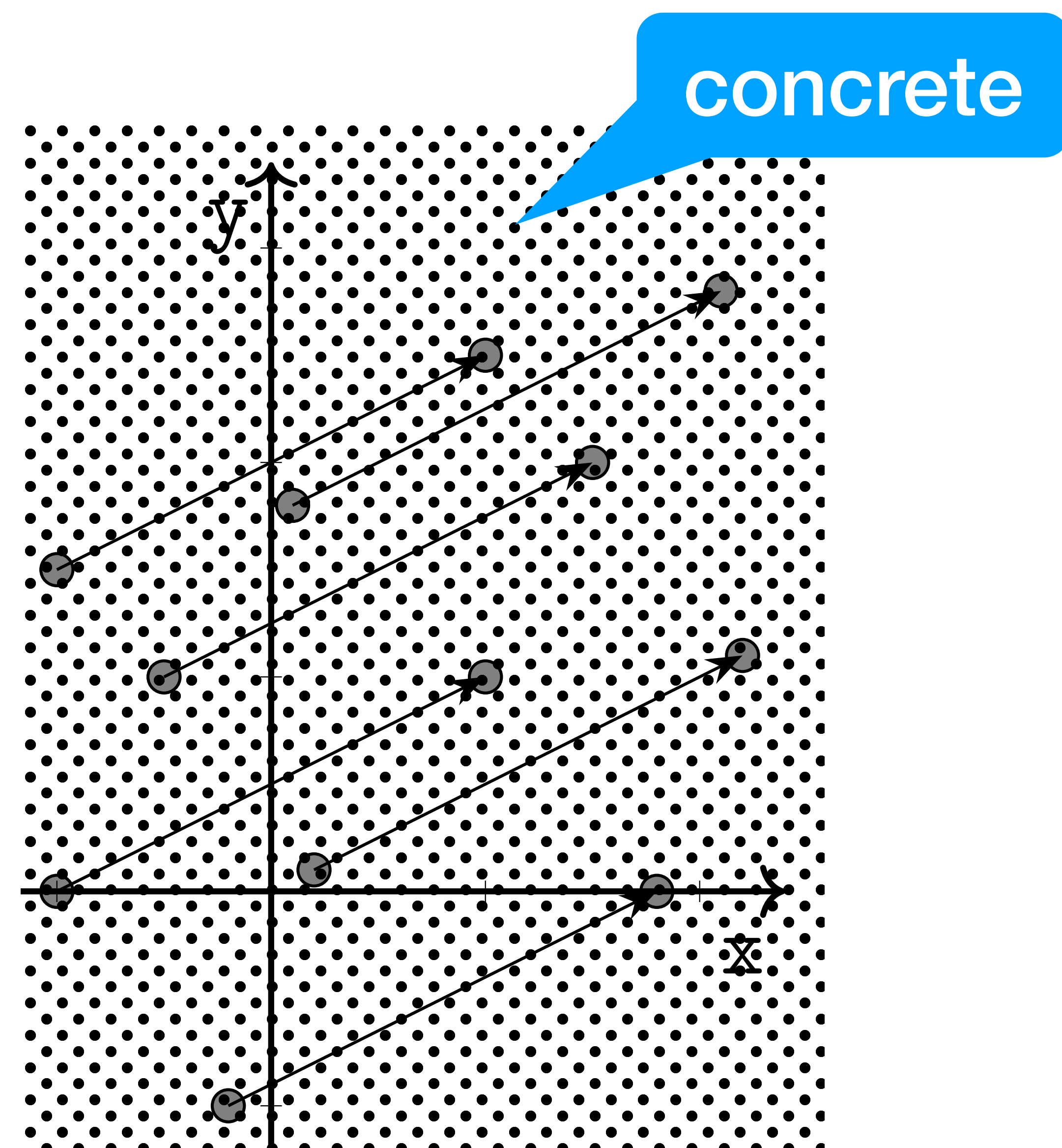


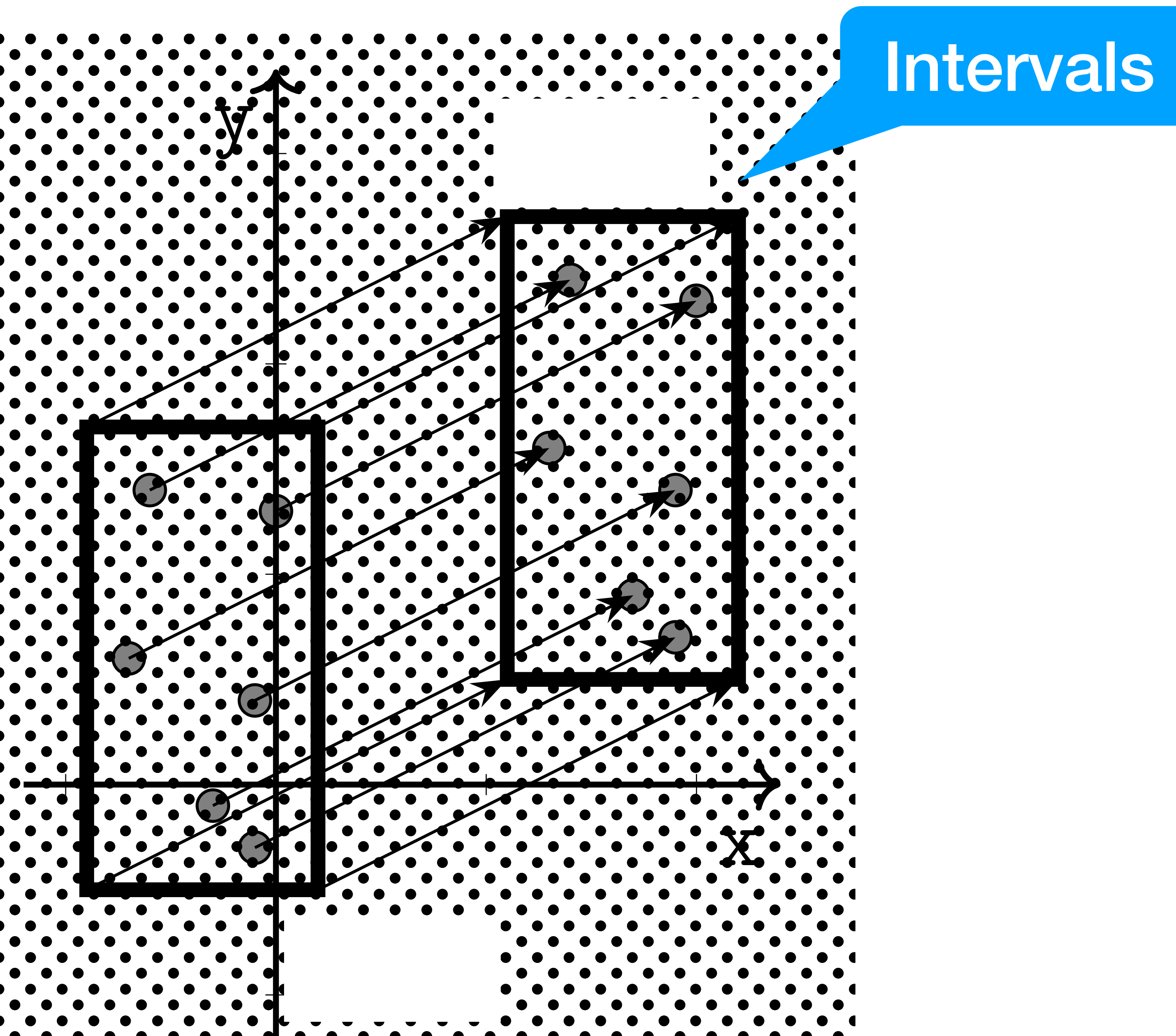
$[1,6]$ $[-3,1]$ $[-2,7]$

Precise result!

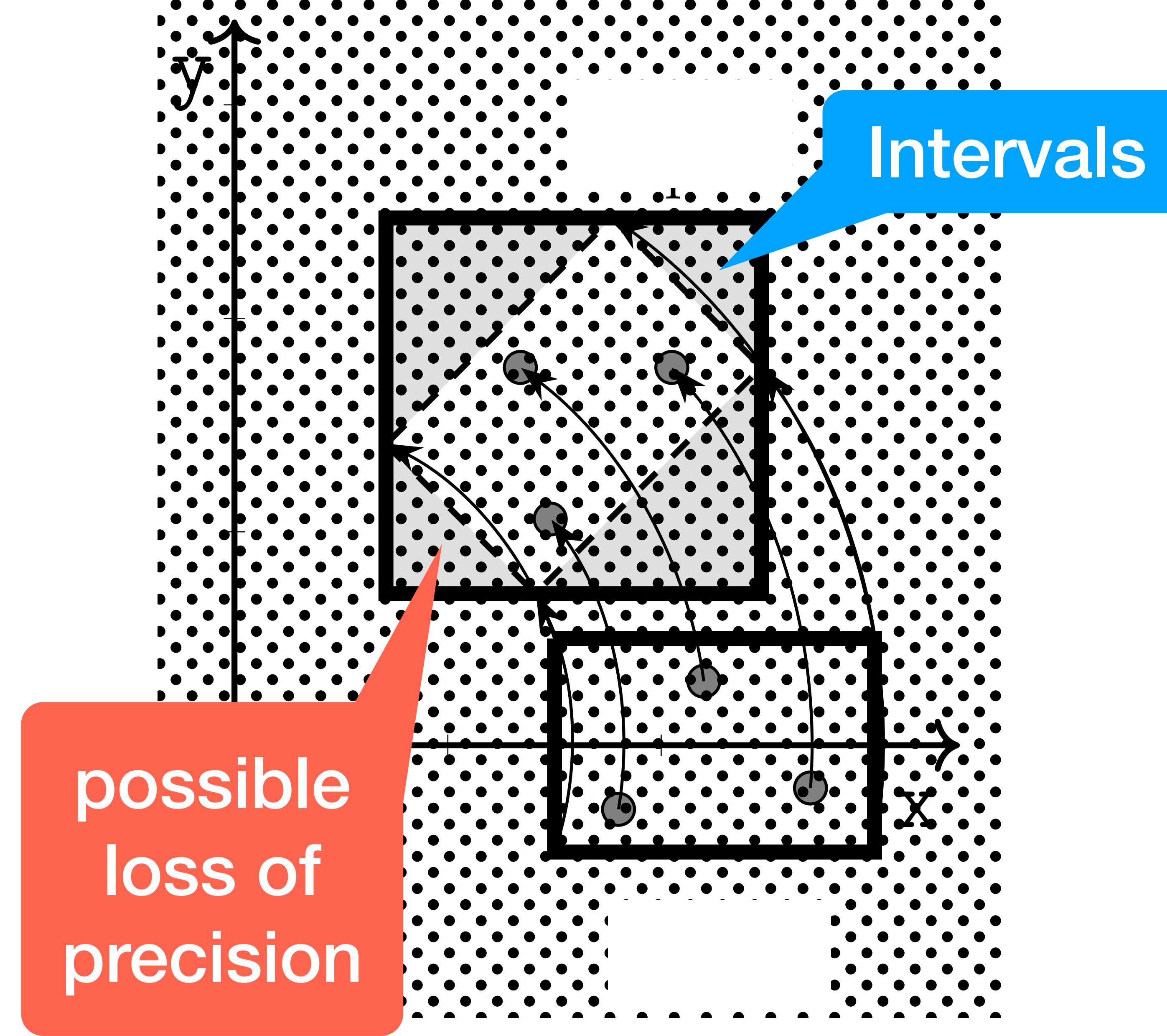
$$\{1,4,6\} + \{-3,1\} = \{-2,1,2,3,5,7\}$$

