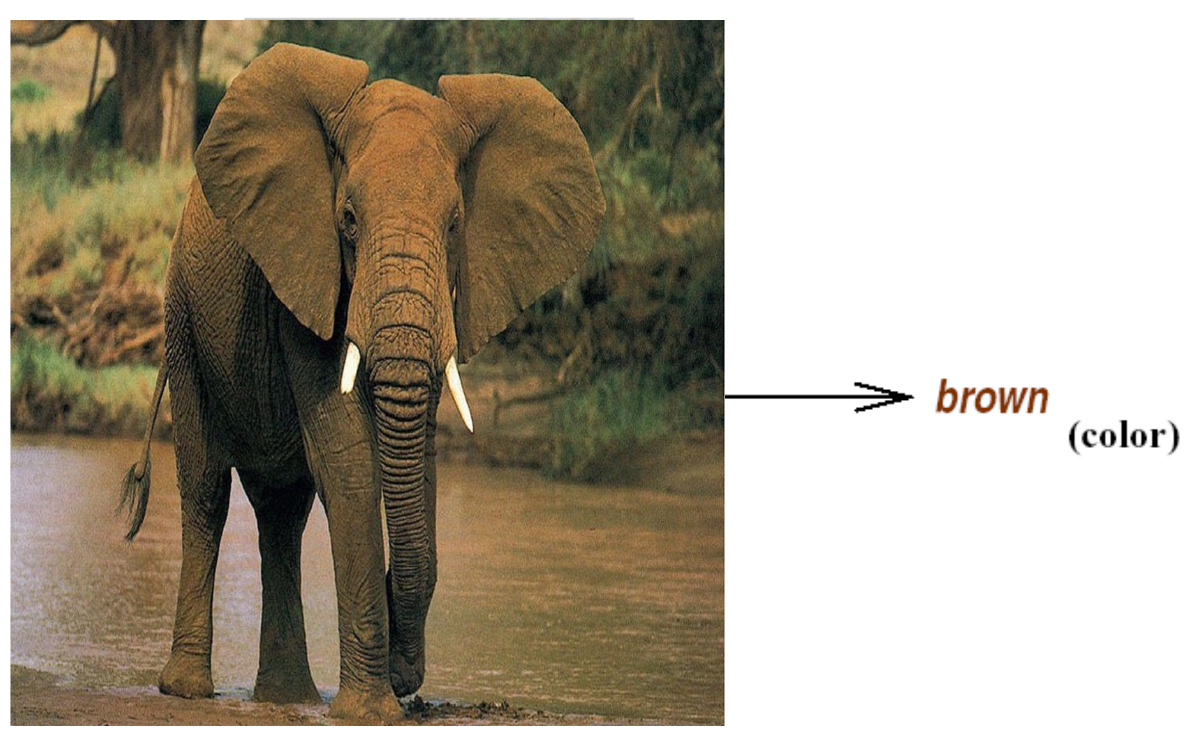
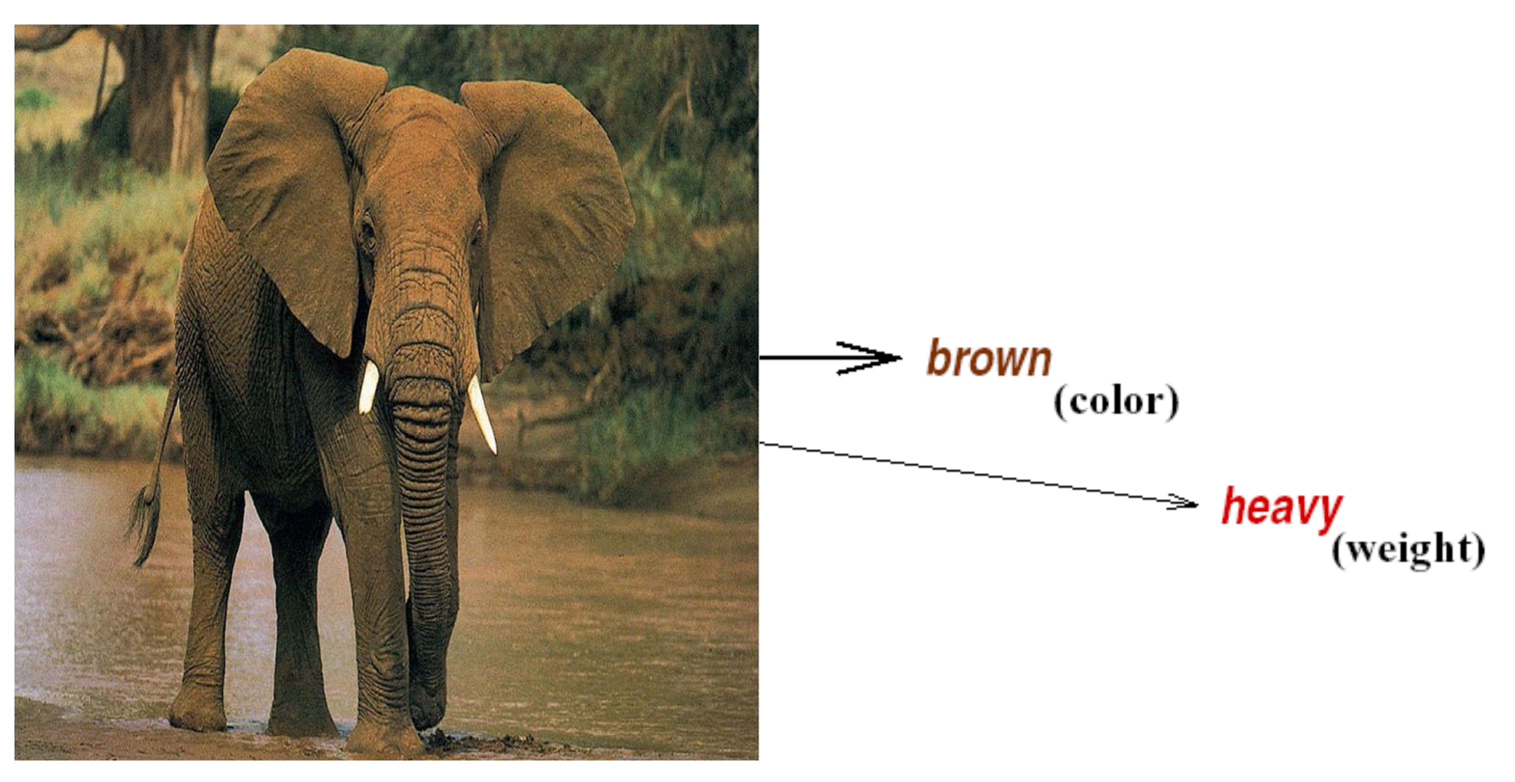
# Abstract Interpretation: an Introduction

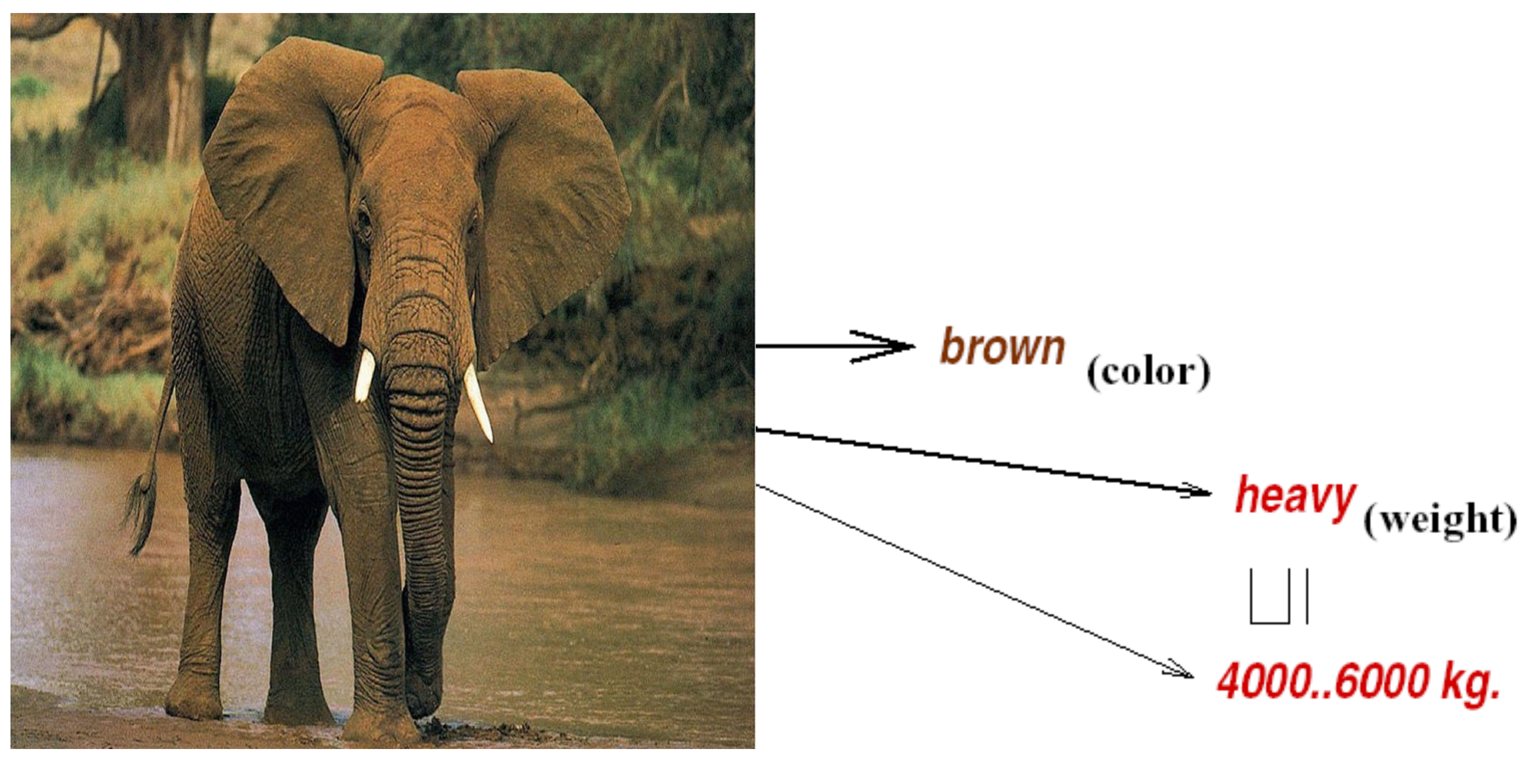
# Abstraction: selecting a property



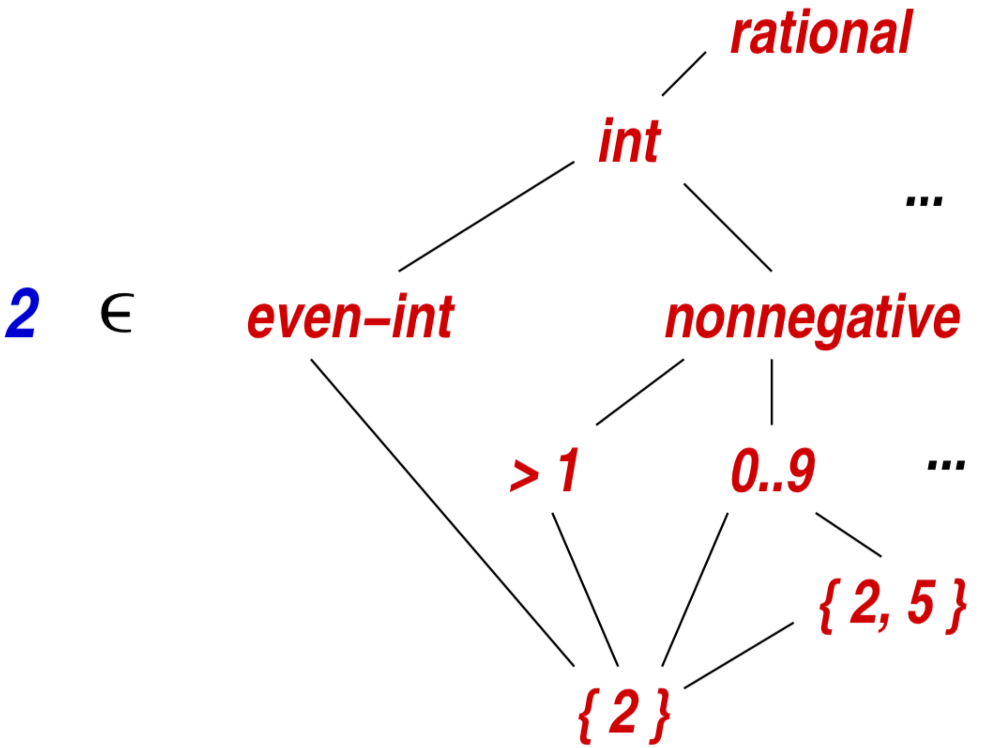
# Abstraction: selecting one out of many properties



# Abstraction and correctness

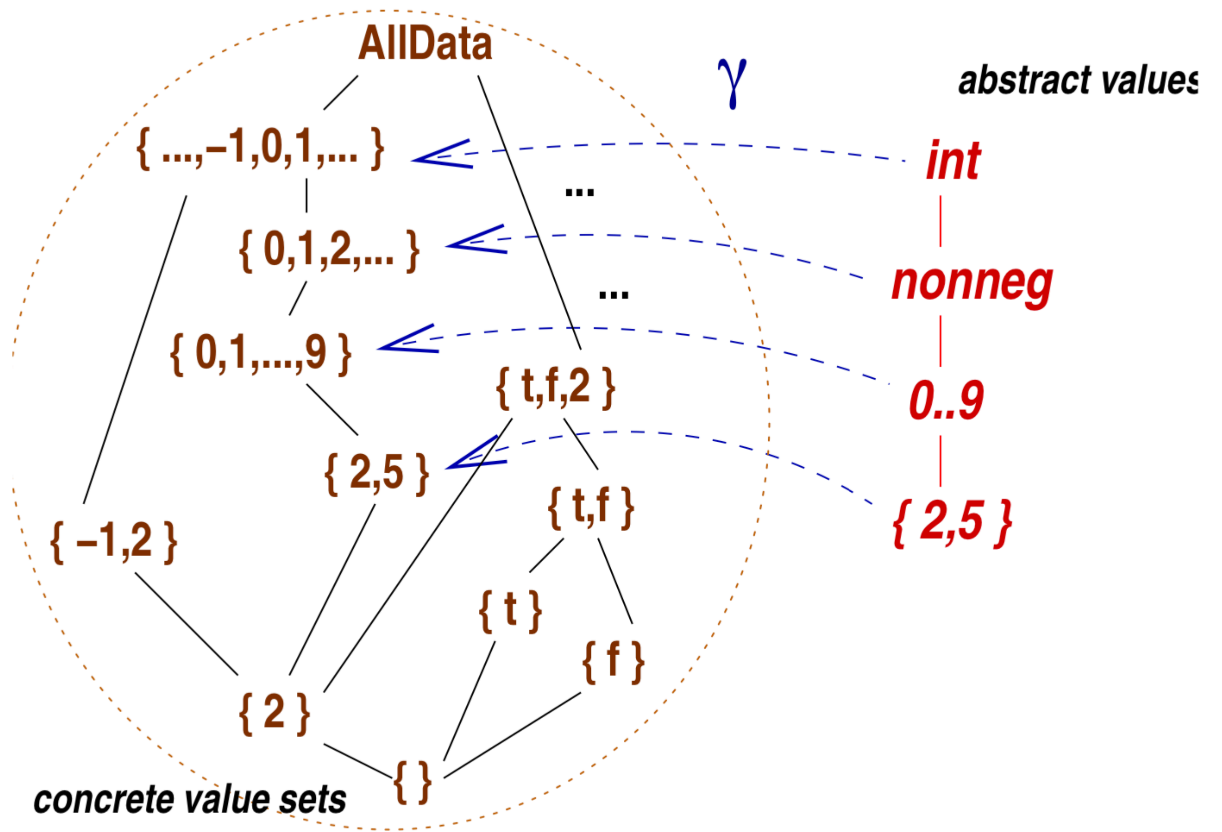


# Value abstraction are classic to computing



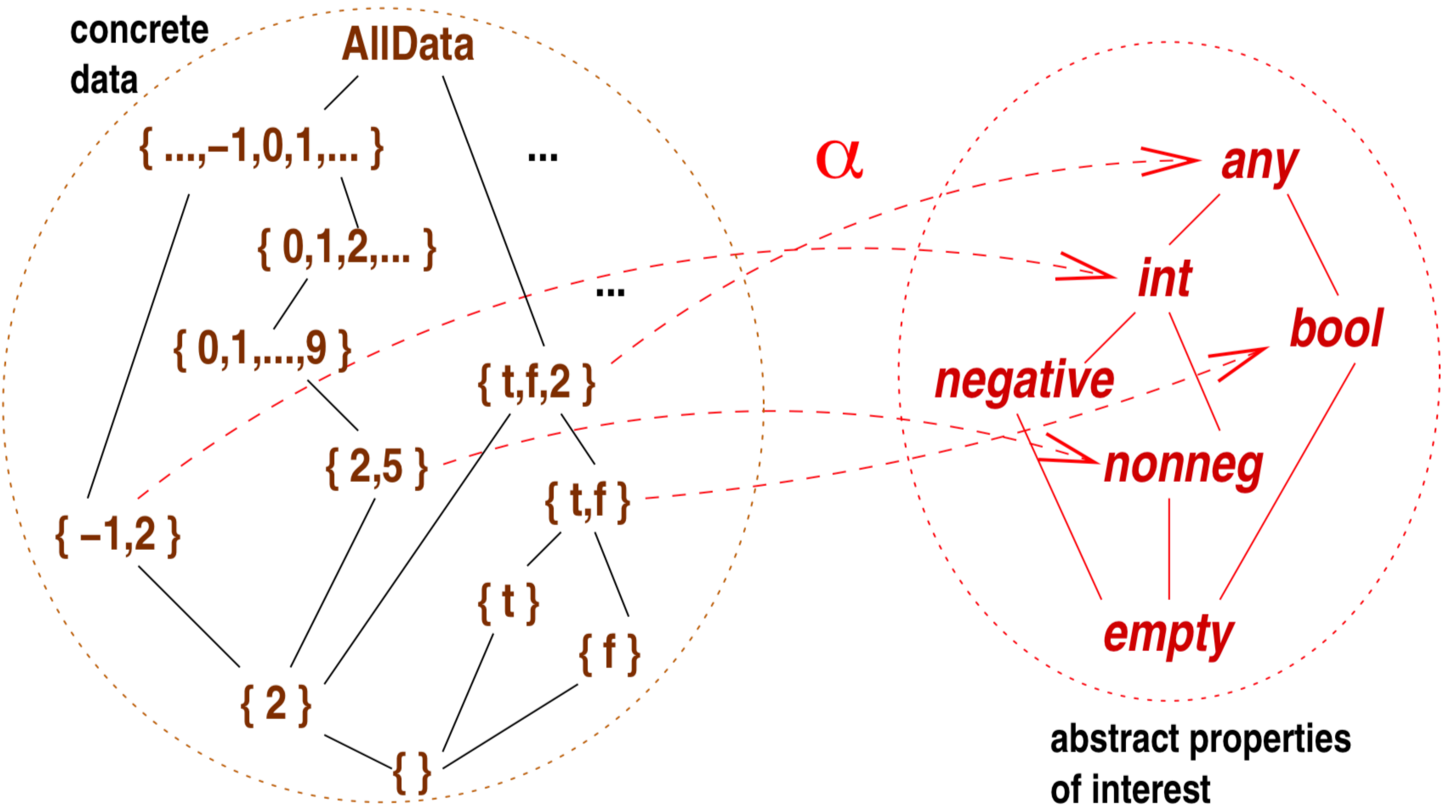

All the properties listed on the right are abstractions of 2; the upwards lines denote $\sqsubseteq$, a loss of precision.

# Abstract values represent sets of concrete values




Function $\gamma$ maps each abstract value to the set of concrete values that it represents

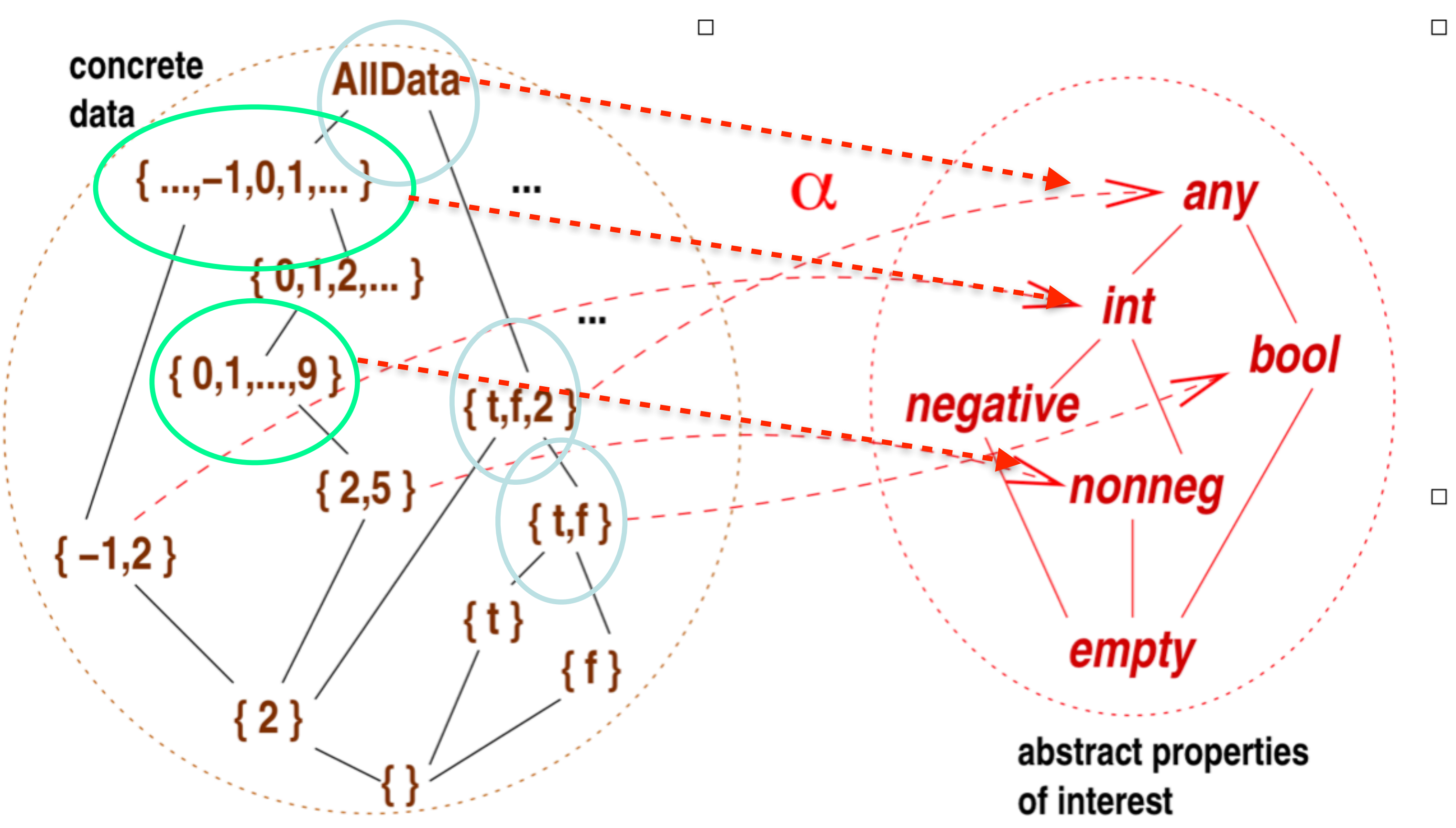
# Sets of concrete values are abstracted imprecisely




Function α maps each set to the abstract value that best describes it.

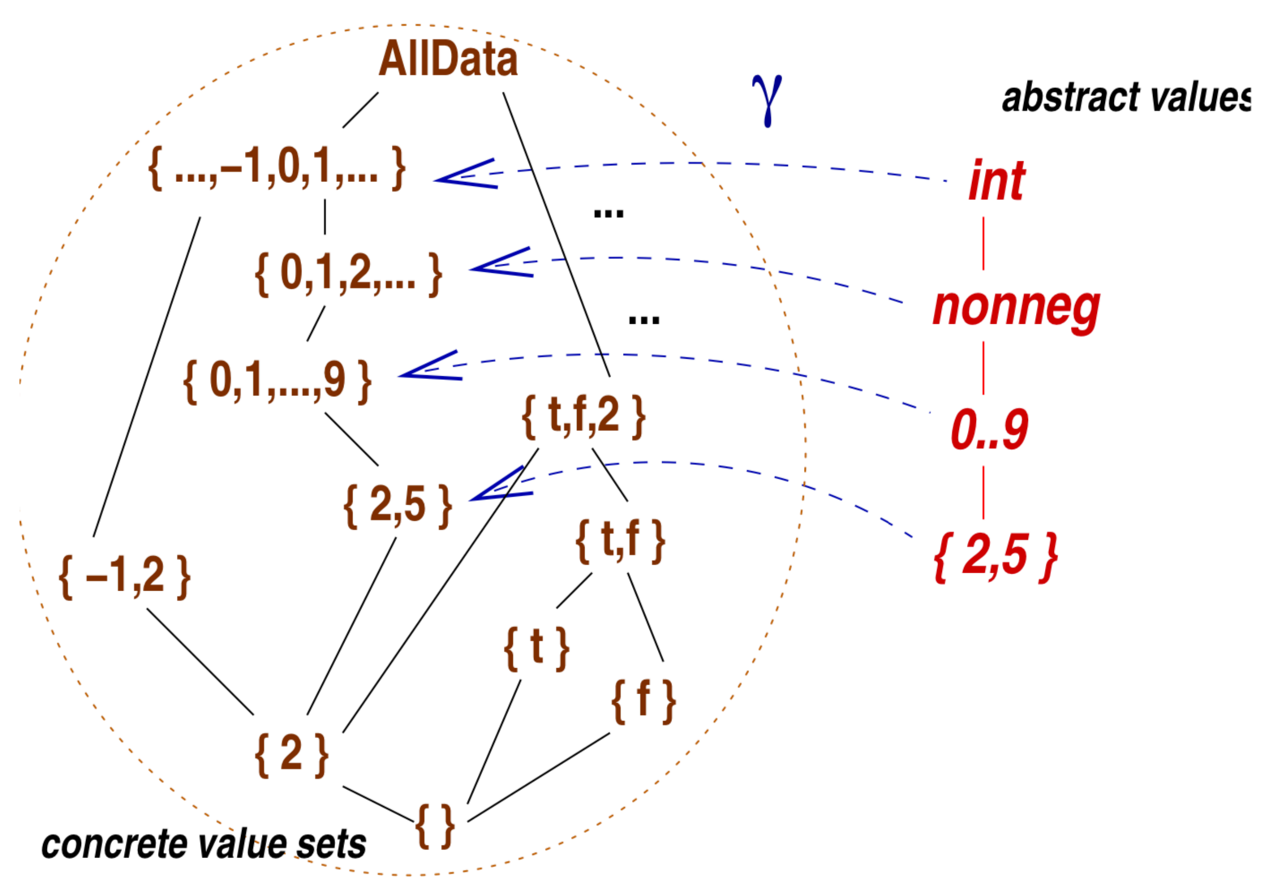
# Intuition I

$\alpha$ must be monotone



# Intuition II

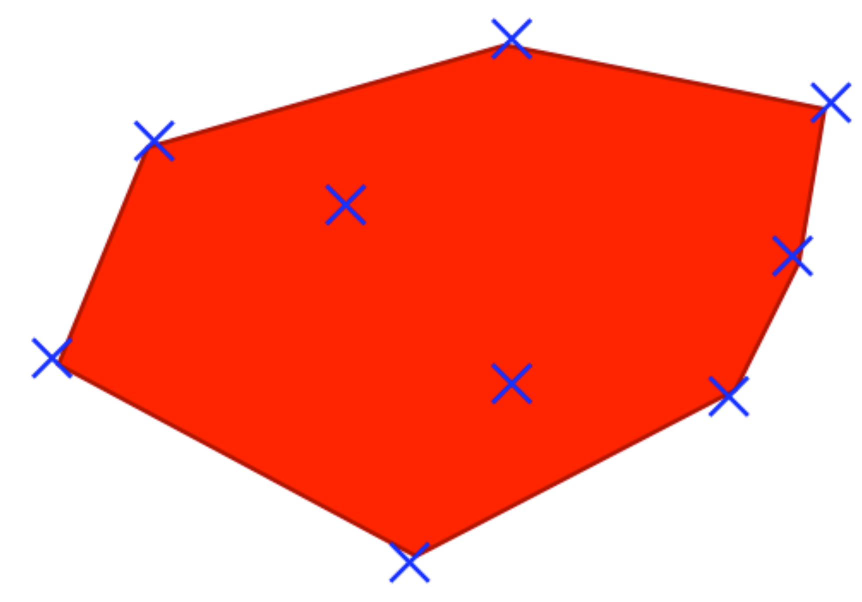
$\gamma$ must be monotone



# Abstracting a set of 2D points

concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

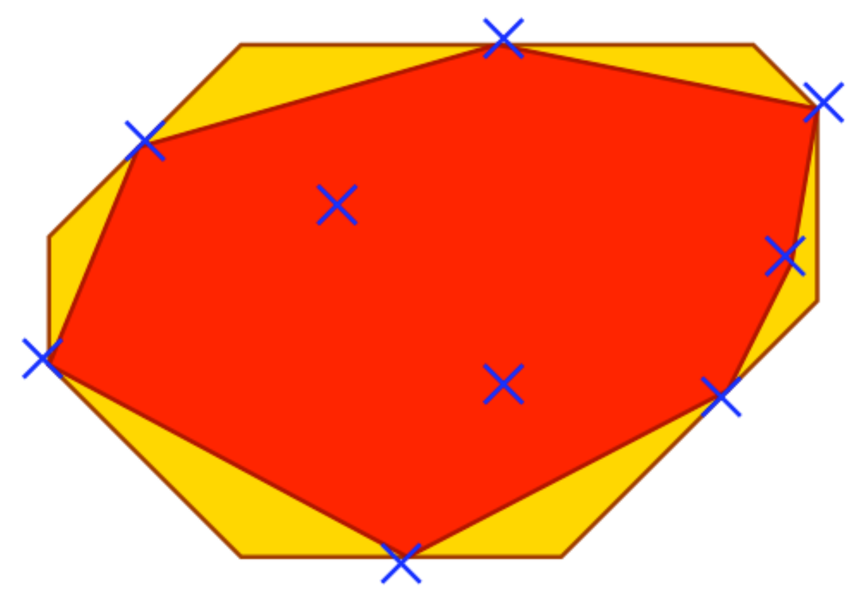
# Abstracting a set of 2D points



concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}^\sharp_p$: $6X + 11Y \geq 33 \wedge \cdots$

# Abstracting a set of 2D points

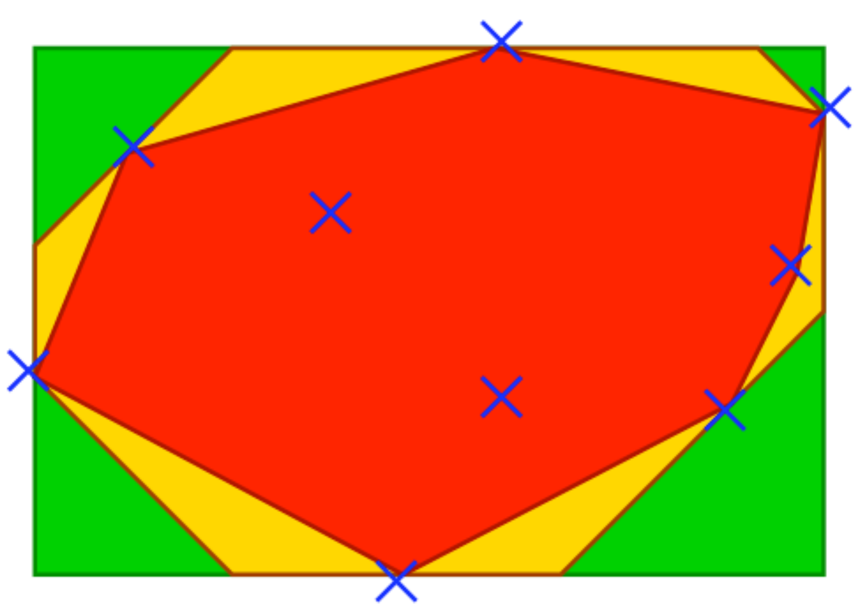


concrete sets $\mathcal{D}$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

abstract polyhedra $\mathcal{D}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

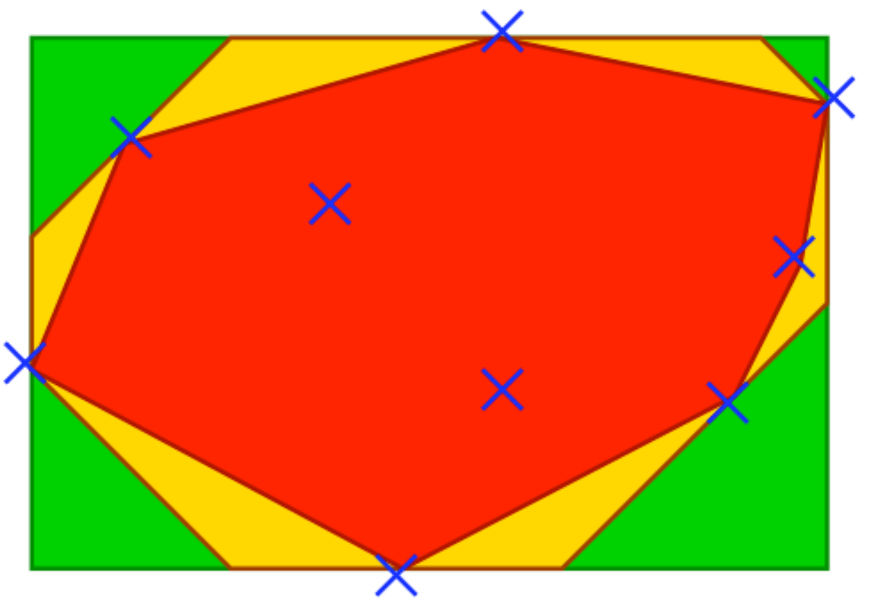
abstract octagons $\mathcal{D}_o^\sharp$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

# Abstracting a set of 2D points



| | |
|---|---|
| concrete sets $\mathcal{D}$: | $\{(0,3),(5.5,0),(12,7),\ldots\}$ |
| abstract polyhedra $\mathcal{D}_p^\sharp$: | $6X + 11Y \geq 33 \wedge \cdots$ |
| abstract octagons $\mathcal{D}_o^\sharp$: | $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$ |
| abstract intervals $\mathcal{D}_i^\sharp$: | $X \in [0,12] \wedge Y \in [0,8]$ |

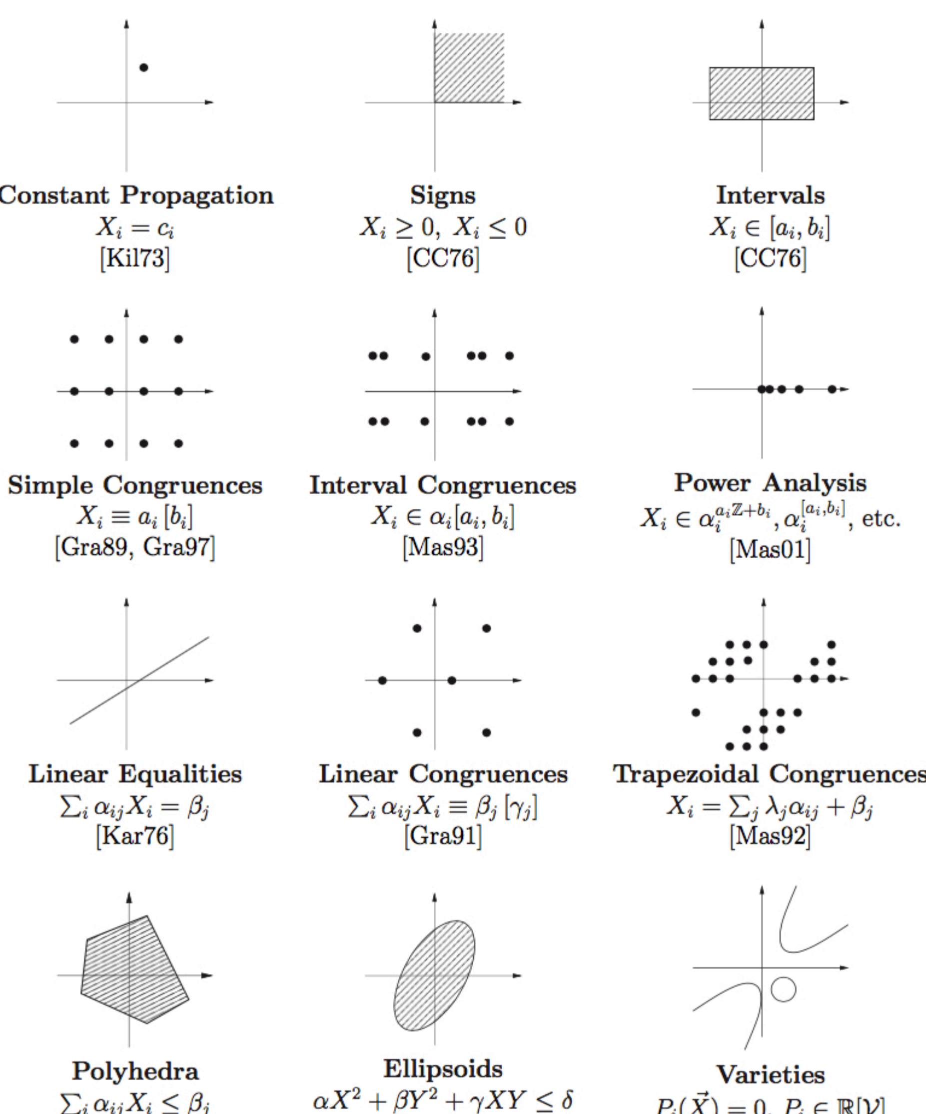
# Abstracting a set of 2D points



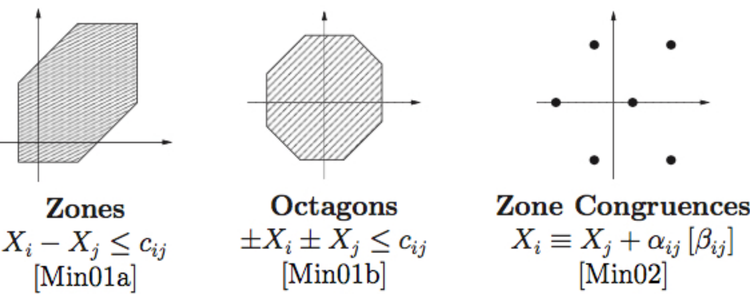
| | | |
|---|---|---|
| concrete sets $\mathcal{D}$: | $\{(0,3),(5.5,0),(12,7),\ldots\}$ | not computable |
| abstract polyhedra $\mathcal{D}^\sharp_p$: | $6X + 11Y \geq 33 \wedge \cdots$ | exponential cost |
| abstract octagons $\mathcal{D}^\sharp_o$: | $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$ | cubic cost |
| abstract intervals $\mathcal{D}^\sharp_i$: | $X \in [0,12] \wedge Y \in [0,8]$ | linear cost |

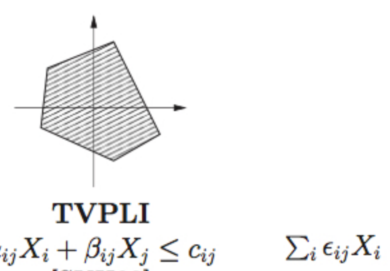
Trade-off between cost and expressiveness / precision

# Abstracting a set of 2D points



# Abstracting a set of 2D points




**Octahedra**

$\sum_i \epsilon_{ij} X_i \leq \beta_j, \ \epsilon_{ij} \in \{-1, 0, 1\}, \ X_i \geq 0$

[CC04]

# Abstract interpretation

Patrick Cousot

THESE

présentée à

**Université Scientifique et Médicale de Grenoble**
**Institut National Polytechnique de Grenoble**

par

Patrick COUSOT

METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE POINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

Président : L. BOLLIET

Examinateurs : C. BENZAKEN
Ph. JORRAND
B. LORHO
C. PAIR
F. ROBERT
M. SINTZOFF

## Abstract Interpretation

A formal technique used since 40 years (Patrick e Radhia Cousot, 1977) to systematically design abstractions and approximations

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

## 10. *Conclusion*

It is our feeling that most program analysis techniques may be understood as abstract interpretations of programs. Let us point out global data flow analysis in optimizing compilers (Kildall[73], Morel and Renvoise[76] , Schwartz[75], Ullman[75], Wegbreit[75], ...), type verification (Naur[65], ...), type discovery (Cousot[76'], Sintzoff[72], Tenenbaum[74], ...), program testing (Henderson [75], ...) symbolic evaluation of programs (Hewitt et al.[73], Karr[76], ...), program performance analysis (Wegbreit[76], ...), formalization of program semantics (Hoare and Lauer[74], Ligler[75], Manna and Shamir[75], ...), verification of program correctness (Floyd[67], Park[69], Sintzoff[75], ...), discovery of inductive invariants (Katz and Manna[76], ...), proofs of program termination (Sites[74], ...), program transformation (Sintzoff [76], ...), ...