Example: translation





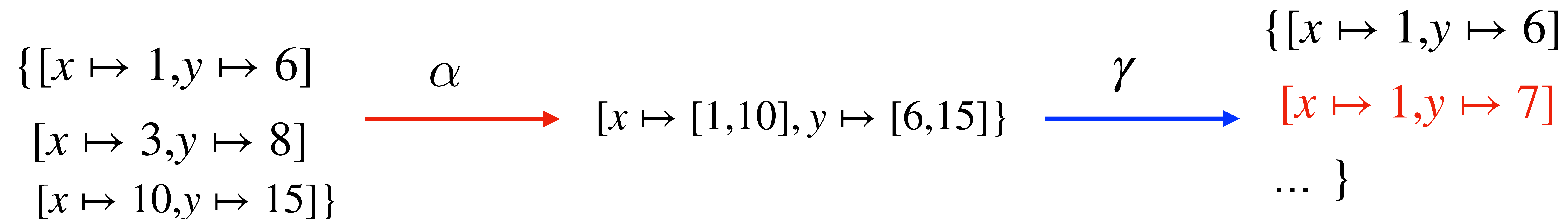
Example: rotation



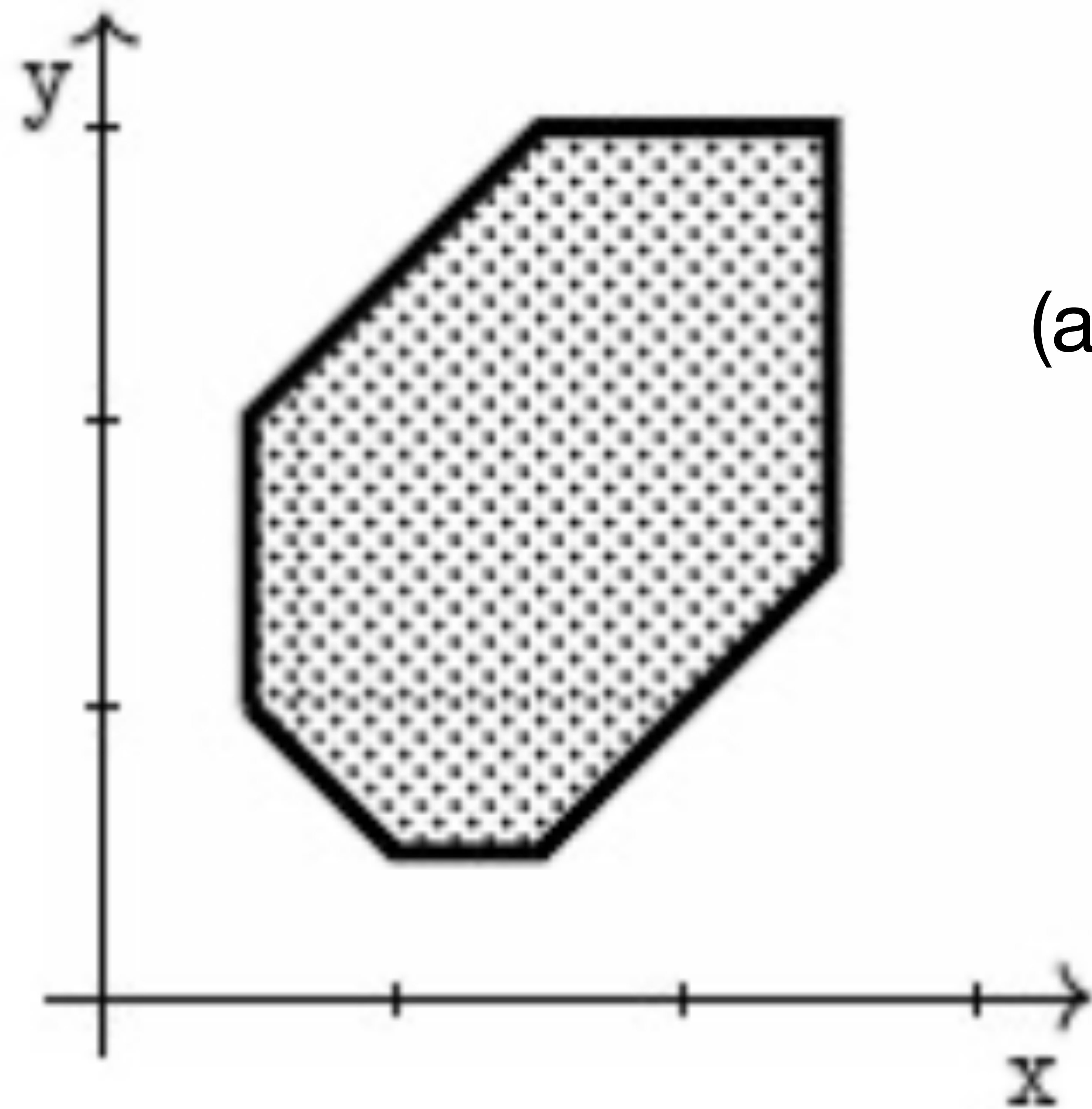
Non-relational domains

The domains of Sign and Interval are non-relational domains

They cannot track relations between variables values



Relational domain
Octagon domain



sets of numerical constraints of the form

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

(at most two variables per constraint, with unit coefficients)

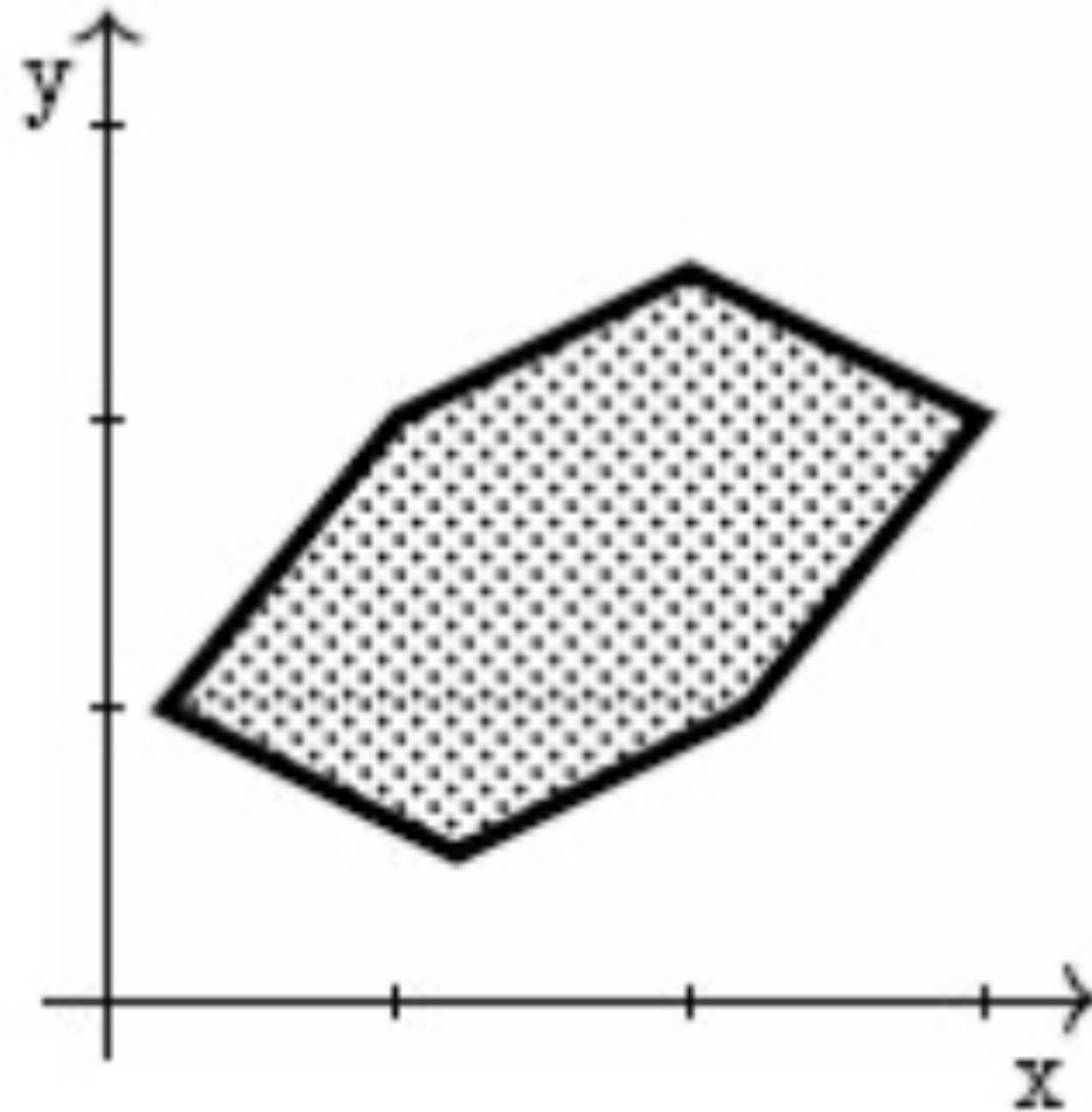
The set of states

$$\{[x \mapsto 1, y \mapsto 6]$$
$$[x \mapsto 3, y \mapsto 8]$$
$$[x \mapsto 10, y \mapsto 15]\}$$