There is a fundamental unity between all these apparently unrelated program analysis techniques : a new interpretation is given to the program text which allows to built an often implicit system of equations. The problem is either to verify that a solution provided by the user is correct, or to discover or approximate such a solution.

The mathematical model we studied in this paper is certainly the weakest which is necessary to unify these techniques, and therefore should be of very general scope. It can be considerably enriched for particular applications so that more powerful results may be obtained.

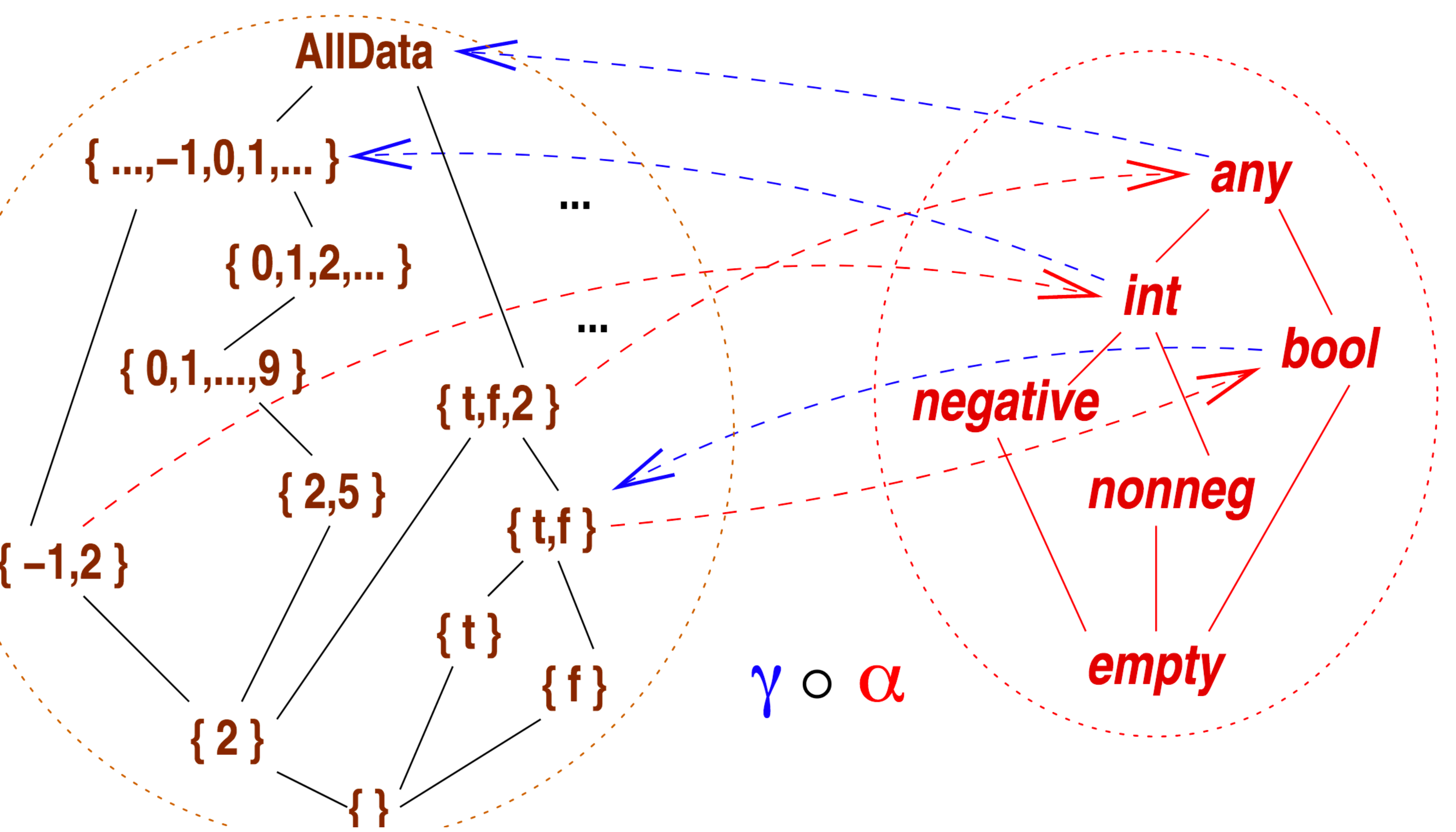
## Abstract Interpretation

- 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

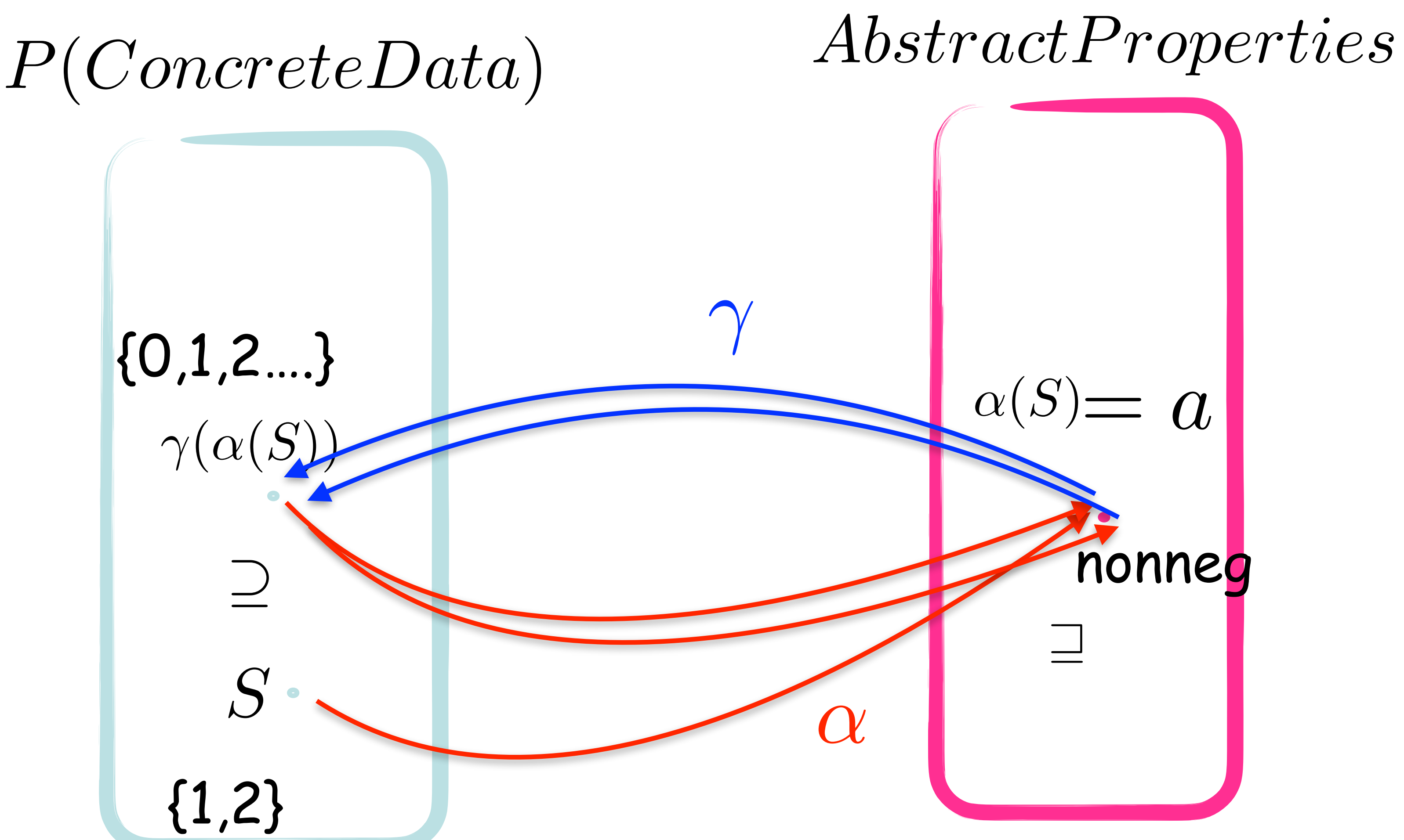
## The general idea

- 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 modeled by the abstract domain.
- The computation of the abstract semantics typically involves fixpoint computations

Relation between $\gamma$ and $\alpha$



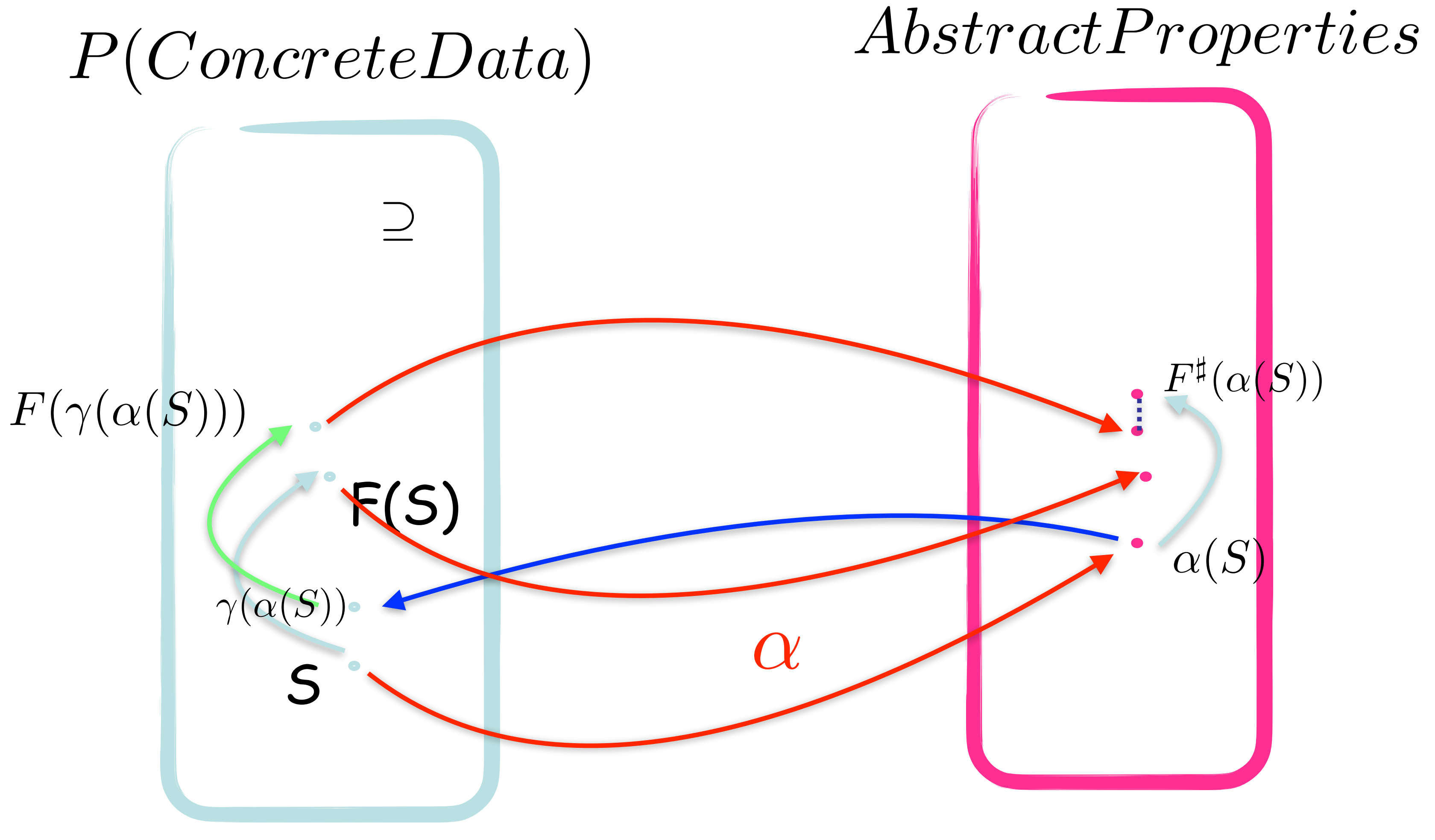
# **Galois insertions:** more properties than monotonicity




$\alpha$ and $\gamma$ are monotone

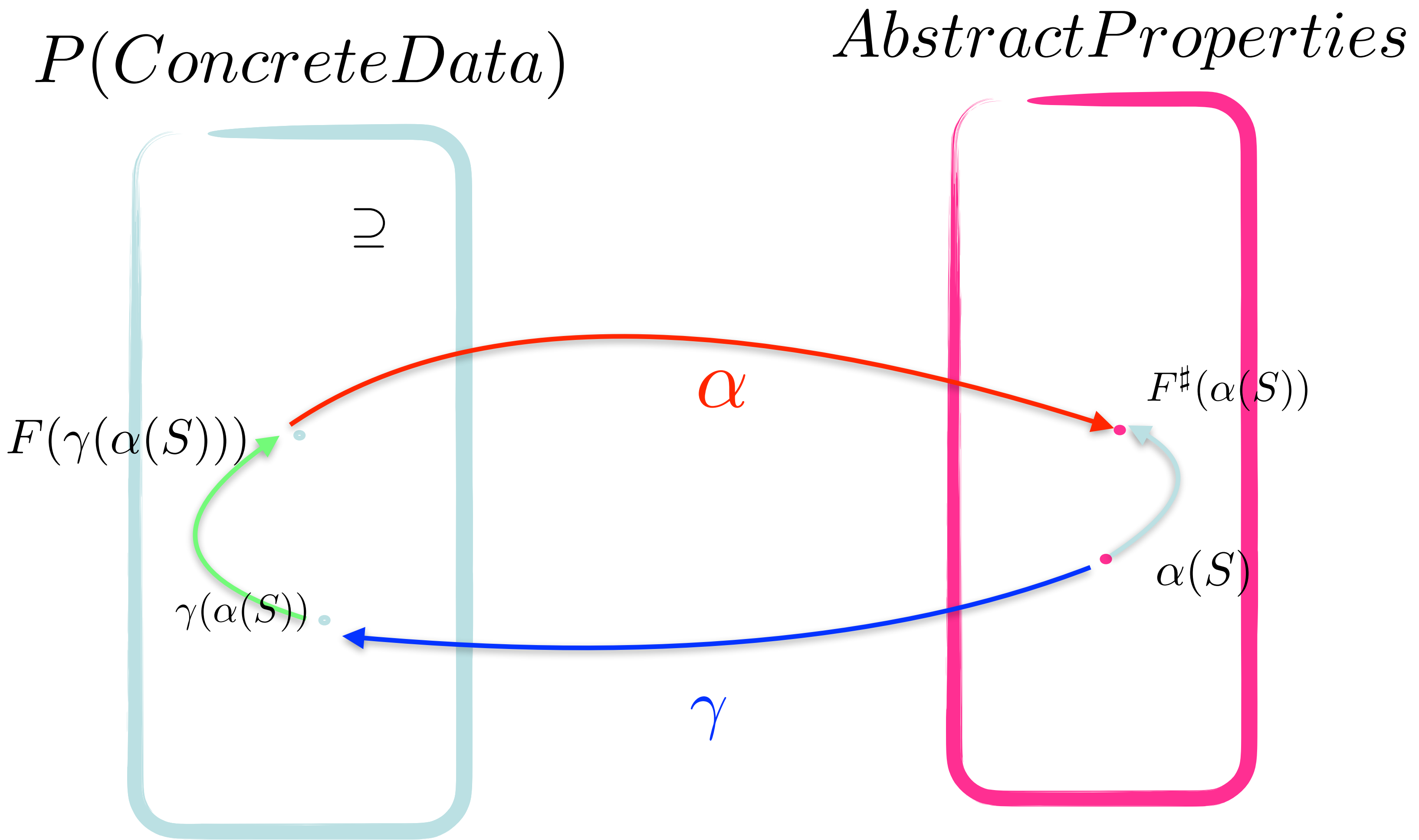
$S \subseteq \gamma \circ \alpha(S)$ $\alpha \circ \gamma(a) = a$

# Correct function abstraction: F monotone



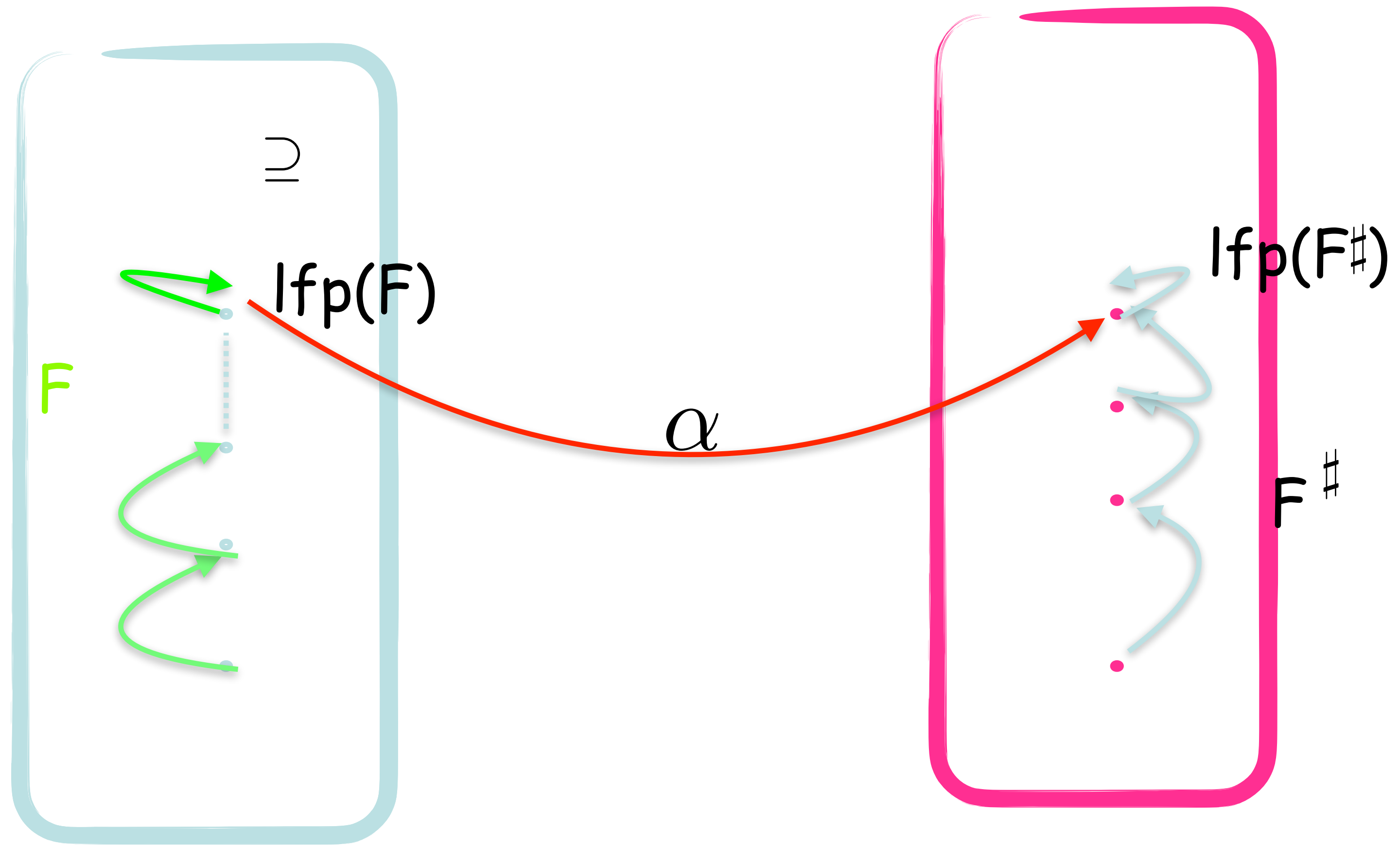

$F^\sharp \sqsupseteq \alpha \circ F \circ \gamma$

# **Optimal function** abstraction: F monotone




F♯ = $\alpha \circ F \circ \gamma$

# Fixpoint computation: F monotone




lfp( F ♯ ) is a correct approximation of $\alpha(lfp(F))$

that is lfp( F ♯ ) $\sqsupseteq \alpha(lfp(F))$

# Abstract interpretation

$(\mathcal{S}_0)$

```
assume X in [0,1000];
```

$(\mathcal{S}_1)$

```
I := 0;
```

$(\mathcal{S}_2)$

`while` $(\mathcal{S}_3)$ `I < X do`

  $(\mathcal{S}_4)$

```
    I := I + 2;
```

  $(\mathcal{S}_5)$

$(\mathcal{S}_6)$

program

# Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

$$\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$$

$$\begin{array}{lll}
\mathcal{S}_0 = \{\,(i,x) \mid i,x \in \mathbb{Z}\,\} & & = \top \\
\mathcal{S}_1 = \{\,(i,x) \in \mathcal{S}_0 \mid x \in [0,1000]\,\} & & = F_1(\mathcal{S}_0) \\
\mathcal{S}_2 = \{\,(0,x) \mid \exists i,\ (i,x) \in \mathcal{S}_1\,\} & & = F_2(\mathcal{S}_1) \\
\mathcal{S}_3 = \mathcal{S}_2 \cup \mathcal{S}_5 & & \\
\mathcal{S}_4 = \{\,(i,x) \in \mathcal{S}_3 \mid i < x\,\} & & = F_4(\mathcal{S}_3) \\
\mathcal{S}_5 = \{\,(i+2,x) \mid (i,x) \in \mathcal{S}_4\,\} & & = F_5(\mathcal{S}_4) \\
\mathcal{S}_6 = \{\,(i,x) \in \mathcal{S}_3 \mid i \geq x\,\} & & = F_6(\mathcal{S}_3)
\end{array}$$

program　　　　semantics

Concrete semantics $\mathcal{S}_i \in \mathcal{D} = \mathcal{P}(\{\mathtt{I},\mathtt{X}\} \to \mathbb{Z})$:

- strongest invariant (and an inductive invariant)
- not computable in general
- smallest solution of a system of equations

# Abstract interpretation

```
(S0)
assume X in [0,1000];
(S1)
I := 0;
(S2)
while (S3) I < X do
    (S4)
    I := I + 2;
    (S5)
(S6)
```

program

$\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$

$$\mathcal{S}_0^\sharp = \top^\sharp$$
$$\mathcal{S}_1^\sharp = F_1^\sharp(\mathcal{S}_0^\sharp)$$
$$\mathcal{S}_2^\sharp = F_2^\sharp(\mathcal{S}_1^\sharp)$$
$$\mathcal{S}_3^\sharp = \mathcal{S}_2^\sharp \cup^\sharp \mathcal{S}_5^\sharp$$
$$\mathcal{S}_4^\sharp = F_4^\sharp(\mathcal{S}_3^\sharp)$$
$$\mathcal{S}_5^\sharp = F_5^\sharp(\mathcal{S}_4^\sharp)$$
$$\mathcal{S}_6^\sharp = F_6^\sharp(\mathcal{S}_3^\sharp)$$

semantics

Abstract semantics $\mathcal{S}_i^\sharp \in \mathcal{D}^\sharp$:

- $\mathcal{D}^\sharp$ is a subset of properties of interest (approximation) with a machine representation
- $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ over-approximates the effect of $F : \mathcal{D} \to \mathcal{D}$ in $\mathcal{D}^\sharp$ (with effective algorithms)

# An example : a possible concrete semantics

```
x = 0; y = 0;

while (x <= 100) {
```

$(x, y) \in S \in \wp(\mathbb{Z}^2)$

```
 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

Concrete domain (sets of pairs of integers):

$$\langle \wp(\mathbb{Z}^2), \subseteq, \varnothing, \mathbb{Z}^2, \cup, \cap \rangle.$$

Concrete semantics (in fixpoint form):

$$S \stackrel{\text{def}}{=} \operatorname{lfp} \mathcal{F} = \mathcal{F}^{\omega}(\varnothing).$$

# A possible concrete semantics

```
x = 0; y = 0;
```

$\{(0, 0)\}$

```
while (x <= 100) {

 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

y

O

x

# A possible concrete semantics

```
x = 0; y = 0;
    {(0,0)}
while (x <= 100) {
    {(0,0)}
 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

y

O

x

# A possible concrete semantics

```
x = 0; y = 0;
```
$\{(0, 0)\}$
```
while (x <= 100) {
```
$\{(0, 0)\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{(2, 0)\}$
```
 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

y

O

x

# A possible concrete semantics

```
x = 0; y = 0;
```

$\{(0,0)\}$

```
while (x <= 100) {
```

$\{(0,0)\}$

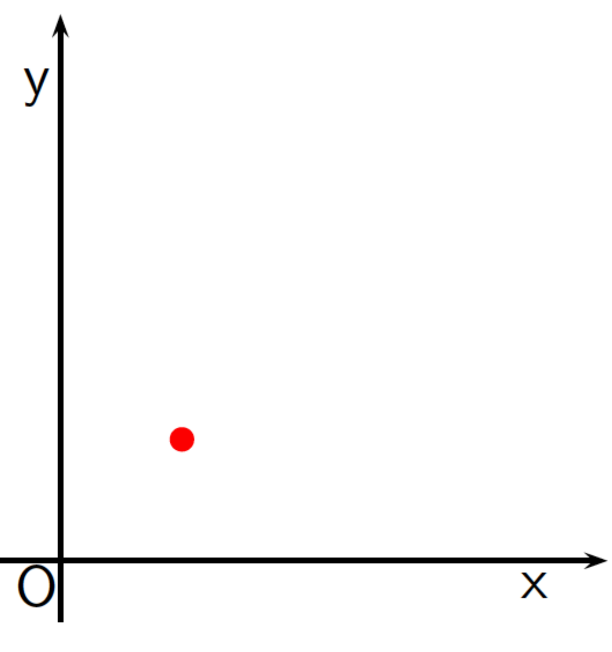
```
 b = foo(x, y);
 if (b) x = x+2
```

$\{(2,0)\}$

```
 else { x = x+1; y = y+1;
```

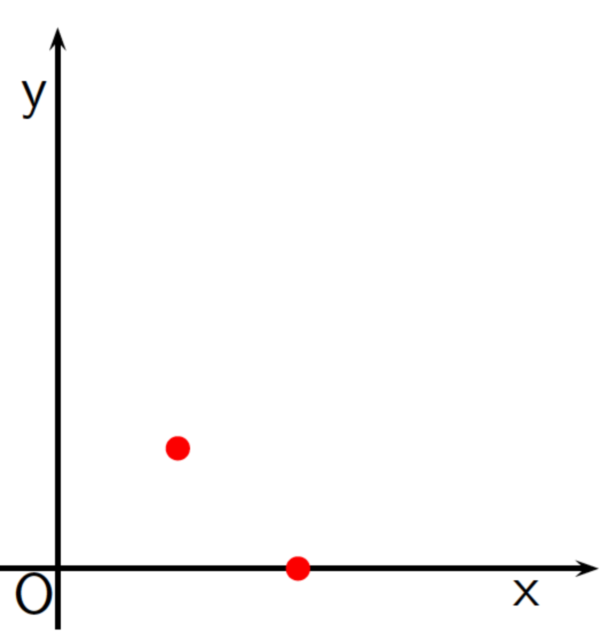
$\{(1,1)\}$

```
 } /* else */

} /* while */
```



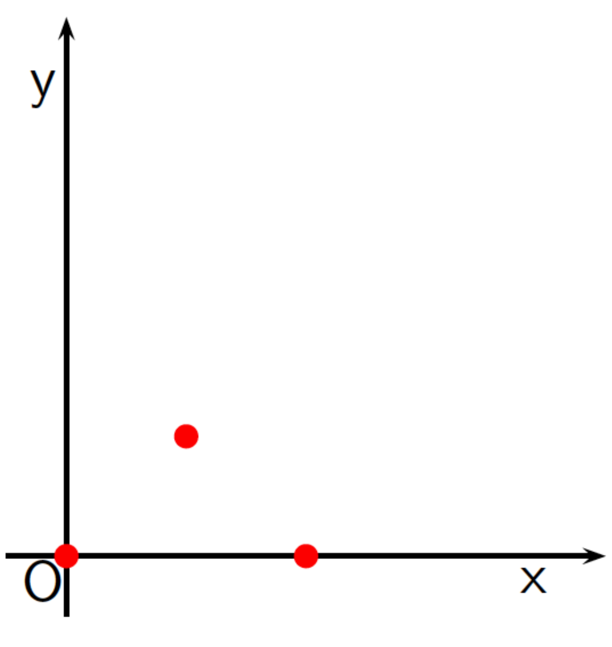
# A possible concrete semantics

```
x = 0; y = 0;
    {(0,0)}
while (x <= 100) {
    {(0,0)}
 b = foo(x, y);
 if (b) x = x+2
    {(2,0)}
 else { x = x+1; y = y+1;
    {(1,1)}
 } /* else */
    {(1,1),(2,0)}
} /* while */
```



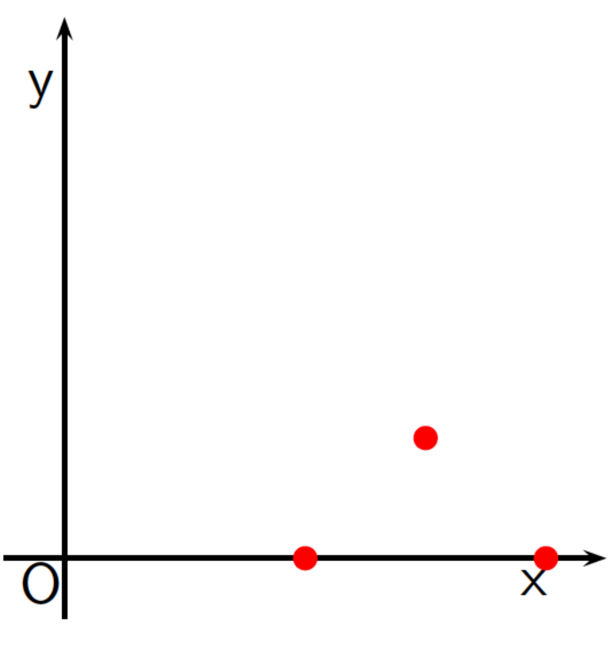
# A possible concrete semantics

```
x = 0; y = 0;
    {(0,0)}
while (x <= 100) {
    {(0,0),(1,1),(2,0)}
 b = foo(x, y);
 if (b) x = x+2
    {(2,0)}
 else { x = x+1; y = y+1;
    {(1,1)}
 } /* else */
    {(1,1),(2,0)}
} /* while */
```



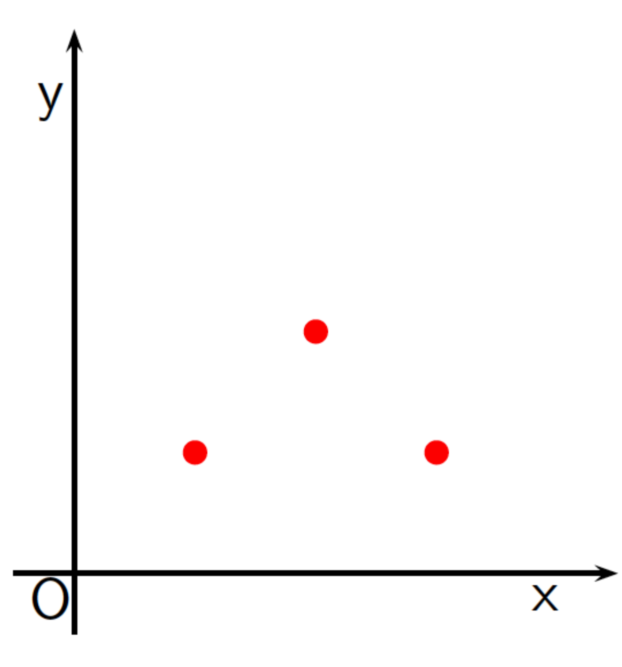
# A possible concrete semantics

```
x = 0; y = 0;
    {(0,0)}
while (x <= 100) {
    {(0,0),(1,1),(2,0)}
 b = foo(x, y);
 if (b) x = x+2
    {(2,0),(3,1),(4,0)}
 else { x = x+1; y = y+1;
    {(1,1)}
 } /* else */
    {(1,1),(2,0)}
} /* while */
```



# A possible concrete semantics

```
x = 0; y = 0;
    {(0, 0)}
while (x <= 100) {
    {(0, 0), (1, 1), (2, 0)}
  b = foo(x, y);
  if (b) x = x+2
    {(2, 0), (3, 1), (4, 0)}
  else { x = x+1; y = y+1;
    {(1, 1), (2, 2), (3, 1)}
  } /* else */
    {(1, 1), (2, 0)}
} /* while */
```



# A possible concrete semantics

```
x = 0; y = 0;
```
$\{(0,0)\}$
```
while (x <= 100) {
```
$\{(0,0),(1,1),(2,0)\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{(2,0),(3,1),(4,0)\}$
```
 else { x = x+1; y = y+1;
```
$\{(1,1),(2,2),(3,1)\}$
```
 } /* else */
```
$\{(1,1),(2,0),(2,2),(3,1),(4,0)\}$
```
} /* while */
```

y
O
x

# ….and so on…………….

```
x = 0; y = 0;
```

$$\{(0,0)\}$$

```
while (x <= 100) {
```

$$\{(0,0),(1,1),(2,0),(2,2),(3,1),(4,0)\}$$

```
 b = foo(x, y);
 if (b) x = x+2
```

$$\{(2,0),(3,1),(4,0)\}$$

```
 else { x = x+1; y = y+1;
```

$$\{(1,1),(2,2),(3,1)\}$$

```
 } /* else */
```

$$\{(1,1),(2,0),(2,2),(3,1),(4,0)\}$$

```
} /* while */
```

# A Possible Abstract Semantics

```
x = 0; y = 0;

while (x <= 100) {
```
$(x, y) \in \mathcal{Q} \in \mathbb{OS}_2$
```
 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

Abstract domain (bidimensional octagonal shapes):

$$\langle \mathbb{OS}_2, \subseteq, \varnothing, \mathbb{R}^2, \uplus, \cap \rangle.$$

Correctness:

$$X \subseteq \mathcal{P} \implies \mathcal{F}(X) \subseteq \mathcal{F}^\sharp(\mathcal{P}).$$

Abstract semantics:

$$\mathcal{Q} \in \mathrm{postfp}(\mathcal{F}^\sharp).$$

# A Possible Abstract Semantics

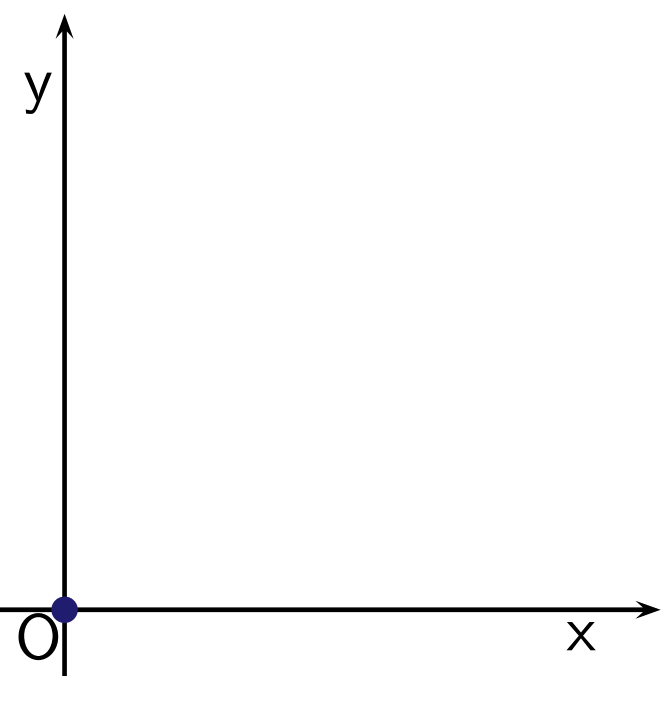
```
x = 0; y = 0;

while (x <= 100) {

 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

# A Possible Abstract Semantics

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

```
while (x <= 100) {

 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

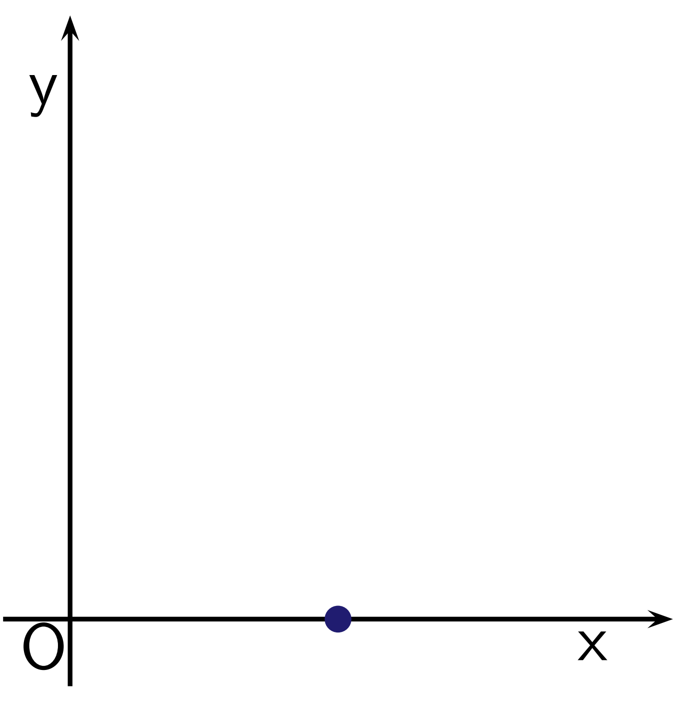
```
while (x <= 100) {
```

$\{x = 0, y = 0\}$

```
 b = foo(x, y);
 if (b) x = x+2

 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```

# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{x = 0, y = 0\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{x = 2, y = 0\}$
```
 else { x = x+1; y = y+1;

 } /* else */

} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

```
while (x <= 100) {
```

$\{x = 0, y = 0\}$

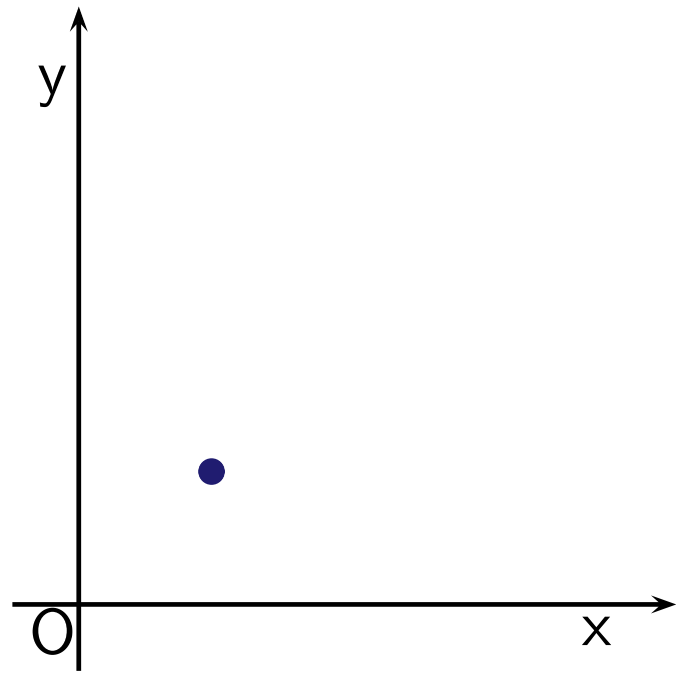
```
 b = foo(x, y);
 if (b) x = x+2
```

$\{x = 2, y = 0\}$

```
 else { x = x+1; y = y+1;
```

$\{x = 1, y = 1\}$

```
 } /* else */

} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

```
while (x <= 100) {
```