α

$$x \leq 10$$
$$x \geq 1$$
$$y \leq 15$$
$$y \geq 6$$
$$y - x = 5$$

Relational domain
Convex Polyhedra domain



sets of numerical constraints of the form

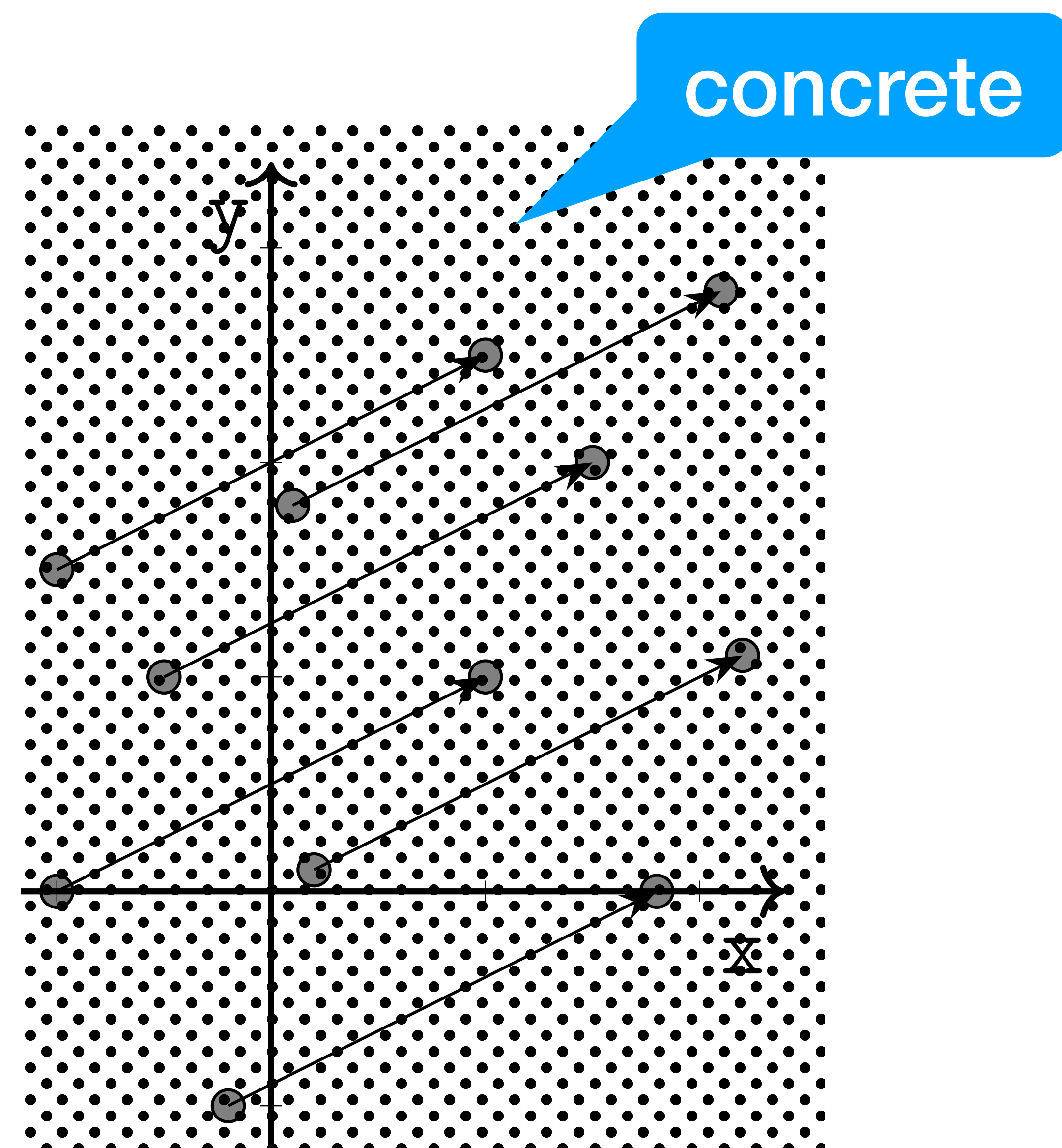
$$c_1 x + c_2 y \leq c$$

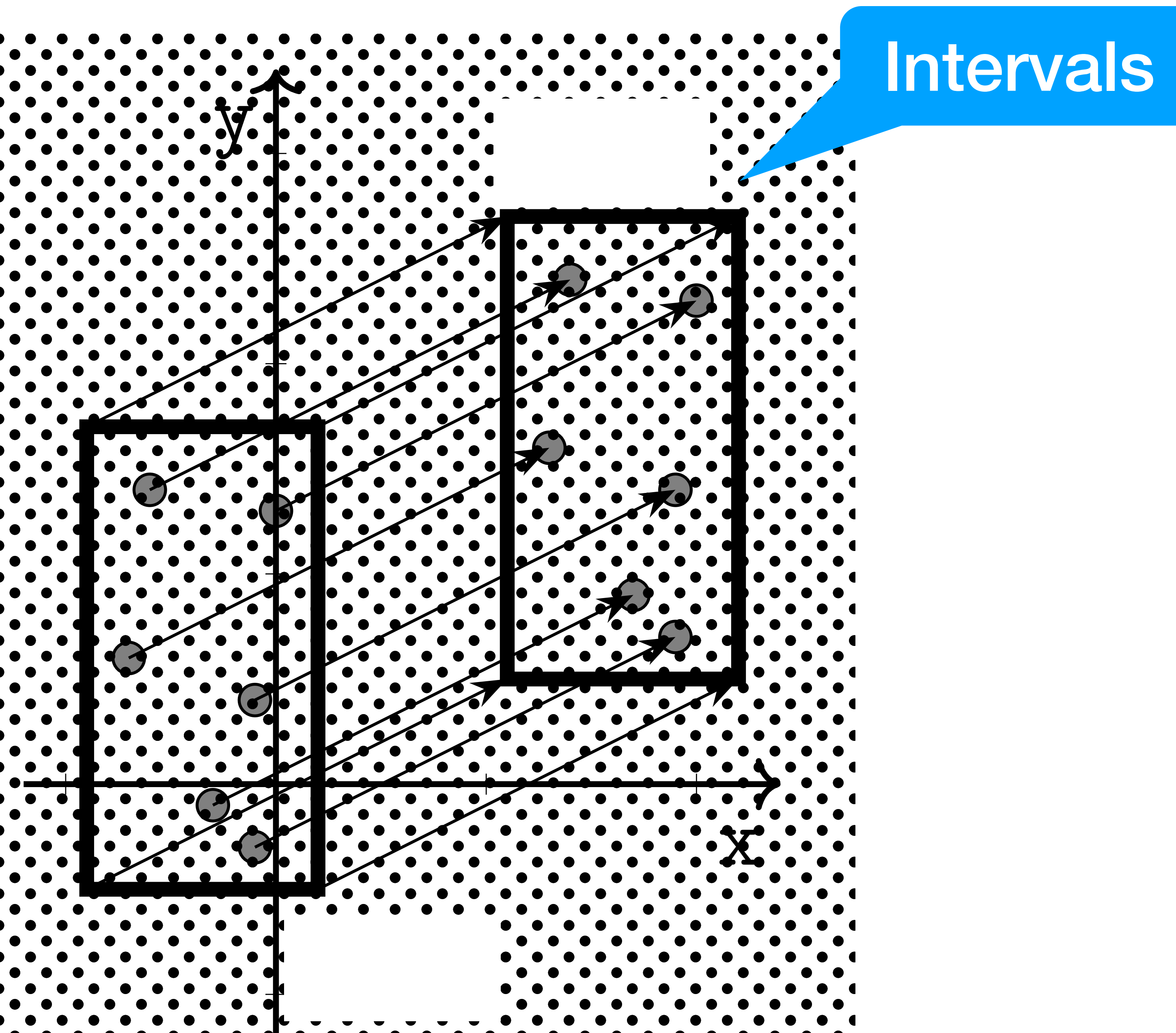
(at most two variables per constraint,
with unit coefficients)