$\{x = 0, y = 0\}$

```
 b = foo(x, y);
 if (b) x = x+2
```

$\{x = 2, y = 0\}$

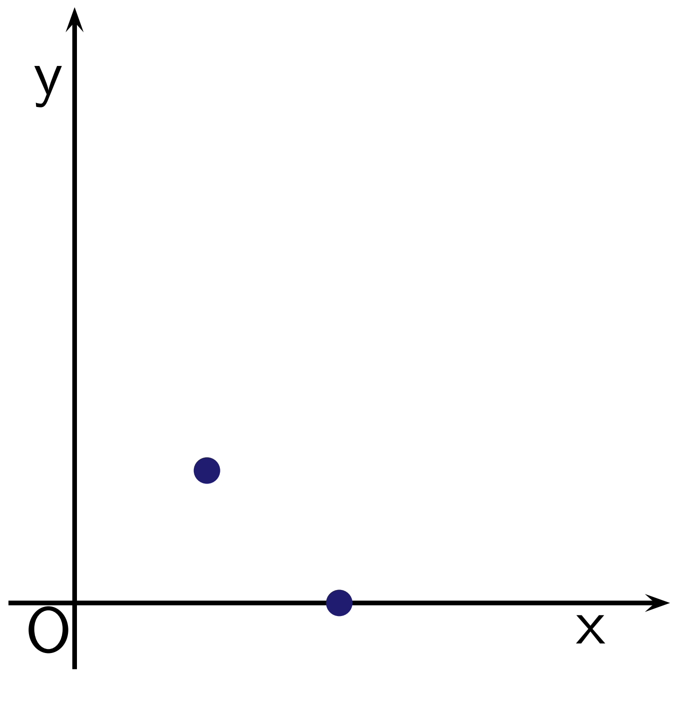
```
 else { x = x+1; y = y+1;
```

$\{x = 1, y = 1\}$

```
 } /* else */
```

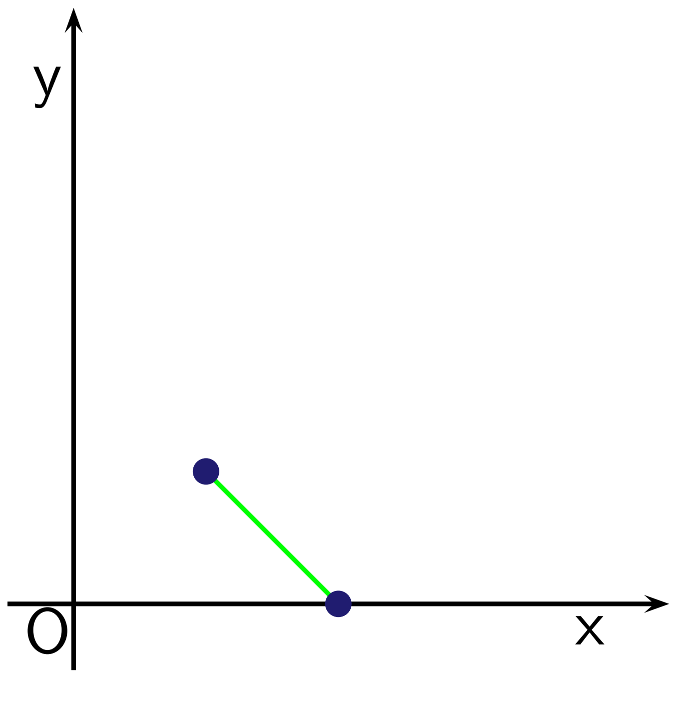
$\{x = 2, y = 0\} \uplus \{x = 1, y = 1\}$

```
} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{x = 0, y = 0\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{x = 2, y = 0\}$
```
 else { x = x+1; y = y+1;
```
$\{x = 1, y = 1\}$
```
 } /* else */
```
$\{1 \leq x \leq 2, x + y = 2\}$
```
} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$

```
while (x <= 100) {
```
$\{x = 0, y = 0\}$

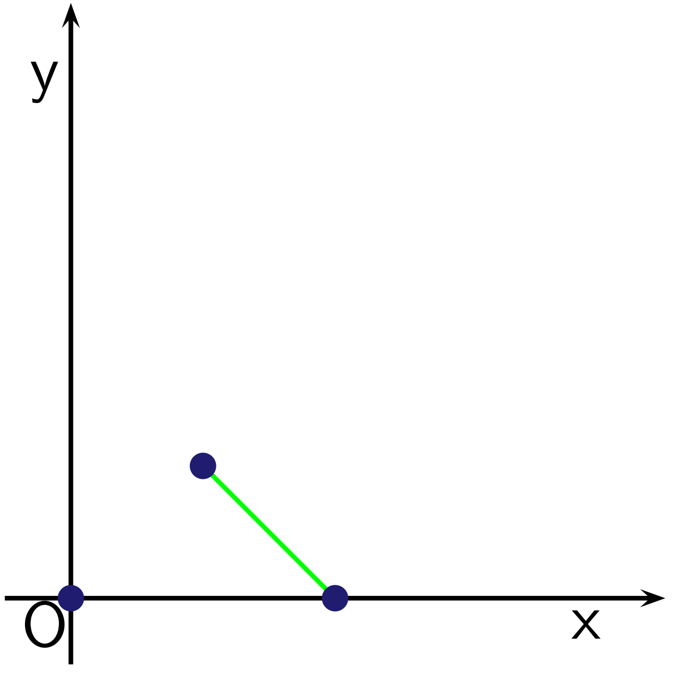
$\uplus \{1 \leq x \leq 2, x + y = 2\}$

```
 b = foo(x, y);
 if (b) x = x+2
```
$\{x = 2, y = 0\}$

```
 else { x = x+1; y = y+1;
```
$\{x = 1, y = 1\}$

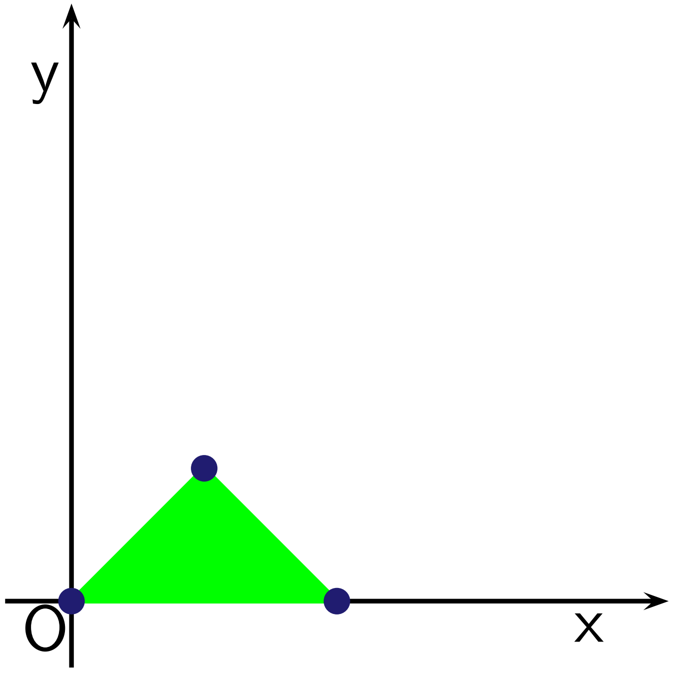
```
 } /* else */
```
$\{1 \leq x \leq 2, x + y = 2\}$

```
} /* while */
```



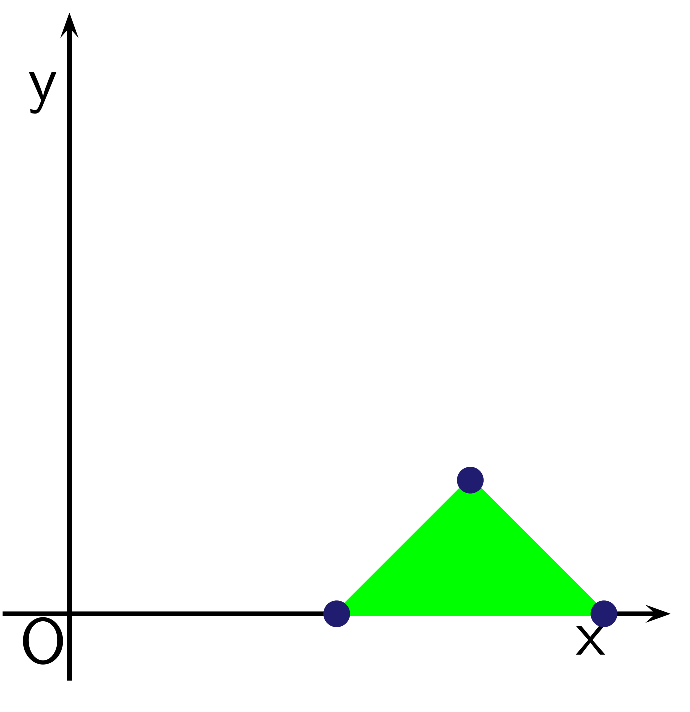
# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{0 \leq y \leq x, x + y \leq 2\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{x = 2, y = 0\}$
```
 else { x = x+1; y = y+1;
```
$\{x = 1, y = 1\}$
```
 } /* else */
```
$\{1 \leq x \leq 2, x + y = 2\}$
```
} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{0 \leq y \leq x, x + y \leq 2\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
```
 else { x = x+1; y = y+1;
```
$\{x = 1, y = 1\}$
```
 } /* else */
```
$\{1 \leq x \leq 2, x + y = 2\}$
```
} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

```
while (x <= 100) {
```

$\{0 \leq y \leq x, x + y \leq 2\}$

```
 b = foo(x, y);
 if (b) x = x+2
```

$\{0 \leq y \leq x - 2, x + y \leq 4\}$

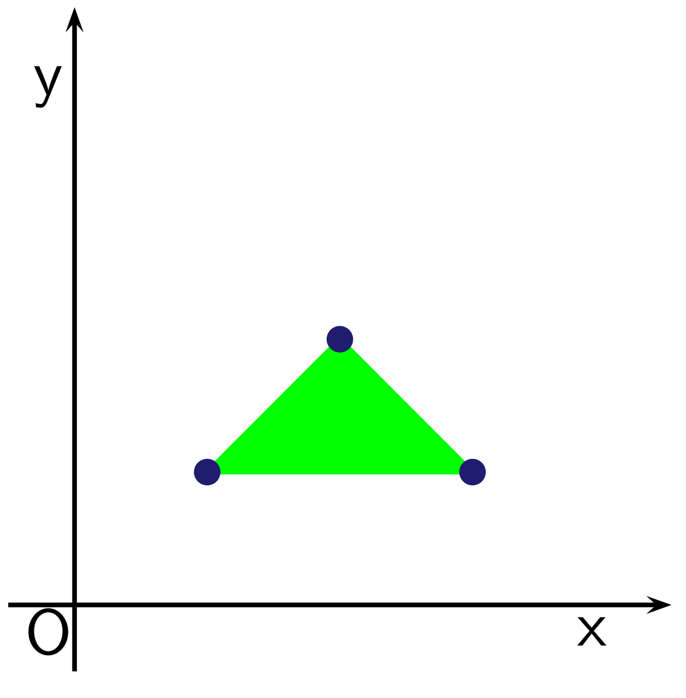
```
 else { x = x+1; y = y+1;
```

$\{1 \leq y \leq x, x + y \leq 4\}$

```
 } /* else */
```

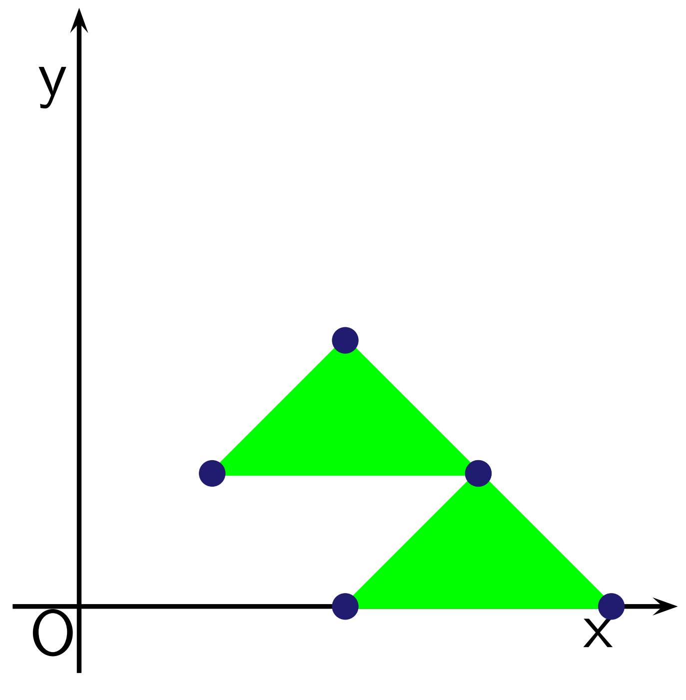
$\{1 \leq x \leq 2, x + y = 2\}$

```
} /* while */
```



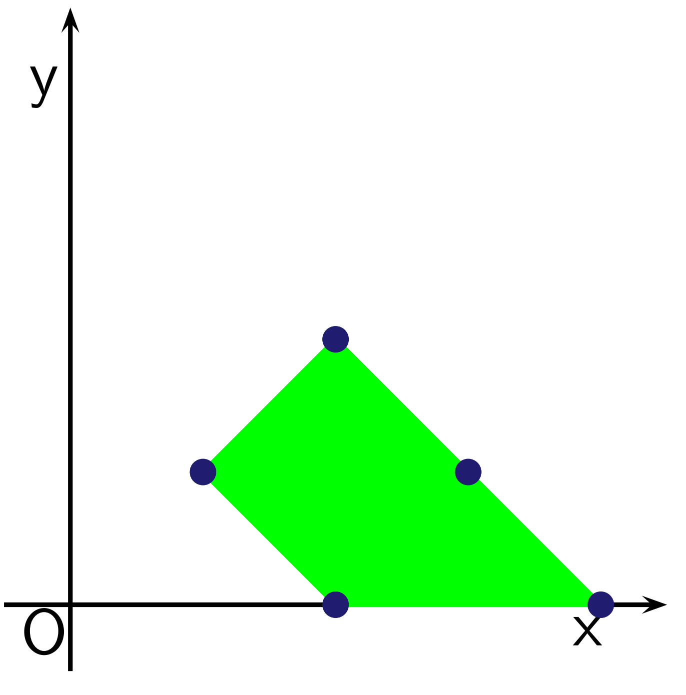
# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{0 \leq y \leq x, x + y \leq 2\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
```
 else { x = x+1; y = y+1;
```
$\{1 \leq y \leq x, x + y \leq 4\}$
```
 } /* else */
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
$\uplus \{1 \leq x \leq 2, x + y = 2\}$
```
} /* while */
```



# A Possible Abstract Semantics

```
x = 0; y = 0;
```
$\{x = 0, y = 0\}$
```
while (x <= 100) {
```
$\{0 \leq y \leq x, x + y \leq 2\}$
```
 b = foo(x, y);
 if (b) x = x+2
```
$\{0 \leq y \leq x - 2, x + y \leq 4\}$
```
 else { x = x+1; y = y+1;
```
$\{1 \leq y \leq x, x + y \leq 4\}$
```
 } /* else */
```
$\{0 \leq y \leq x, 2 \leq x + y \leq 4\}$
```
} /* while */
```



# ...and so on?

widening! we talk about it later!

```
x = 0; y = 0;
```

$\{x = 0, y = 0\}$

```
while (x <= 100) {
```

$\{0 \leq y \leq x, x + y \leq 4\}$

```
 b = foo(x, y);
 if (b) x = x+2
```

$\{0 \leq y \leq x - 2, x + y \leq 4\}$

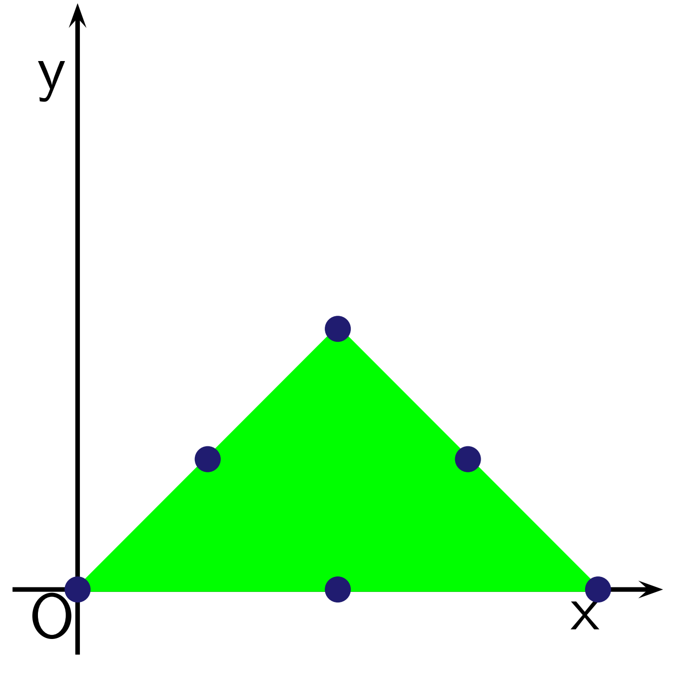
```
 else { x = x+1; y = y+1;
```

$\{1 \leq y \leq x, x + y \leq 4\}$

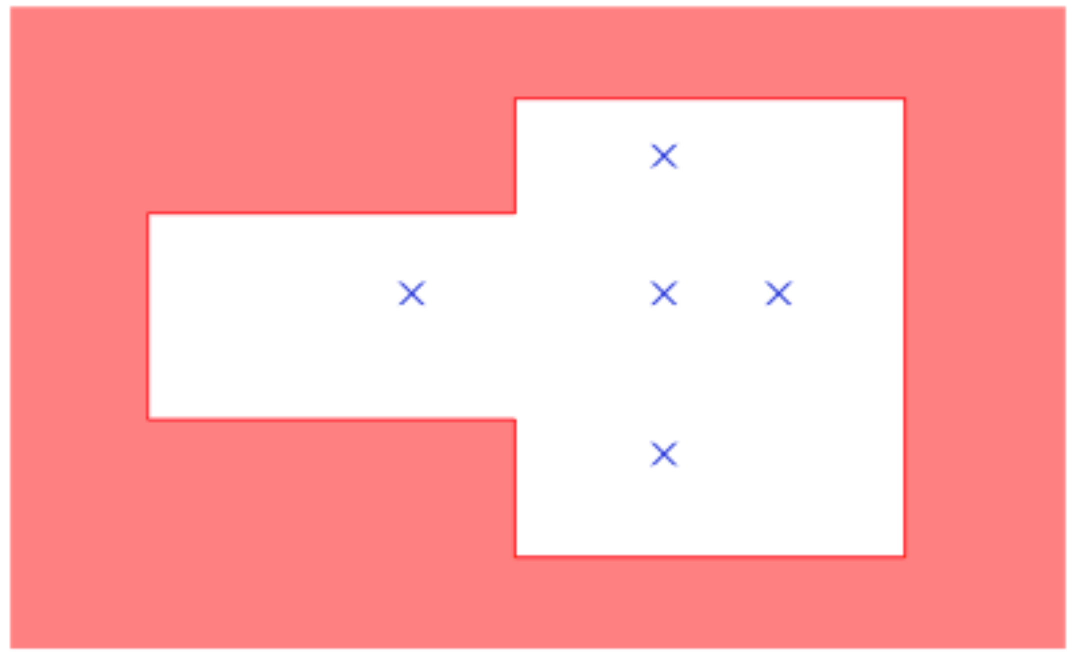
```
 } /* else */
```

$\{0 \leq y \leq x, 2 \leq x + y \leq 4\}$

```
} /* while */
```

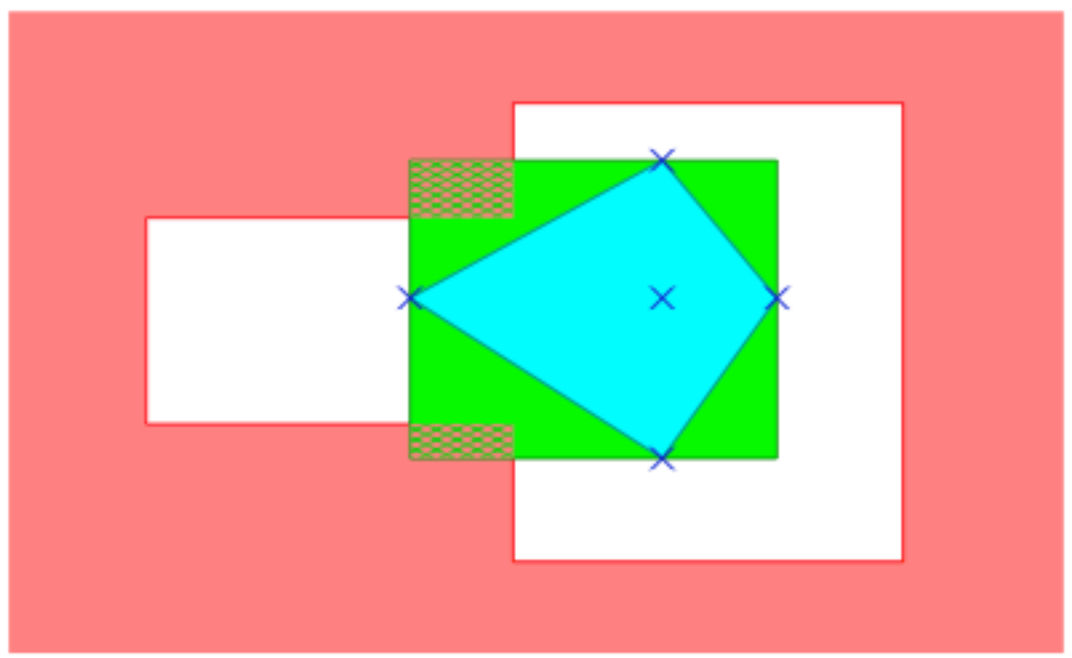


# Correctness proof and false alarms



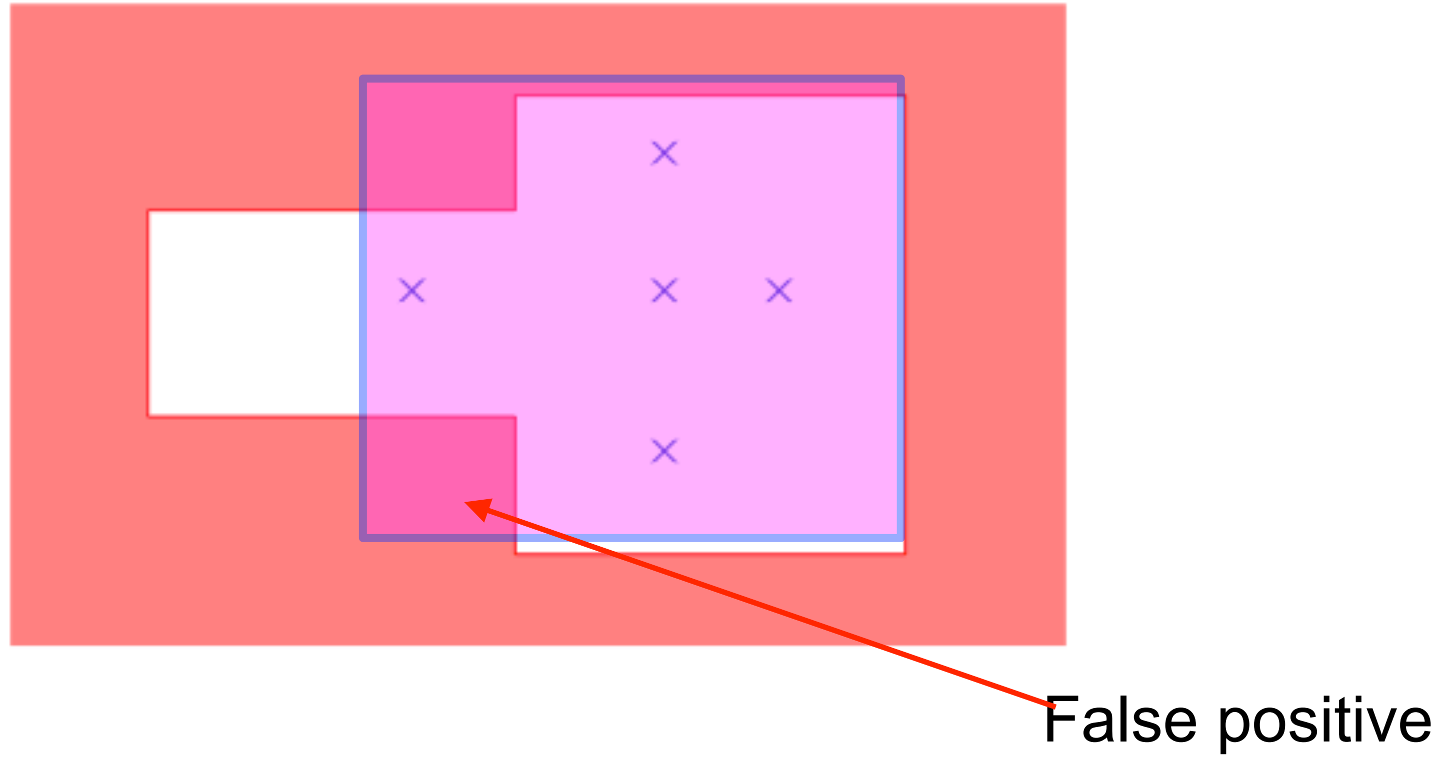
The program is correct (blue $\cap$ red $= \emptyset$).

# Correctness proof and false alarms



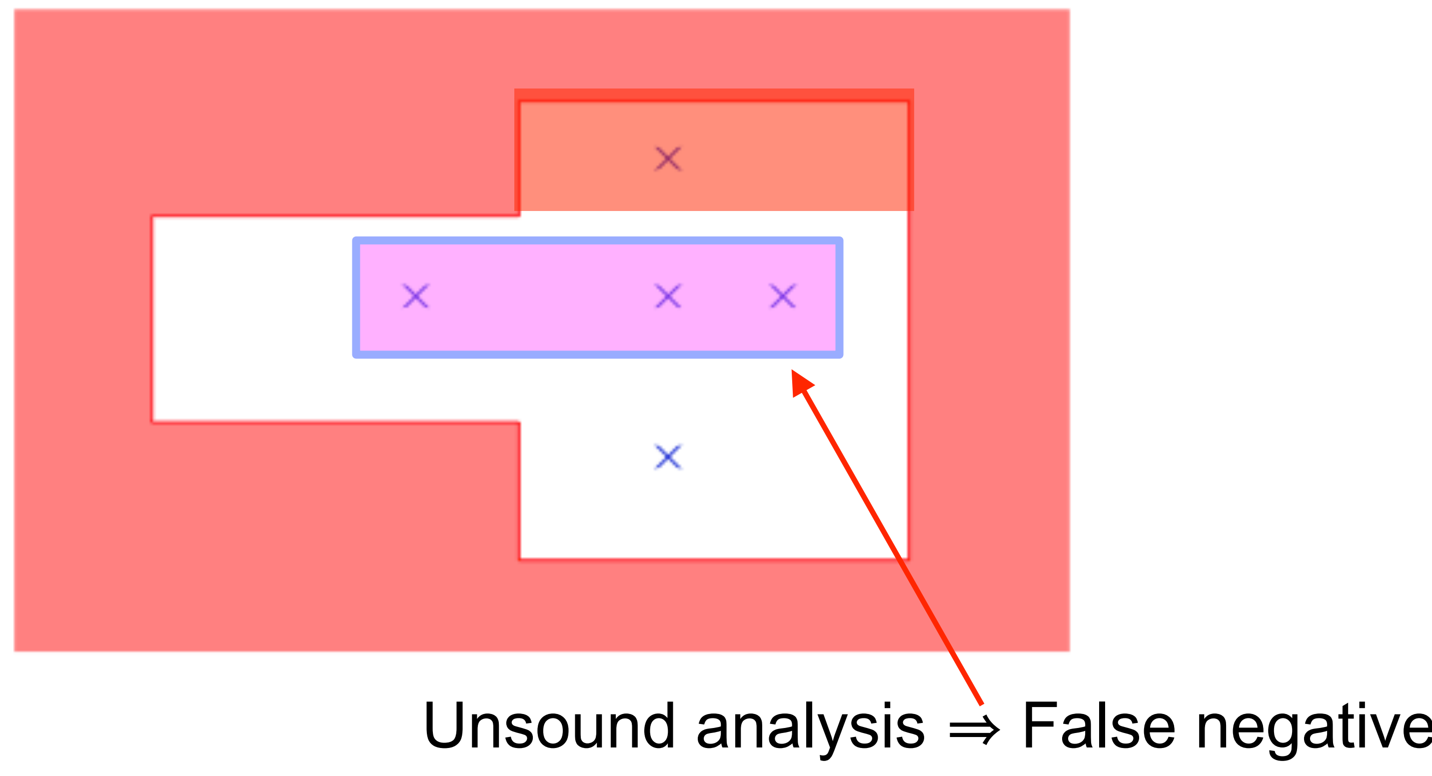
The program is correct (blue $\cap$ red $= \emptyset$).
The polyhedra domain can prove the correctness (cyan $\cap$ red $= \emptyset$).
The interval domain cannot (green $\cap$ red $\neq \emptyset$, false alarm).

# Correctness proof and false alarms




The program is correct ($\text{blue} \cap \text{red} = \emptyset$).

# Correctness proof and false alarms

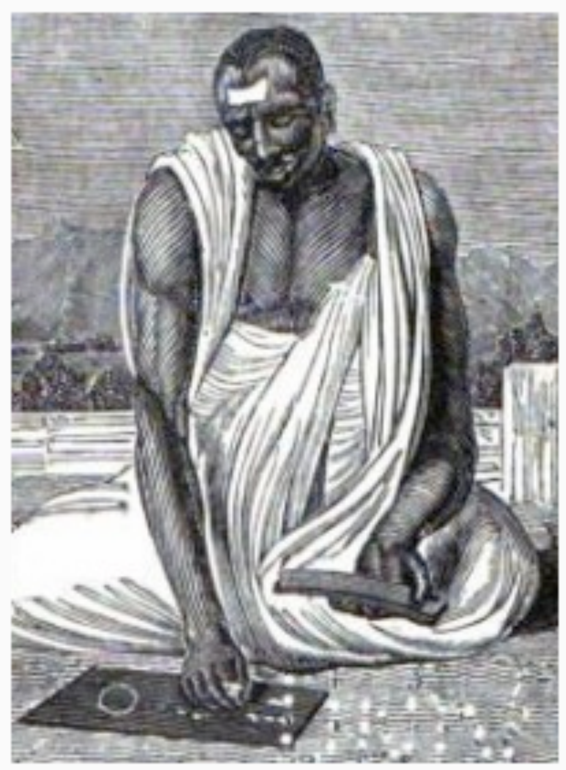



The program is bugged

# Brahmagupta

**Brahmagupta** (Sanskrit: ब्रह्मगुप्त; (598–c.670 CE) was an Indian mathematician and astronomer who wrote two important works on Mathematics and Astronomy: the *Brāhmasphuṭasiddhānta* (Extensive Treatise of Brahma) (628), a theoretical treatise, and the *Khaṇḍakhādyaka*, a more practical text.

**Brahmagupta**



| | |
|---|---|
| **Born** | 598 CE |
| **Died** | c.670 CE |
| **Fields** | Mathematics, Astronomy |
| **Known for** | Zero, modern Number system |

# The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative; [...]

18.32. A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.

18.33. The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.

18.34. A positive divided by a positive or a negative divided by a negative is positive; a zero divided by a zero is zero; a positive divided by a negative is negative; a negative divided by a positive is [also] negative.

wrong

# Concrete semantics

- Let us begin with basic expressions that only allow to multiply integer numbers.
  - **Exp** $\ni$ e ::= n | e*e
- The concrete semantics of these expressions is given by a function $\mathcal{S}$ defined as follows:
  - $\mathcal{S}$ : **Exp** $\rightarrow$ Z
  - $\mathcal{S}(n) = n$
  - $\mathcal{S}(e_1 * e_2) = \mathcal{S}(e_1) \times \mathcal{S}(e_2)$

# Abstract semantics

We consider an abstract semantics σ that computes the **sign** of expressions:

$$\sigma : \mathbf{Exp} \rightarrow \{-,0,+\}$$

$$\sigma(n) = \begin{cases} - & \text{if } n<0 \\ 0 & \text{if } n=0 \\ + & \text{if } n>0 \end{cases}$$

$$\sigma(e_1 * e_2) = \sigma(e_1) \times^a \sigma(e_2)$$

| $\times^a$ | - | 0 | + |
|---|---|---|---|
| - | + | 0 | - |
| 0 | 0 | 0 | 0 |
| + | - | 0 | + |

# Correctness

- Let us check that this abstract semantics σ is correct (or sound), namely, it correctly computes the sign of expressions.

- This simple proof relies on a structural induction on the expression e and exploits some basic properties of integer multiplications (a product of positive numbers is positive, etc.).

For any expression $e \in$ **Exp**:

$$\sigma(e) = - \;\Rightarrow\; \mathcal{S}(e) < 0$$

$$\sigma(e) = 0 \;\Rightarrow\; \mathcal{S}(e) = 0$$

$$\sigma(e) = + \;\Rightarrow\; \mathcal{S}(e) > 0$$

# A different perspective

Let us map each abstract value in {-,0,+} to the set of concrete values that it represents:

$$\gamma : \{-,0,+\} \rightarrow \mathcal{P}(Z)$$

$$\gamma(-) = \{x \in Z \mid x < 0\}$$

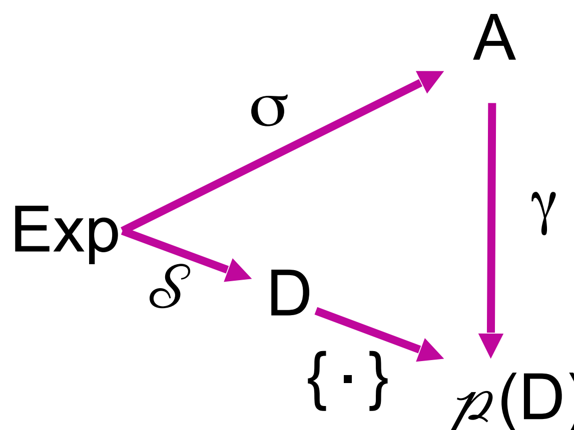
$$\gamma(0) = \{0\}$$

$$\gamma(+) = \{x \in Z \mid x > 0\}$$

# Concretization

- The concretization function $\gamma$ maps an abstract value to a set of concrete values
- Let us denote the concrete domain by D and the abstract domain by *A*

$$\mathcal{S}(e) \in \gamma(\sigma(e))$$



## Abstract interpretation

- We have specified a naive abstract interpretation.
  - Abstract computations happen in an abstract domain
  - In this case, the abstract domain is {+,0,-}.

- The abstract semantics is correct, meaning that $\sigma$ is a correct approximation of the concrete semantics $\mathcal{S}$: $\{\mathcal{S}(e)\} \subseteq \gamma(\sigma(e))$

- The concretization map establishes the relationship between the notions of approximation in the concrete and abstract domains

## Adding -

- Let us add to expressions the unary operator that changes the sign of an expression

$$\mathbf{Exp} \ni e ::= n \mid e*e \mid -e$$

$$\mathcal{S}(-e) = -\mathcal{S}(e)$$

$$\sigma(-e) = -^a\sigma(e) \quad \text{where} \quad -^a(-) = +,\ -^a(0) = 0,\ -^a(+) = -$$