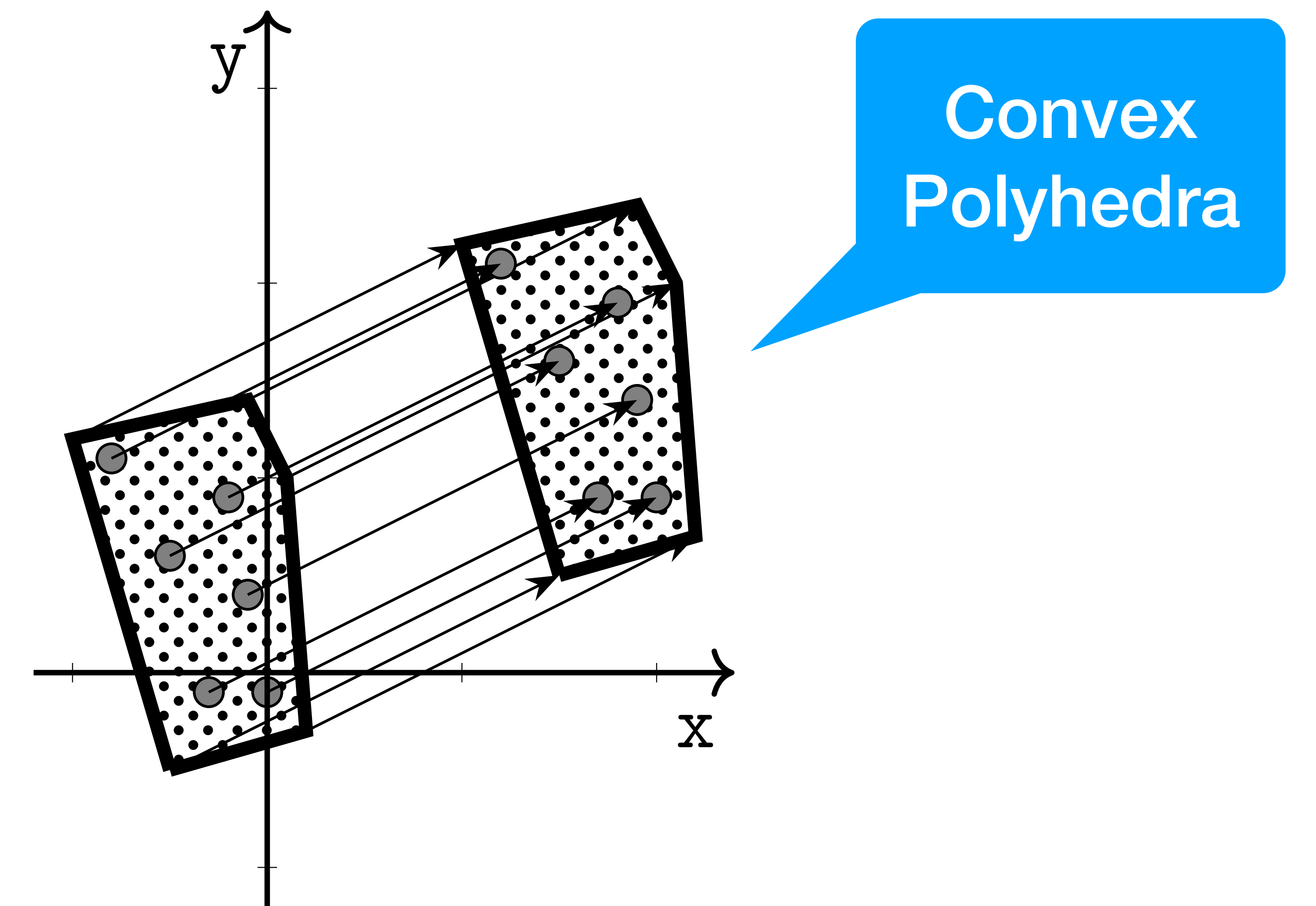
does not admit an abstraction map

best abstraction of ◯ ?

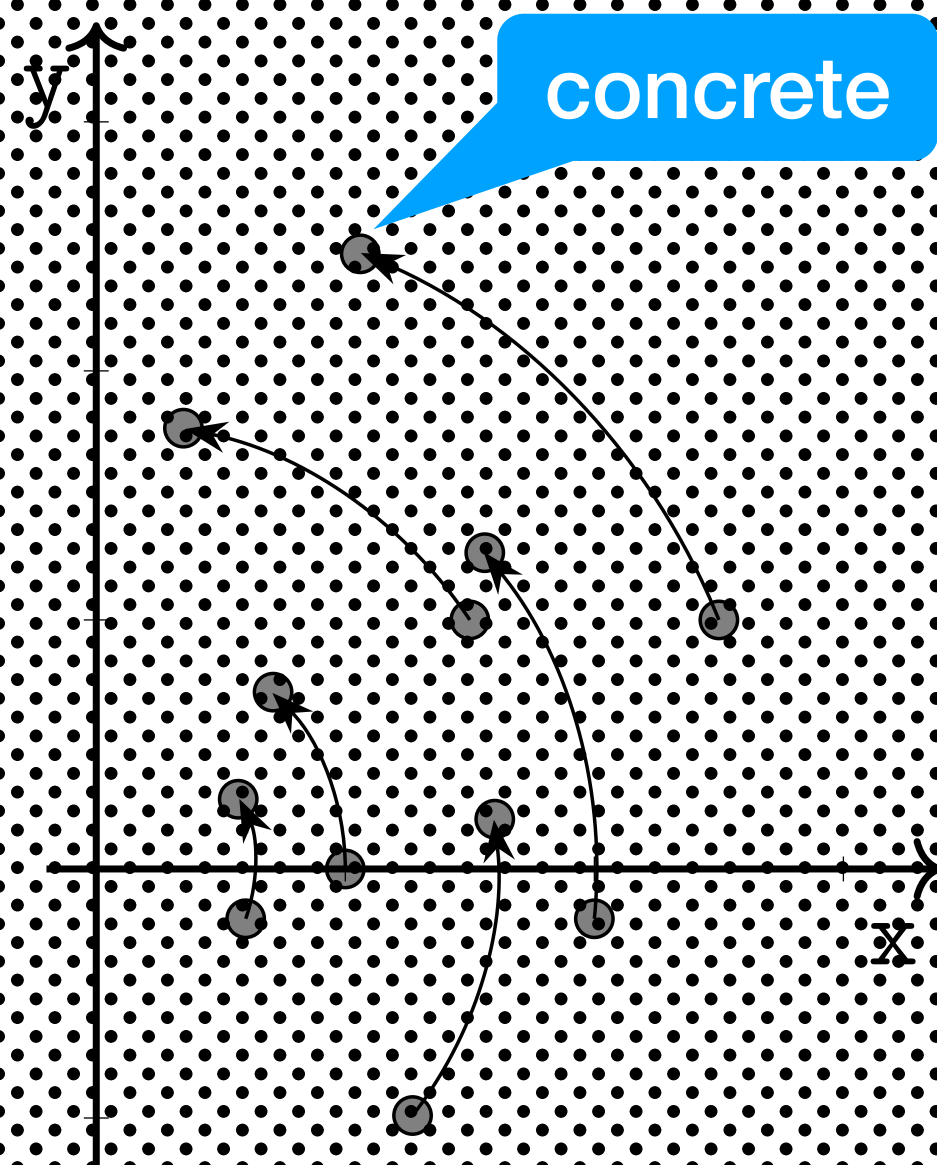
Example: translation

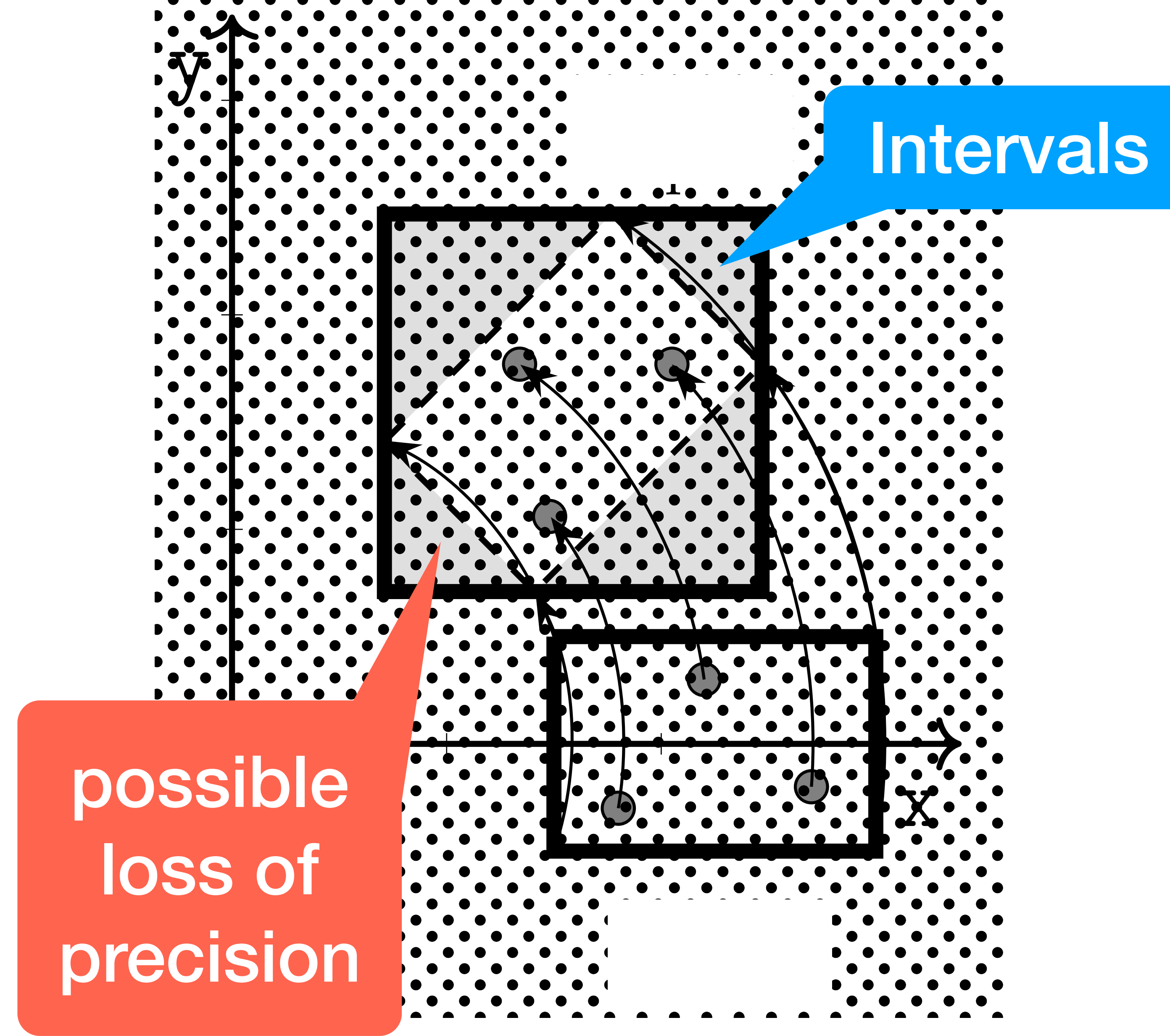


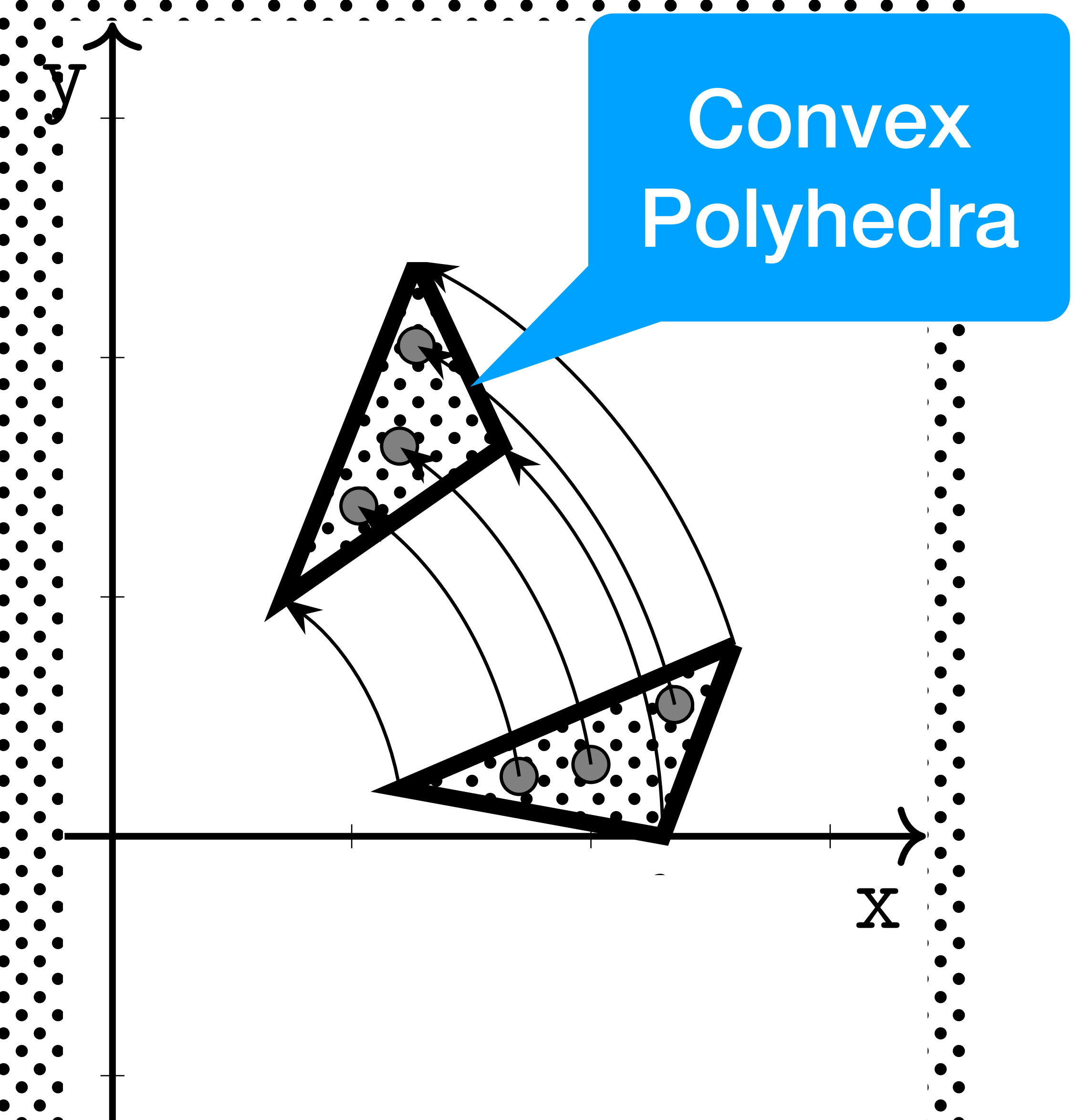




Example: rotation



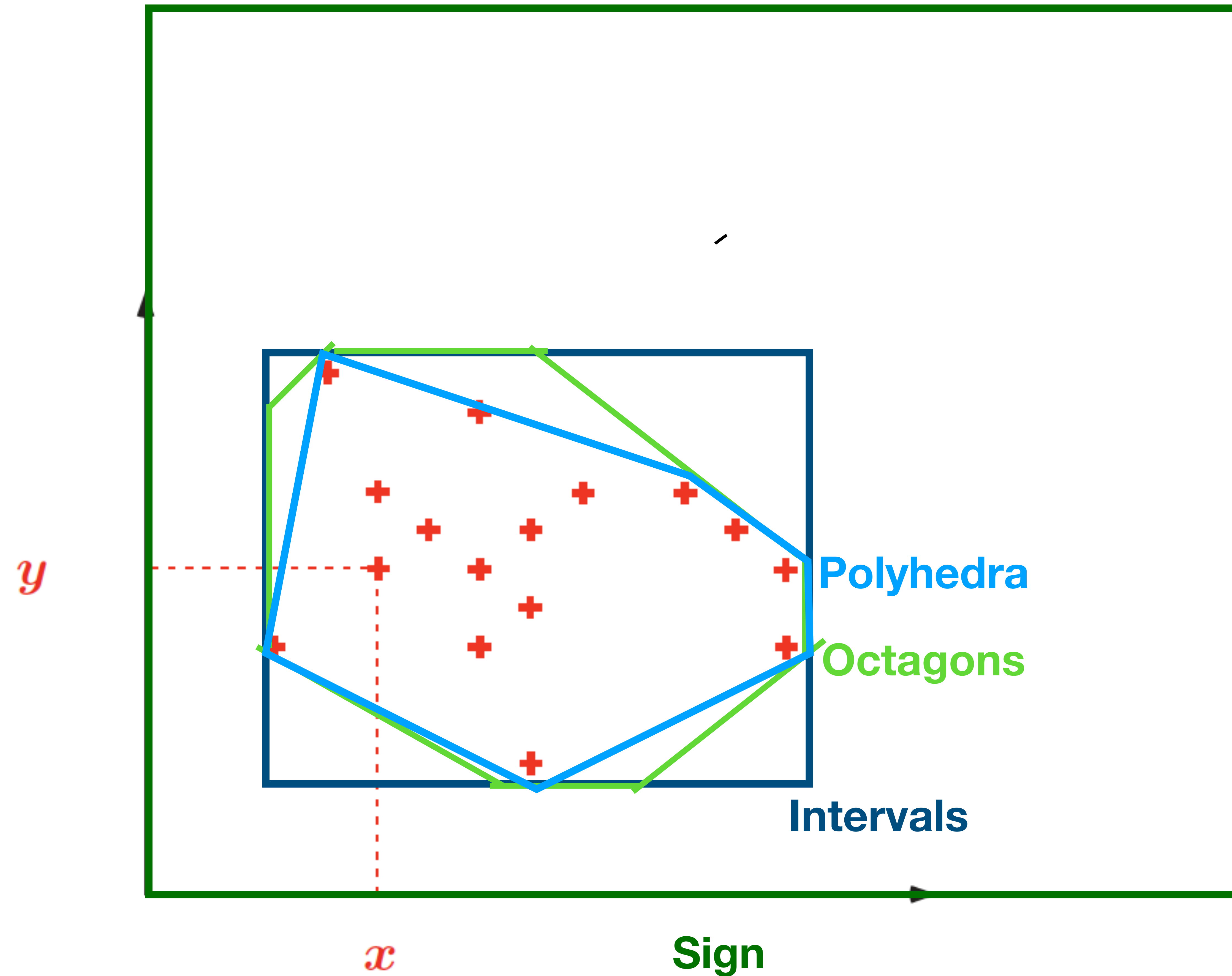




Refinements of abstraction

An (in)-finite set of points :

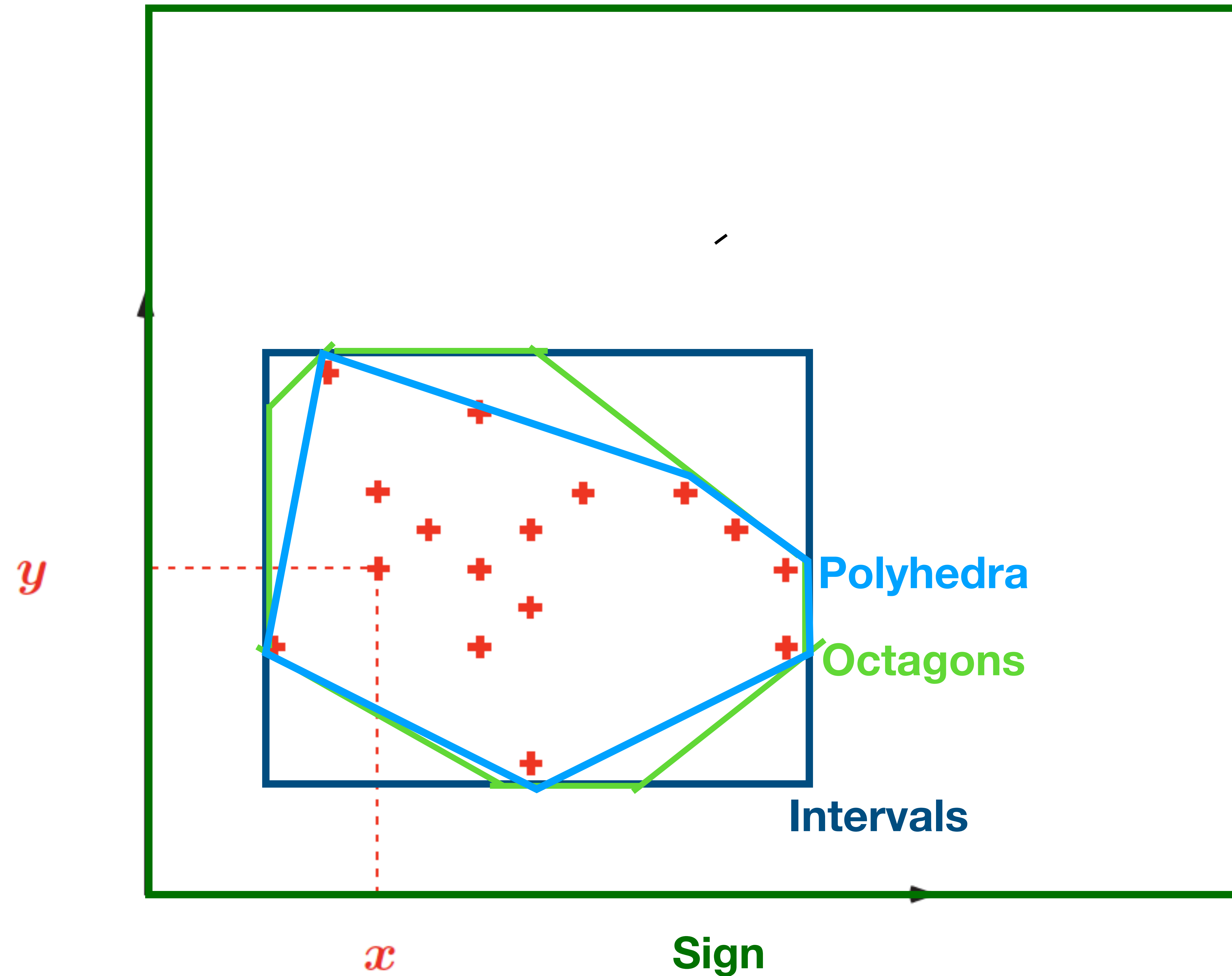
$$\{\ldots(19{,}77)\ldots(20{,}03)\ldots.\}$$



Refinements of abstraction

An (in)-finite set of points :

$\{\ldots(19{,}77)\ldots(20{,}03)\ldots.\}$



Let us analyse a code fragment on Int

```
if (x>7) {
  y := x-7
} else {
  y := 7-x
}
```

{ y ≥ 0 }?