| $-^a(-) = +,\ -^a(0) = 0,$ <br> $-^a(+) = -$ |
|---|

# Adding now the top element

We add a new abstract value $\top$ that represents any integer number

$$\gamma(\top) = Z$$

| $x^a$ | - | 0 | + | T |
|---|---|---|---|---|
| - | + | 0 | - | T |
| 0 | 0 | 0 | 0 | 0 |
| + | - | 0 | + | T |
| T | T | 0 | T | T |

# We add the + operator

We can now add the + operator

| +a | - | 0 | + | ⊤ |
| --- | --- | --- | --- | --- |
| - | - | - | ⊤ | ⊤ |
| 0 | - | 0 | + | ⊤ |
| + | ⊤ | + | + | ⊤ |
| ⊤ | ⊤ | ⊤ | ⊤ | ⊤ |

# Examples

- In some cases we observe a **loss of information** in the abstract semantics

$\mathcal{S}((1+2) - 3) = 0$

$\sigma((1+2)+ -3) = (+ +^a +) +^a - = + +^a - = T$

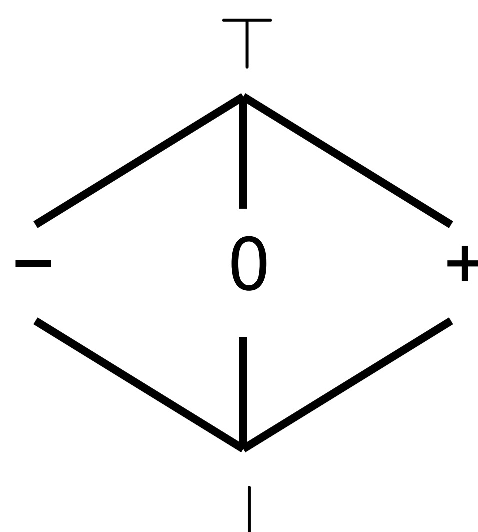
- In some cases this loss of information does not happen

$\mathcal{S}((5*4) + 6) = 26$

$\sigma((5*4) + 6) = (+ x^a +) +^a + = + +^a + = +$

# The abstract domain

- The abstract domain is a poset whose partial order represents the notion of approximation/ precision
- The partial order must be coherent with the concretization map:
  $$x \leq y \Leftrightarrow \gamma(x) \subseteq \gamma(y)$$
- Each subset of the abstract domain has lub and glb: it is a (complete) lattice

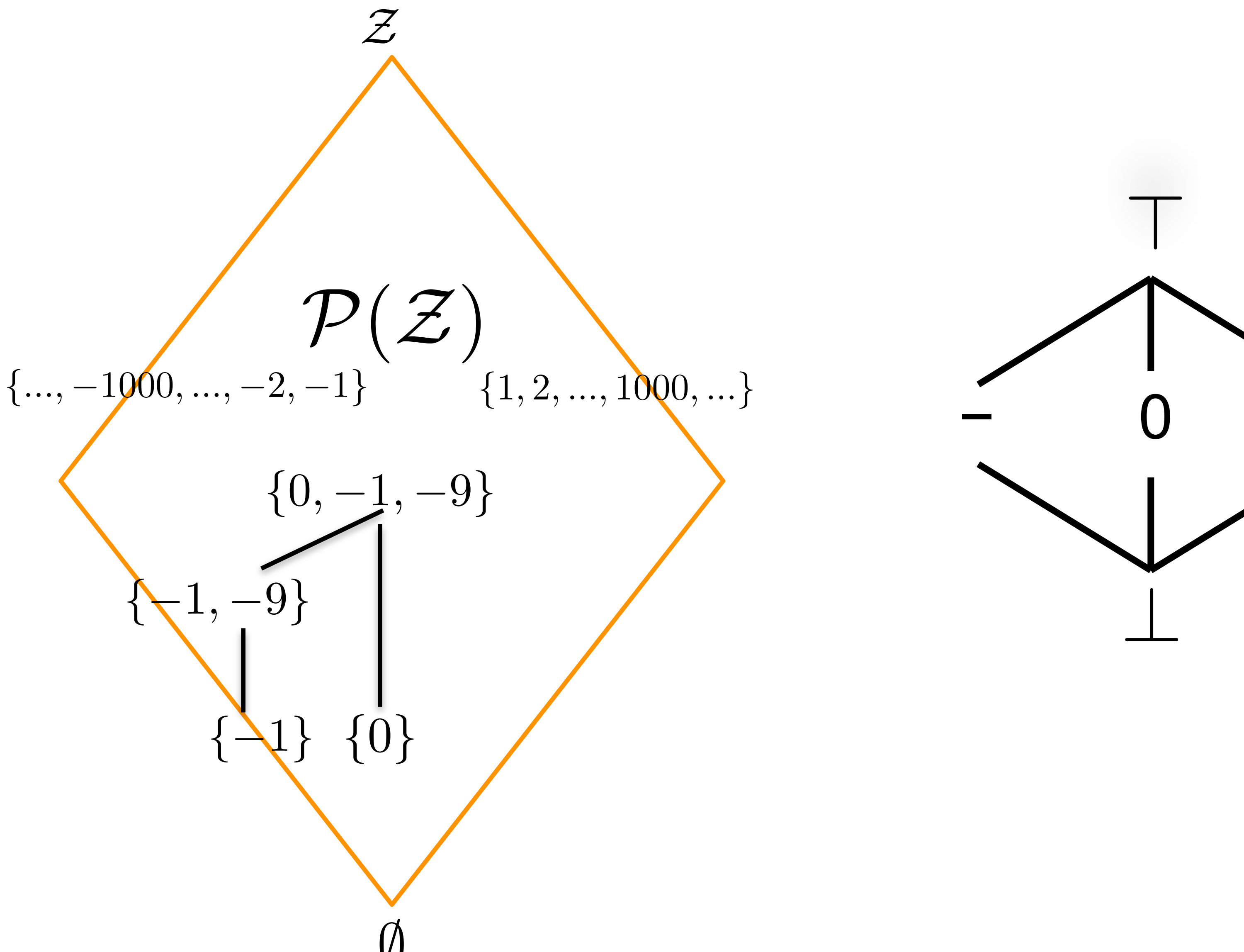


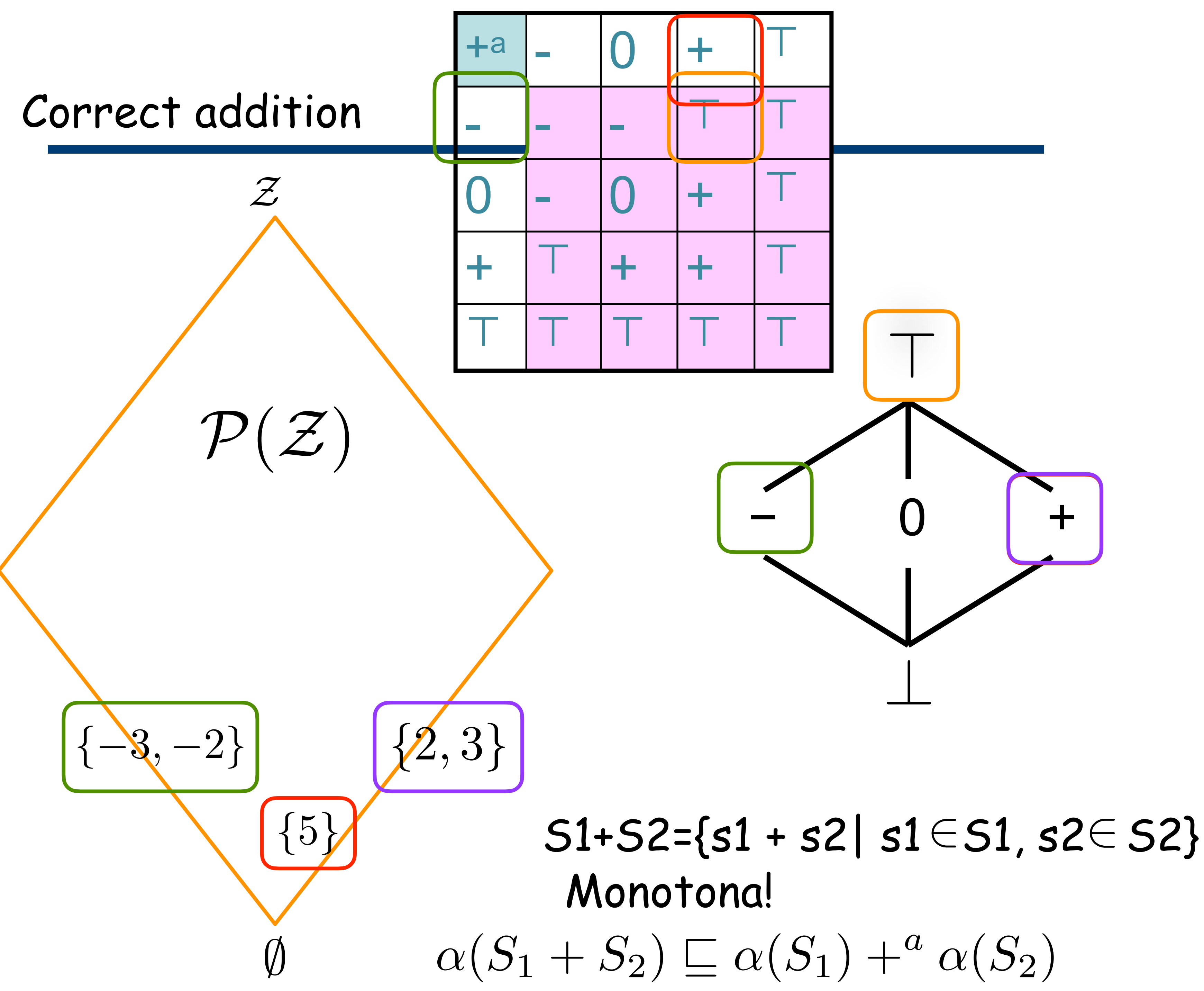
# The abstraction function

- The abstraction function $\alpha$ is the counterpart of the concretization map $\gamma$.
- The function $\alpha$ maps a set S of concrete values into the most precise abstract value that represents S.
- In our example

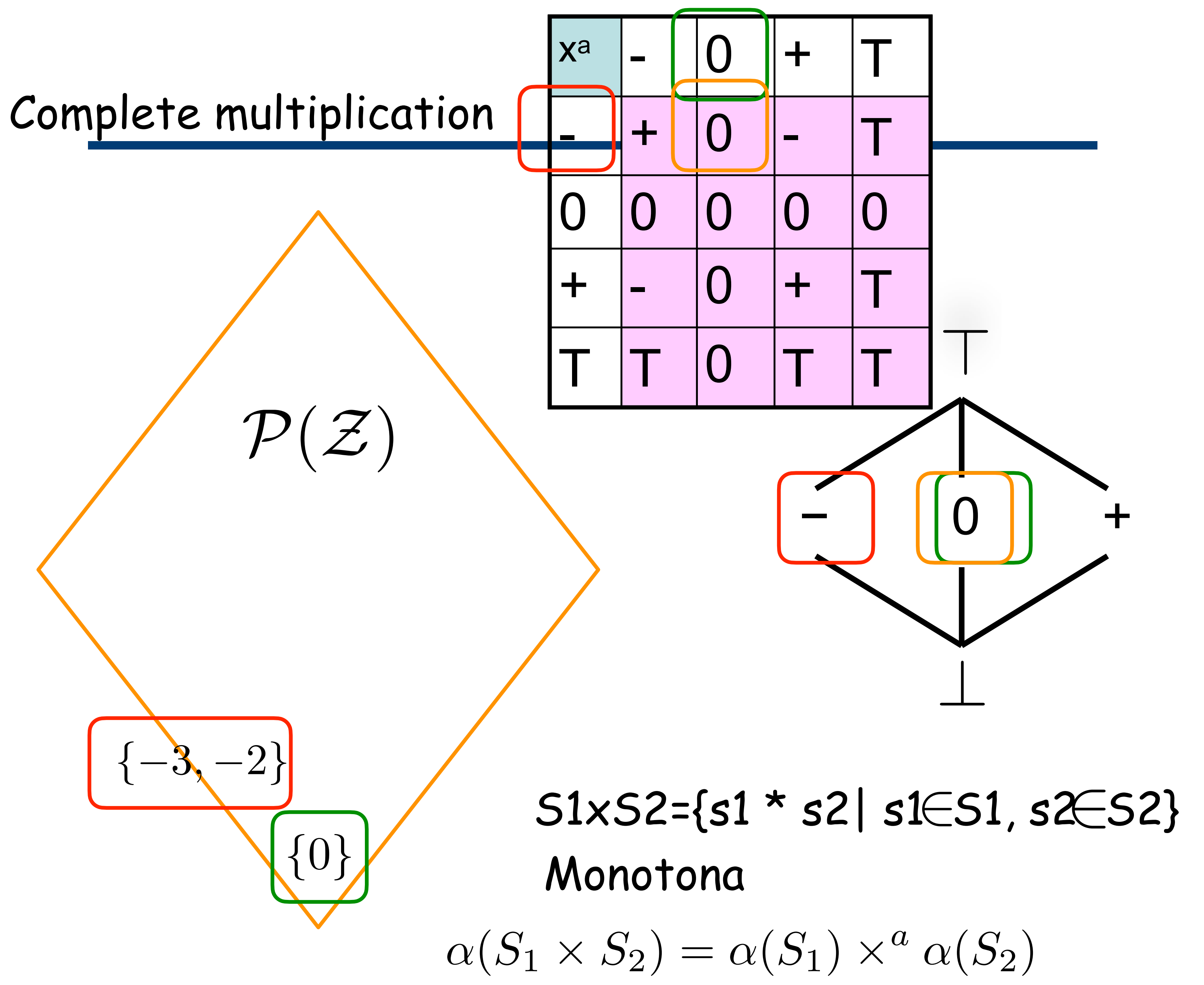
$$\alpha : \wp(\mathbb{Z}) \rightarrow A \qquad \alpha(S) = \begin{cases} \bot & \text{if } S = \emptyset \\ - & \text{if } S \neq \emptyset,\ S \subseteq \mathbb{Z}_{<0} \\ 0 & \text{if } S = \{0\} \\ + & \text{if } S \neq \emptyset,\ S \subseteq \mathbb{Z}_{>0} \\ \top & \text{otherwise} \end{cases}$$

# Let us picture it






| +a | - | 0 | + | ⊤ |
| --- | --- | --- | --- | --- |
| - | - | - | ⊤ | ⊤ |
| 0 | - | 0 | + | ⊤ |
| + | ⊤ | + | + | ⊤ |
| ⊤ | ⊤ | ⊤ | ⊤ | ⊤ |


Complete multiplication
xa
-
0
+
T
-
+
0
-
T
0
0
0
0
0
+
-
0
+
T
T
T
0
T
T
P(Z)
{−3, −2}
{0}
⊤
−
0
+
⊥
S1xS2={s1 * s2| s1∈S1, s2∈S2}
Monotona
α(S1 × S2) = α(S1) ×a α(S2)

# Trace-based operational semantics

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

The operational semantics updates a program-point, storage-cell pair, $pp, x$, using these four transition rules:

$$p_0, 2n \longrightarrow p_1, 2n \qquad p_1, n \longrightarrow p_0, n/2$$

$$p_0, 2n+1 \longrightarrow p_2, 2n+1 \qquad p_2, n \longrightarrow p_3, 4n$$

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 abstractly interpret, say, for parity

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

$p_0, even \longrightarrow p_1, even$

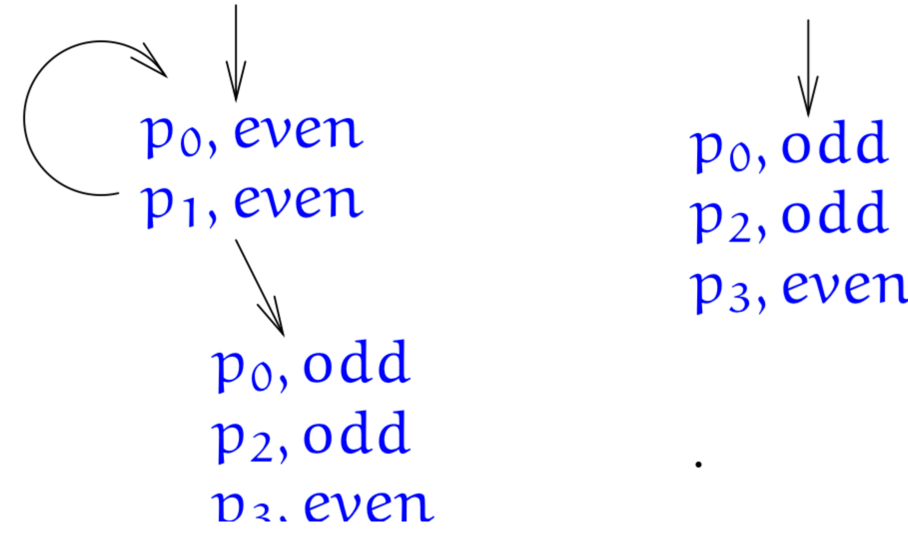
$p_0, odd \longrightarrow p_2, odd$

$p_1, even \longrightarrow p_0, even$

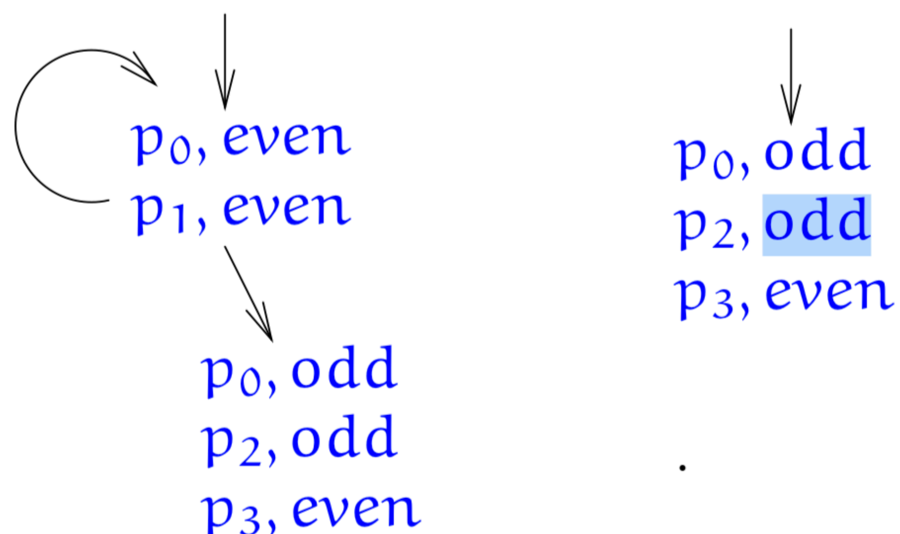
$p_1, even \longrightarrow p_0, odd$

$p_2, a \longrightarrow p_3, even$

Two trace trees cover the full range of inputs:



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



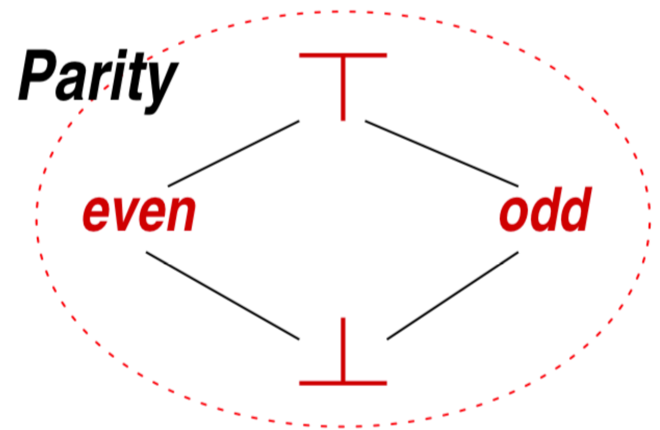

.

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.)

# The underlying abstract interpretation semantics




$\gamma : \mathrm{Parity} \to \mathcal{P}(\mathrm{Int})$

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

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

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

$\alpha : \mathcal{P}(\mathrm{Int}) \to \mathrm{Parity}$

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

The abstract transition rules are synthesized from the orginals:

$$p_i, a \longrightarrow p_j, \alpha(v'), \text{ if } v \in \gamma(a) \text{ and } p_i, v \longrightarrow p_j, v'$$

This recipe ensures that every transition in the original, “concrete” semantics is simulated by one in the abstract semantics.

# Another example: array bounds using intervals

Integer variables receive values from the *interval domain*,

$$I = \{[i, j] \mid i, j \in \text{Int} \cup \{-\infty, +\infty\}\}.$$

We define $[a, b] \sqcup [a', b'] = [\min(a, a'), \max(b, b')]$.