$[\, x \mapsto \top ,\ y \mapsto \top \,]$

$if(x > 7)$

$[\, x \mapsto [8,\infty],\ y \mapsto \top \,]$

$y := x - 7$

$[\, x \mapsto [8,\infty],\ y \mapsto [1,\infty] \,]$

$else$

$[\, x \mapsto [-\infty,7],\ y \mapsto \top \,]$

$y := 7 - x$

$[\, x \mapsto [-\infty,7],\ y \mapsto [0,+\infty] \,]$

$[\, x \mapsto \top ,\ y \mapsto [0,+\infty] \,]$

Example on Interval

c_1

```
x := 10;
while (x>0) {
        x := x-1
    };
```

{ x = 0 }?

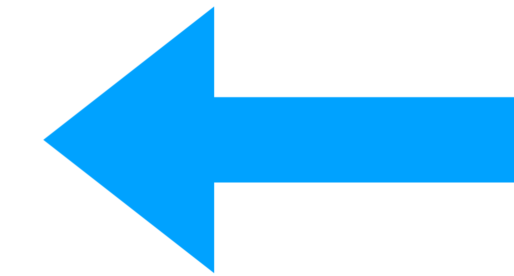
[x $\mapsto$ $\top$]

$x := 10;$

[x $\mapsto$ [10,10]]

$while(x > 0)$

[x $\mapsto$ [10,10]]



Abstract loop invariant

$x := x - 1$

[x $\mapsto$ [9,9]]

[x $\mapsto$ [0,10]]

$endwhile$

[x $\mapsto$ [0,0]]

Trace-based operational semantics

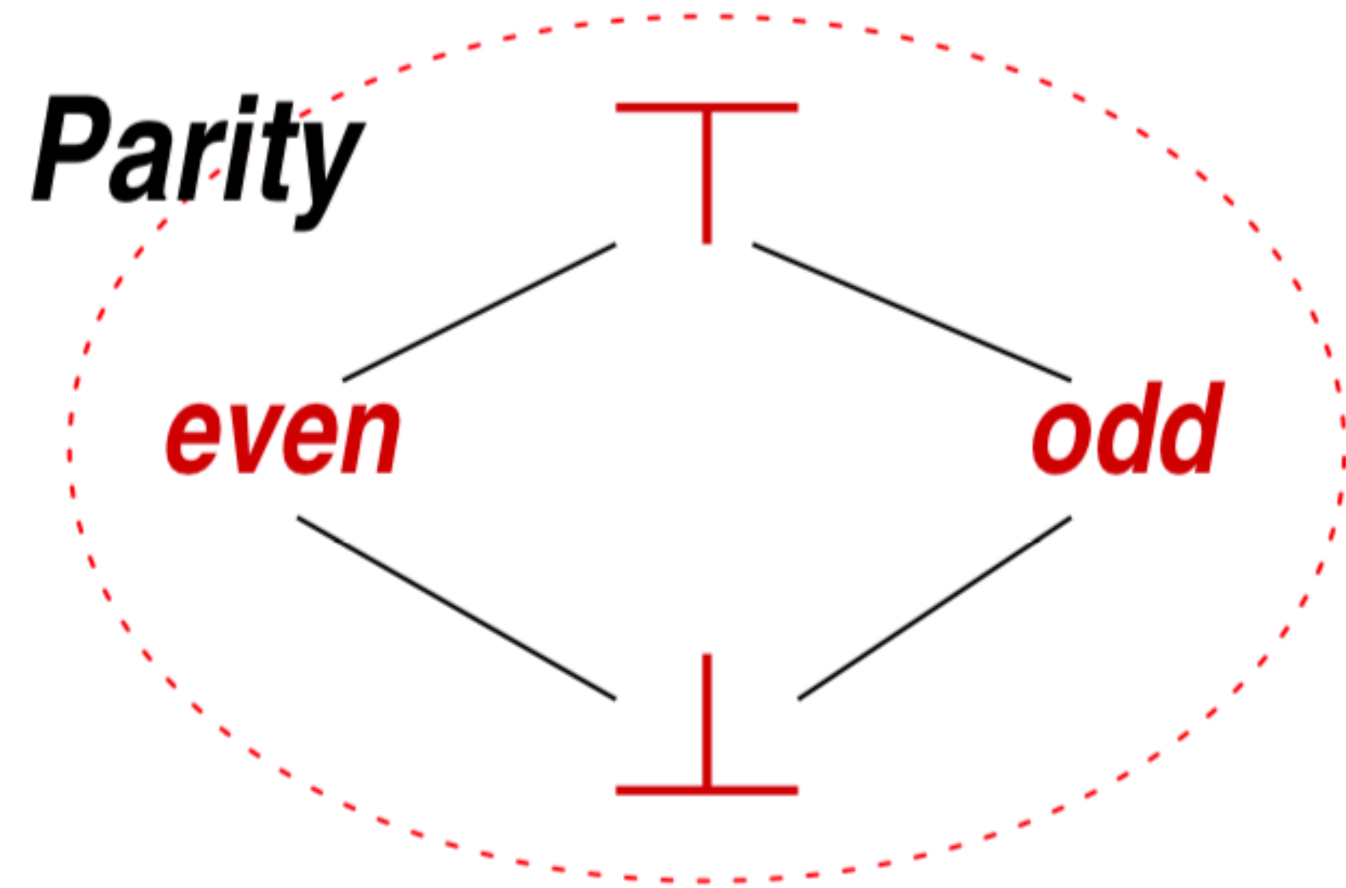
```
p0 : while isEven(x) {
       p1 : x = x div 2;
     }
p2 : x = 4 * x;
p3 : exit
```

A program's operational semantics is written as a trace:

$$p_0, 12 \longrightarrow p_1, 12 \longrightarrow p_0, 6 \longrightarrow p_1, 6 \longrightarrow p_0, 3 \longrightarrow p_2, 3 \longrightarrow p_3, 12$$

We would have infinite traces!

The parity domain



$\gamma : \text{Parity} \rightarrow \mathcal{P}(\text{Int})$

$\gamma(\text{even}) = \{\ldots, -2, 0, 2, \ldots\}$

$\gamma(\text{odd}) = \{\ldots, -1, 1, 3, \ldots\}$

$\gamma(\top) = \text{Int}, \quad \gamma(\bot) = \{\,\}$

$\alpha : \mathcal{P}(\text{Int}) \rightarrow \text{Parity}$

$\alpha(S) = \sqcup\{\beta(v) | v \in S\}$, where $\beta(2n) = \text{even}$ and $\beta(2n+1) = \text{odd}$

We interpret for parity

```
p0 : while isEven(x) {
       p1 : x = x div 2;
     }
p2 : x = 4 * x;
p3 : exit
```

$p_0, even \longrightarrow p_1, even$

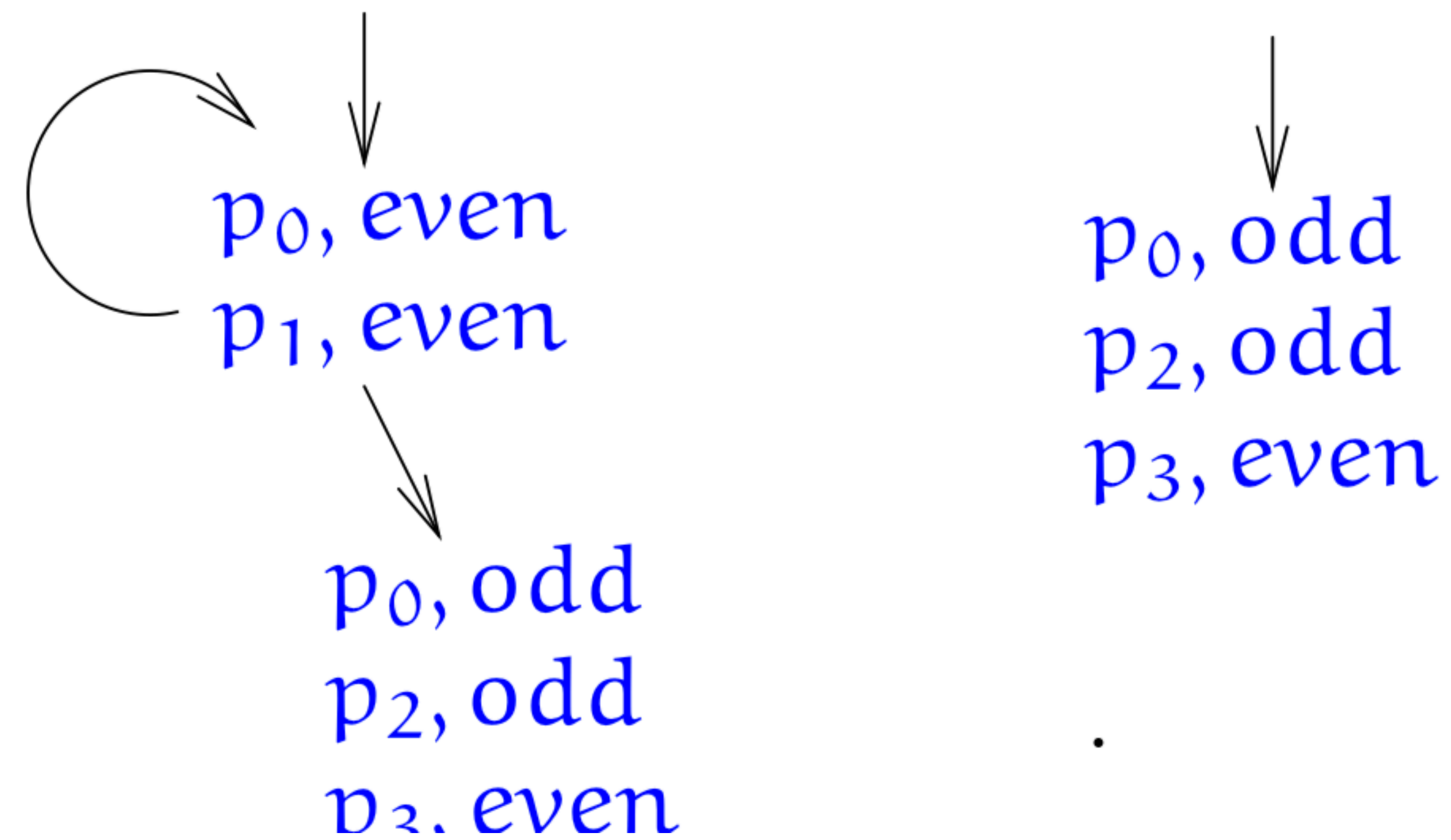
$p_0, odd \longrightarrow p_2, odd$

$p_1, even \longrightarrow p_0, even$

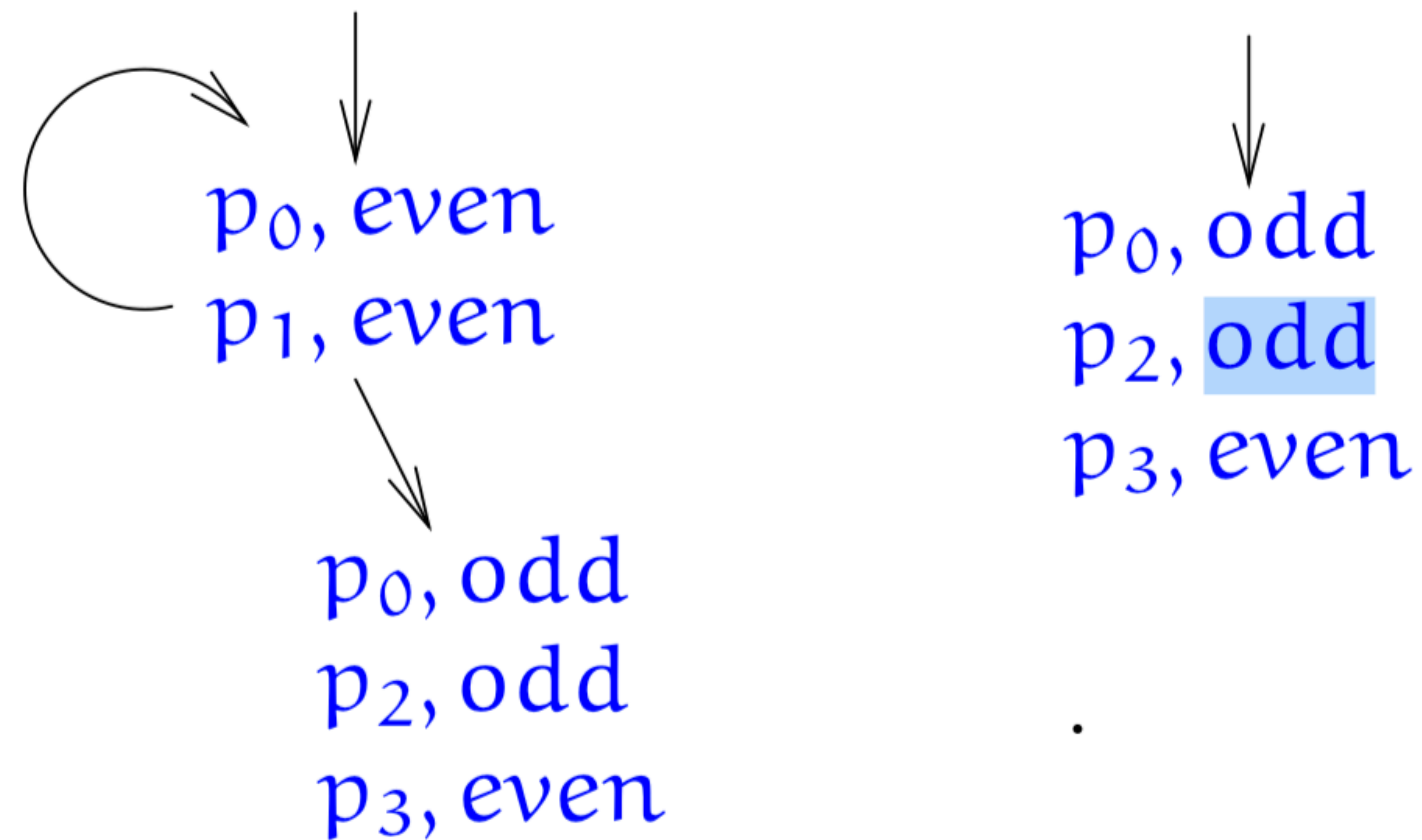
$p_1, even \longrightarrow p_0, odd$

$p_2, a \longrightarrow p_3, even$

Two trace trees cover the full range of inputs:



The interpretation of the program's semantics with the abstract values is an *abstract interpretation*:



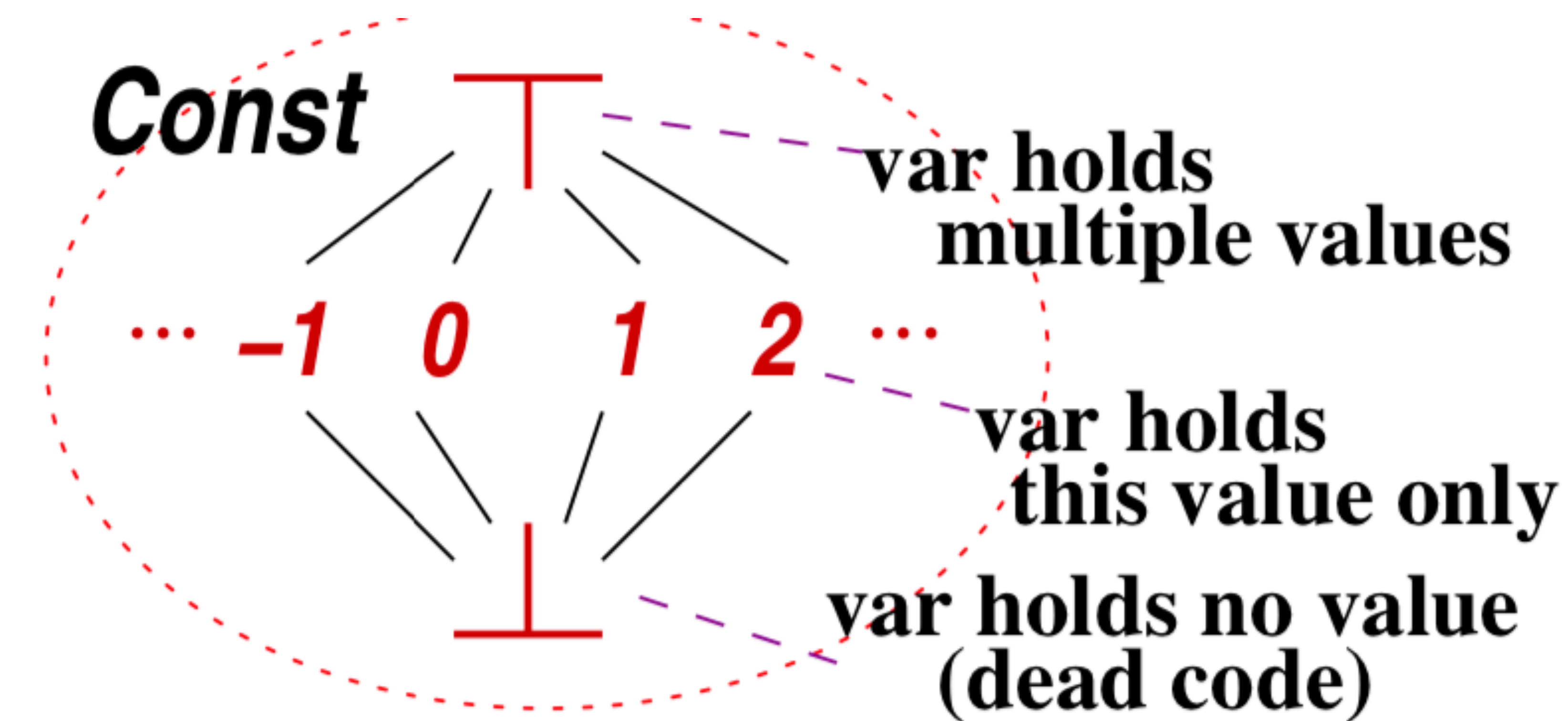
We conclude that

- if the program terminates, `x` is even-valued
- if the input is odd-valued, the loop body, p_1, will not be entered

Due to the loss of precision, we can not decide termination for almost all the even-valued inputs. (Indeed, only 0 causes nontermination.)

Constant Propagation analysis

```
p0 :  x = 1; y = 2;
p1 :  while (x < y + z)
        p2 : x = x + 1;
      }
p3 :  exit
```



where $m + n$ is interpreted

$k_1 + k_2 \longrightarrow sum(k_1, k_2),$

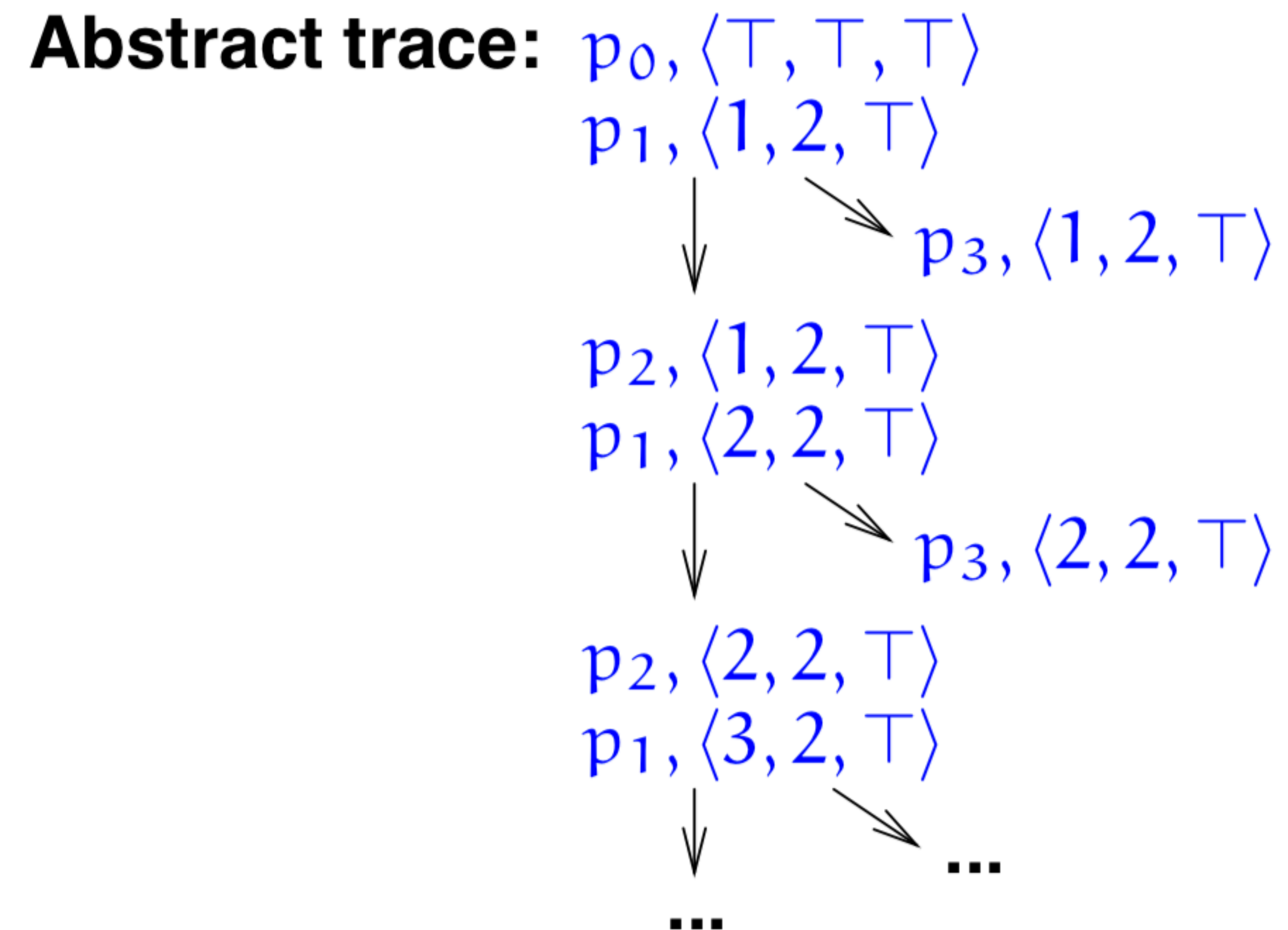
$\top \neq k_i \neq \bot, i \in 1..2$

$\top + k \longrightarrow \top$

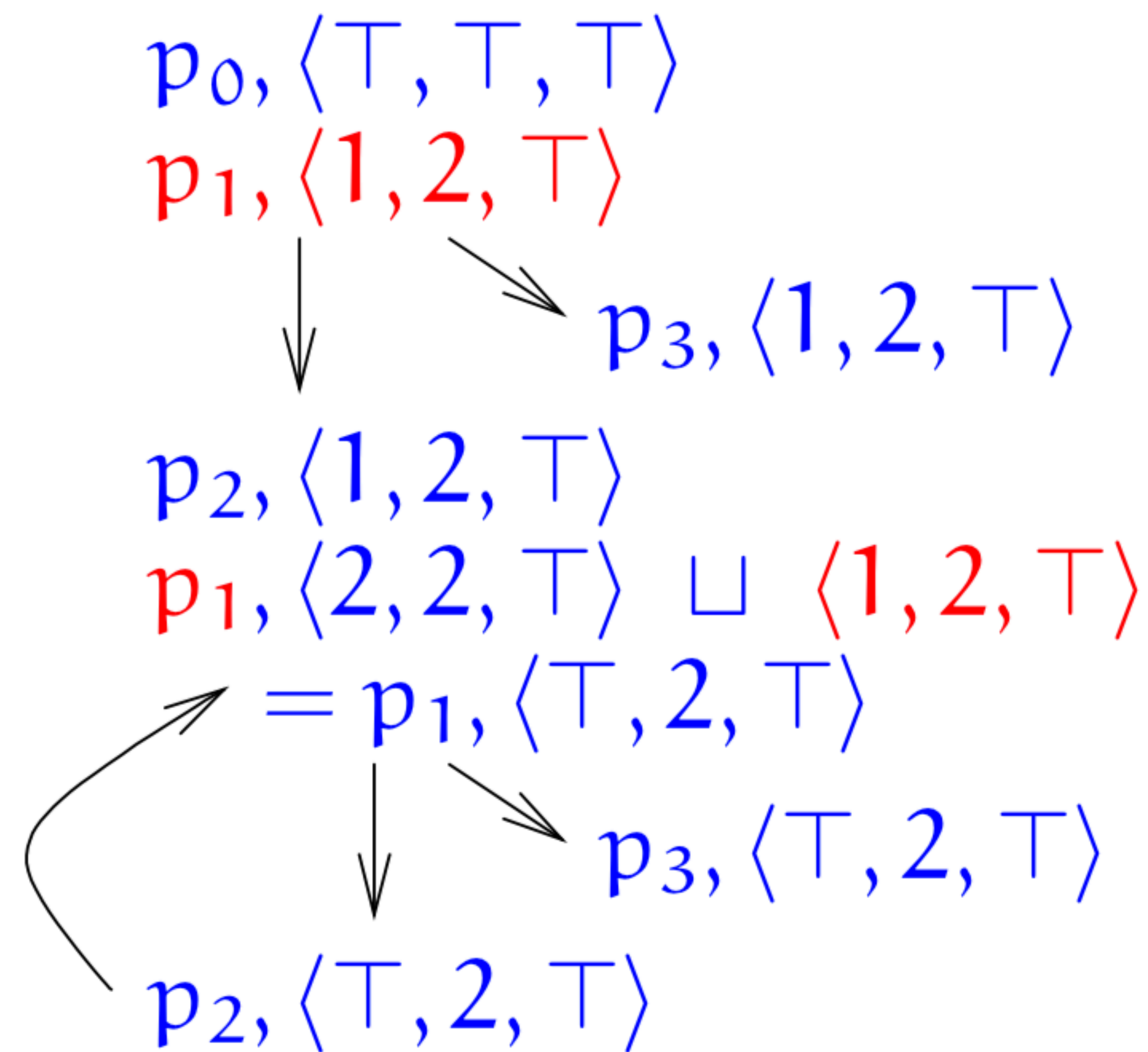
$k + \top \longrightarrow \top$

Let $\langle u, v, w \rangle$ abbreviate

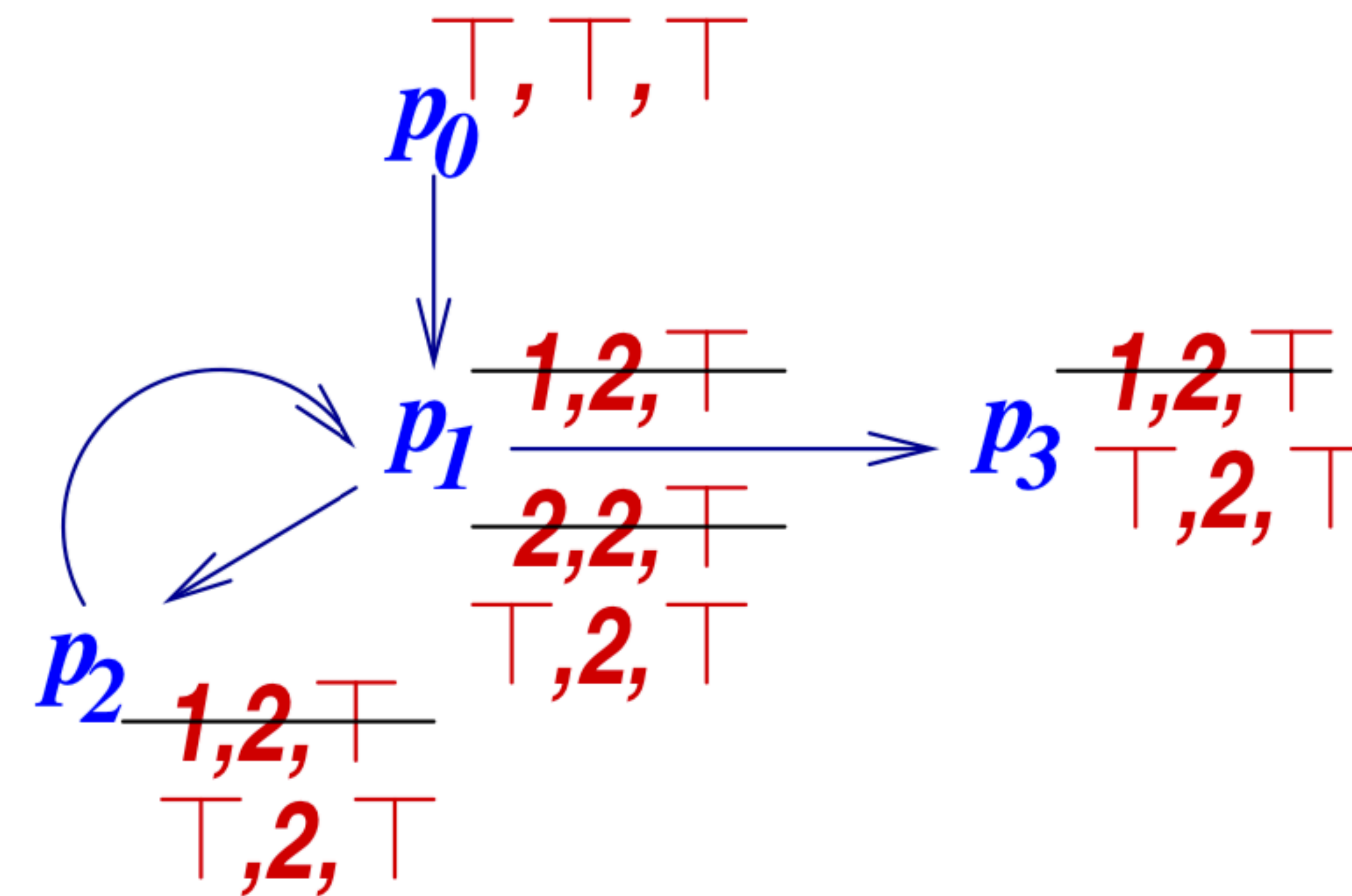
$\langle x : u, y : v, z : w \rangle$



An acceleration is needed for finite convergence: widening



Drawn as a data–flow analysis:



The analysis tells us to replace y at p_1 by 2:

```
p0 :  x = 1; y = 2;
p1 :  while (x < y + z)  {
        p2 : x = x + 1;
      }
p3 :  exit
```

2