```
int a = new int[10];
i = 0;
while (i < 10)  {
      ... a[i] ...
      i = i + 1;
}
```



At convergence, i's ranges are

at $p_1$ : $[0..9]$

at $p_2$ : $[1..10]$

at loop exit : $[1..10] \sqcap [10, +\infty] = [10, 10]$

# Constant Propagation analysis

$p_0$ : `x = 1; y = 2;`
$p_1$ : `while (x < y + z)`
  $p_2$ : `x = x + 1;`
`}`
$p_3$ : *exit*

**Const**

$\top$ — var holds multiple values

$\cdots\ -1\ \ 0\ \ 1\ \ 2\ \cdots$ — var holds this value only

$\bot$ — var holds no value (dead code)

where $m + n$ is interpreted

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

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

$\top + k \longrightarrow \top$

$k + \top \longrightarrow \top$

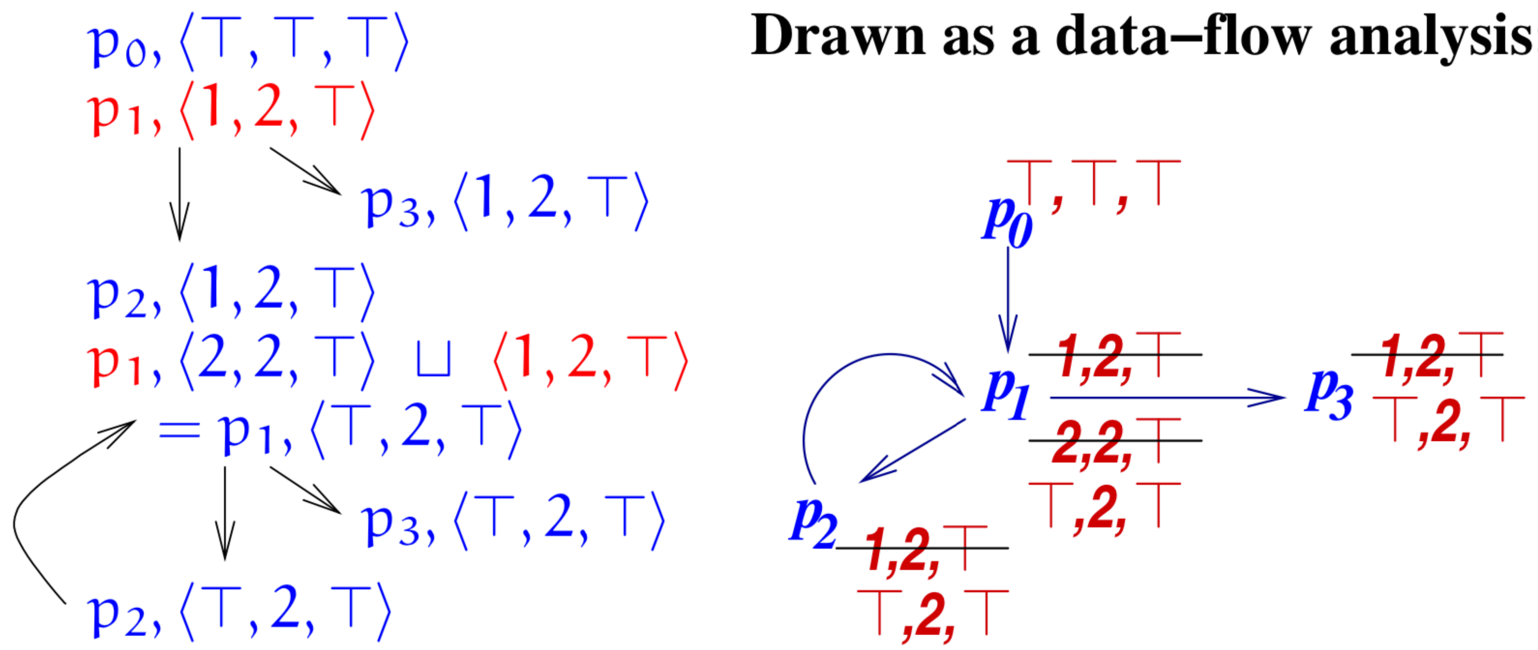
Let $\langle u, v, w \rangle$ abbreviate

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

**Abstract trace:**

$p_0, \langle \top, \top, \top \rangle$
$p_1, \langle 1, 2, \top \rangle$ → $p_3, \langle 1, 2, \top \rangle$
$p_2, \langle 1, 2, \top \rangle$
$p_1, \langle 2, 2, \top \rangle$ → $p_3, \langle 2, 2, \top \rangle$
$p_2, \langle 2, 2, \top \rangle$
$p_1, \langle 3, 2, \top \rangle$ → ...
...

# An acceleration is needed for finite convergence




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

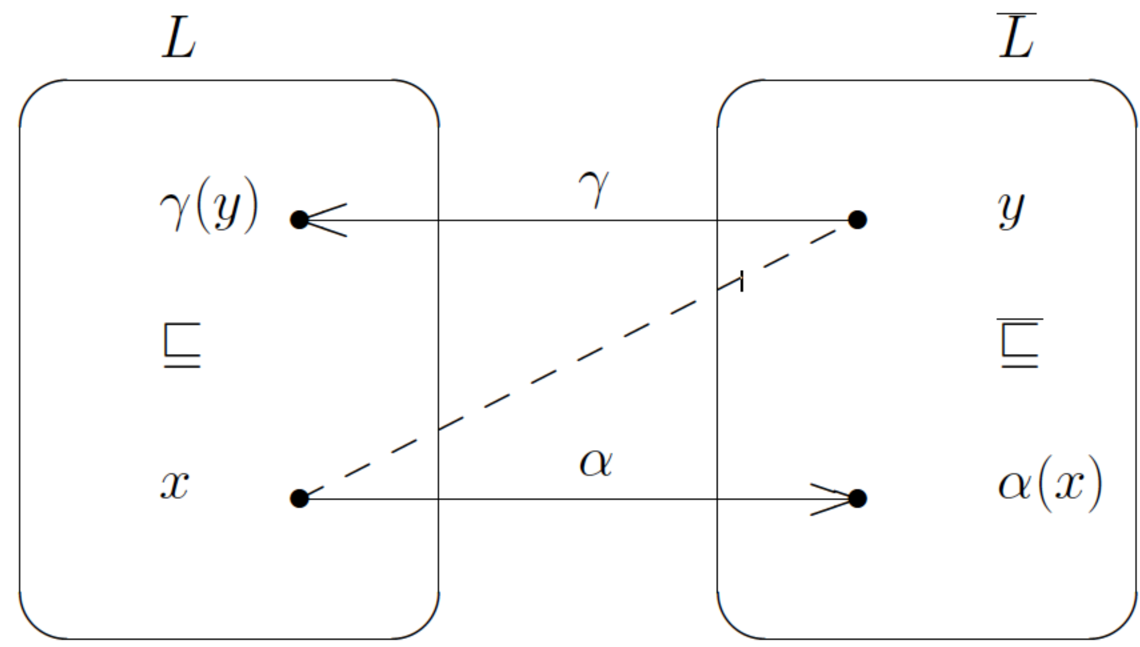
```
p0 :  x = 1; y = 2;
p1 :  while (x < y + z)  {
        p2 :  x = x + 1;
      }
p3 :  exit
```

2

# Summary what we saw so far:
# Property approximations using Galois insertions

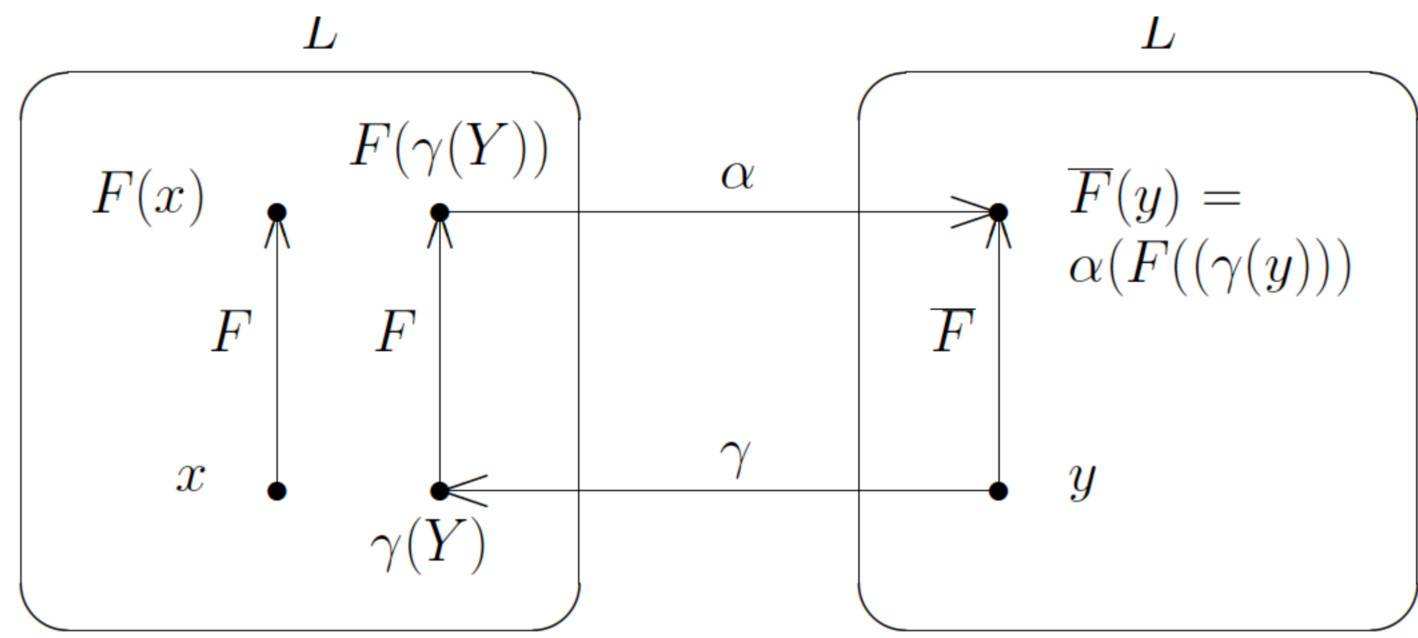
- Chose an abstract version $\overline{L}$ of the concrete properties $L$.
- Chose an abstract version $\overline{\sqsubseteq}$ of the concrete approximation relation $\sqsubseteq$.
- For each abstract property $y \in \overline{L}$ chose its concrete meaning $\gamma(y) \in L$.
- Decide once for all of the abstract approximation $\alpha(x) \in \overline{L}$ of any concrete property $x \in L$.

# Galois insertions




- $y$ is an approximation of $x$
- $\Leftrightarrow x \sqsubseteq \gamma(y)$
- $\Leftrightarrow \alpha(x) \overline{\sqsubseteq} y$

# Extending approximations to functions




- $\overline{F}$ is an approximation of $F$
- $\Leftrightarrow \alpha \circ F \circ \gamma \sqsubseteq \overline{F}$
- $\Leftrightarrow F \sqsubseteq \gamma \circ \overline{F} \circ \alpha$

# Fixpoint approximations using Galois insertions

- $L(\sqsubseteq, \bot, \sqcup)$ is a cpo of concrete properties,
  $F \in (L \stackrel{\text{con}}{\longmapsto} L)$ is continuous,
  $\text{lfp}_{\bot} F = \sqcup_{n \geq 0} F^n(\bot)$ is not computable.
- Chose a cpo $\overline{L}(\overline{\sqsubseteq}, \overline{\bot}, \overline{\sqcup})$ of abstract properties such that $L \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \overline{L}$.
- Define $\overline{F} = \alpha \circ F \circ \gamma$.
  and $\quad \overline{\bot} = \alpha(\bot)$.
- then $\text{lfp}_{\bot} F \sqsubseteq \gamma(\text{lfp}_{\overline{\bot}} \overline{F})$.

# Fixpoint approximations using Galois insertions

- If $\overline{L}$ is finite (or satisfies the ascending chain condition), you have got an effective program analysis algorithm:

$$\langle \overline{\bot}, \overline{F} \rangle := \text{analysis}(\text{Program});$$
$$\%\%\ \alpha(\bot) \sqsubseteq \overline{\bot} \wedge \alpha \circ F \circ \gamma \sqsubseteq \overline{F}$$
$$X := \overline{\bot};$$

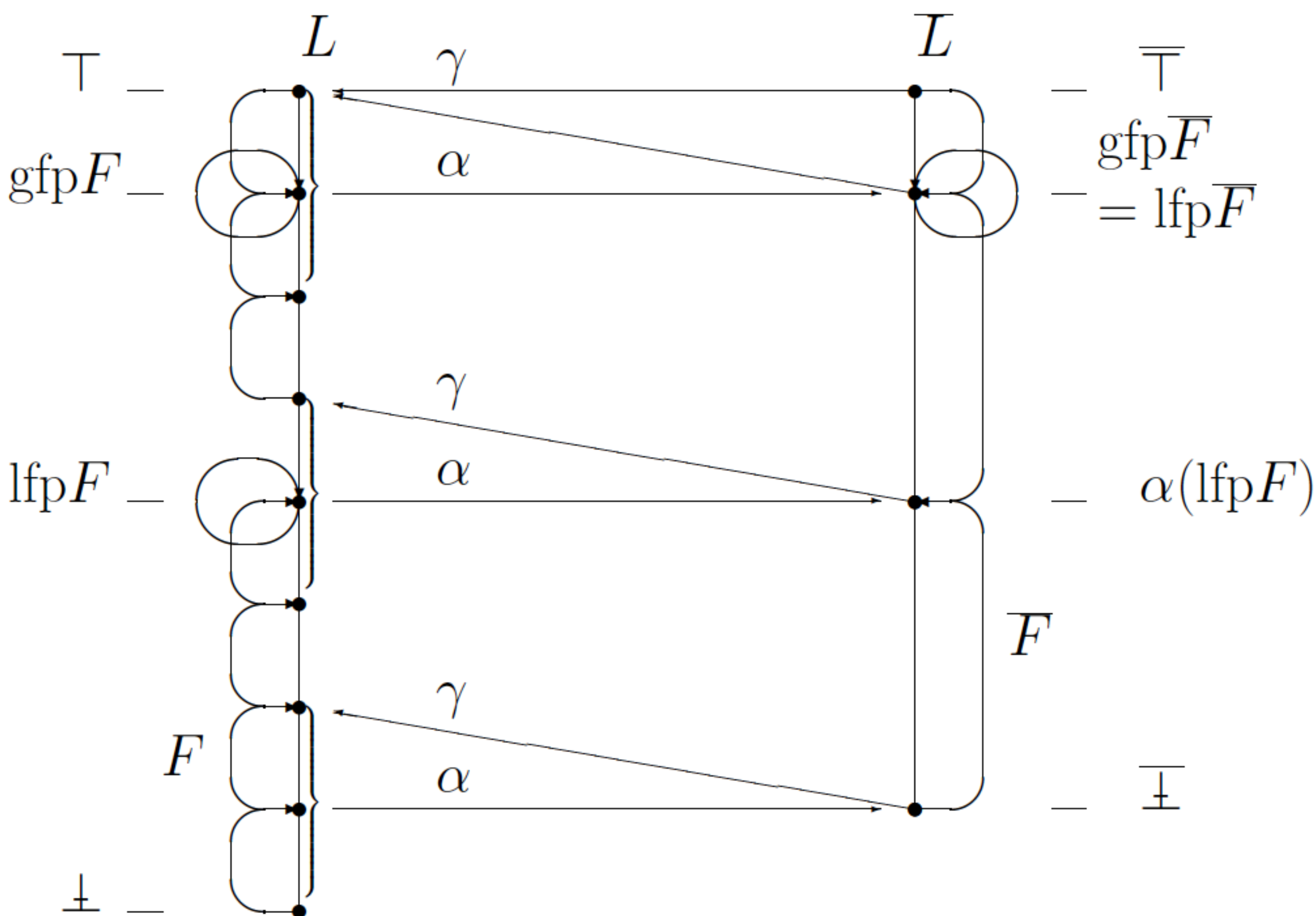
**repeat**

$$Y := X;$$
$$X := \overline{F}(X)$$

**until** $Y = X$;

$$\%\%\ \text{lfp}_{\bot} F \sqsubseteq \gamma(X) \wedge \text{lfp}_{\overline{\bot}} \overline{F} \sqsubseteq X$$

# Fixpoint approximations using Galois insertions



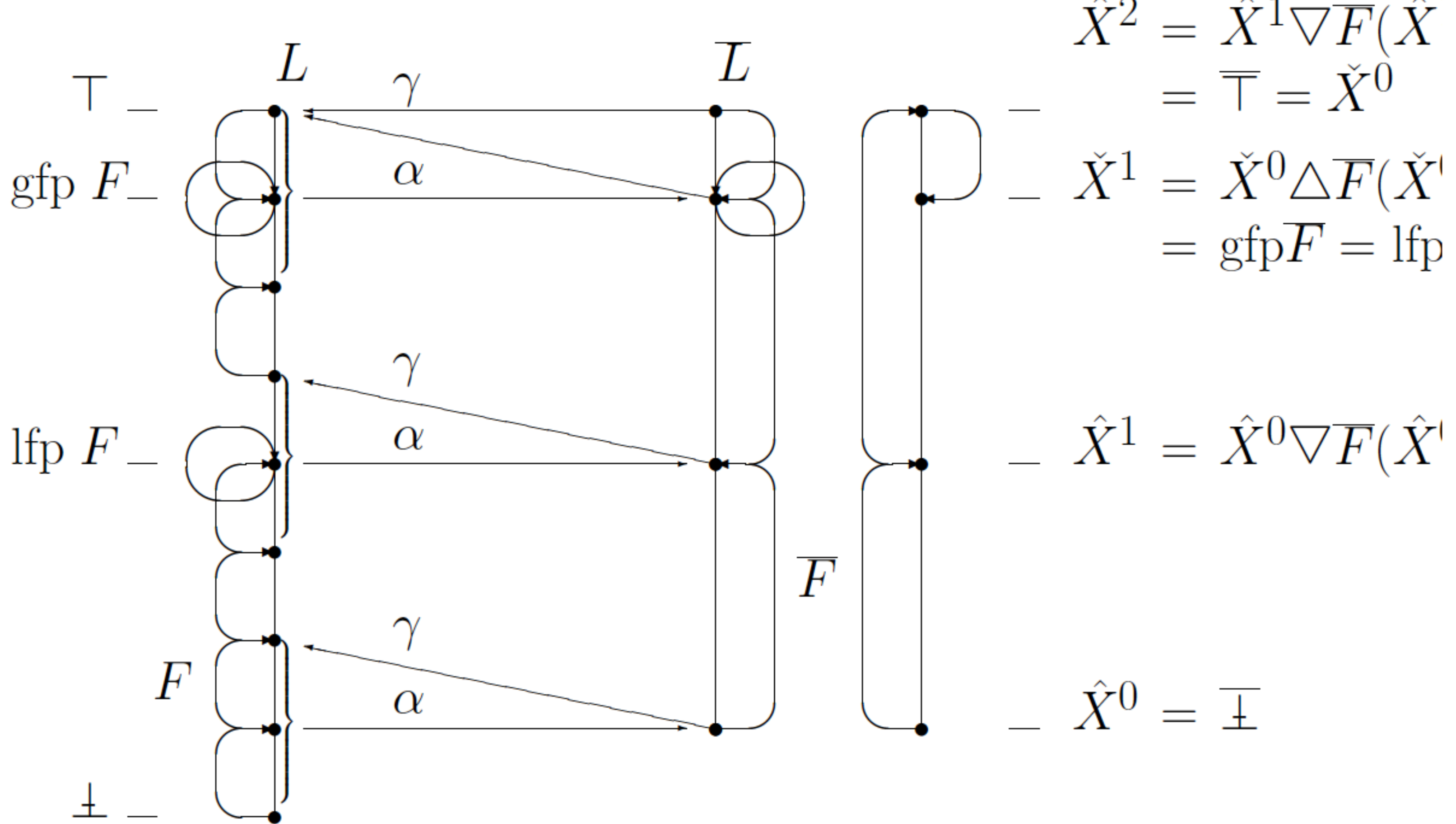
# and if the abstract domain does not satisfy the ACC?

## Widening Operator

A widening operator $\nabla \in \overline{L} \times \overline{L} \mapsto \overline{L}$ is such that:

- $\forall x, y \in L : x \sqsubseteq x \nabla y$
- $\forall x, y \in L : y \sqsubseteq x \nabla y$
- for all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not strictly increasing

# Fixpoint Approximation with Widening/Narrowing



# Fixpoint approximation with widening

The upward iteration sequence with widening:

- $\hat{X}^0 = \overline{\bot}$
- $\hat{X}^{i+1} = \hat{X}^i$ if $\overline{F}(\hat{X}^i) \sqsubseteq \hat{X}^i$
  $= \hat{X}^i \triangledown \overline{F}(\hat{X}^i)$ otherwise

This is a an ascending chain that eventually stabilises when

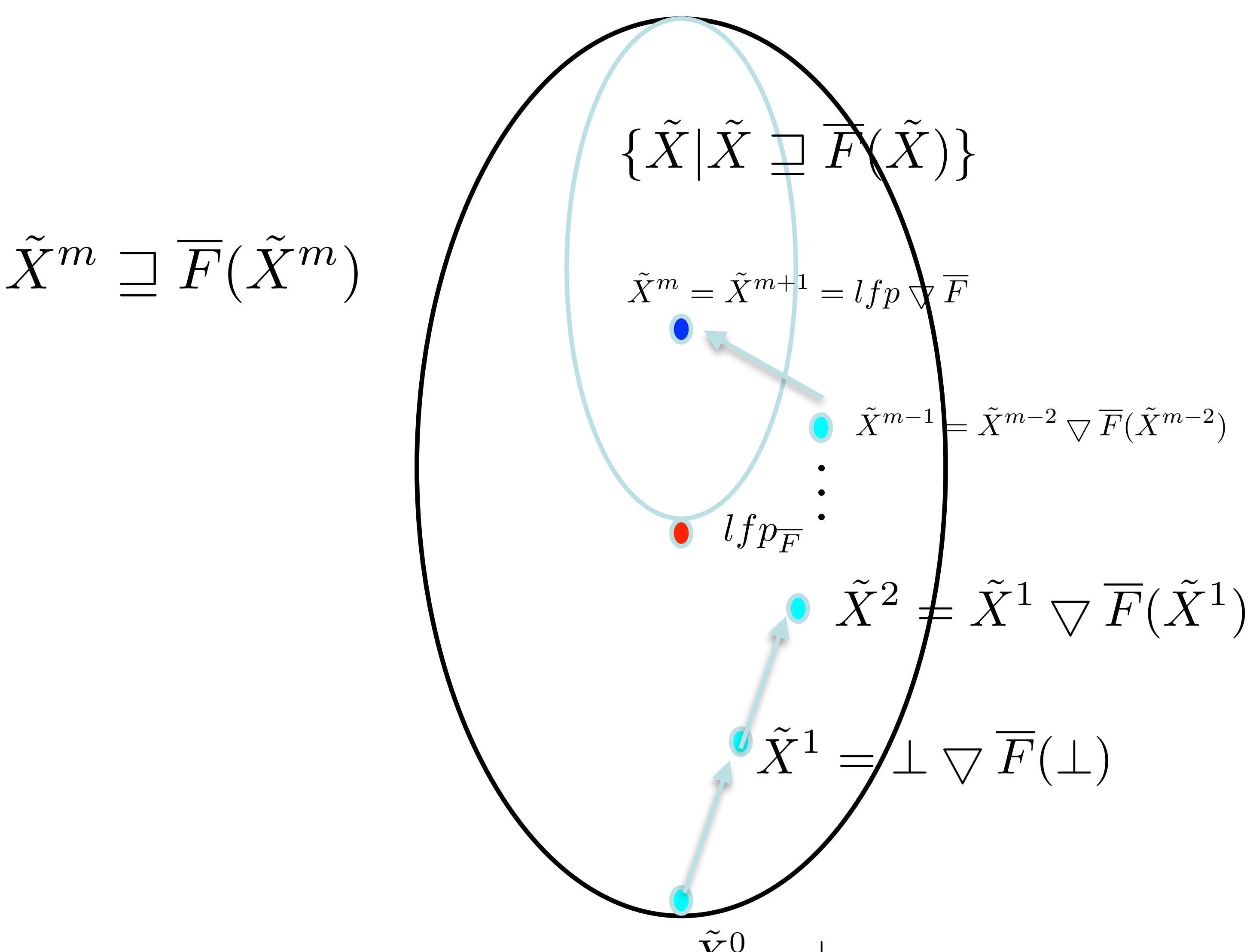
$$\overline{F}(X^m) \sqsubseteq X^m \quad \text{for some m}$$

Tarsky thm then gives

$$X^m \sqsupseteq lfp_{\overline{F}}$$

# Fixpoint approximation with widening

$$\tilde{X}^m \sqsupseteq \overline{F}(\tilde{X}^m)$$



# Example: widening on intervals

- $\overline{L} = \{\bot\} \cup \{[\ell,\, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{+\infty\} \wedge \ell \leq u\}$
- The widening extrapolates unstable bounds to infinity:

$$
\begin{aligned}
\bot \mathbin{\nabla} X &= X \\
X \mathbin{\nabla} \bot &= X \\
[\ell_0,\ u_0] \mathbin{\nabla} [\ell_1,\ u_1] &= [\text{if } \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0, \\
&\qquad \text{if } \mathrm{u}_1 > \mathrm{u}_0 \text{ then } +\infty \text{ else } \mathrm{u}_0]
\end{aligned}
$$

# Improve widening on intervals

- Extrapolate to zero, one or infinity:

$$
\begin{aligned}
\bot \bigtriangledown X &= X \\
X \bigtriangledown \bot &= X \\
[\ell_0,\ u_0] \bigtriangledown [\ell_1,\ u_1] &= [\text{if } \ell \leq \ell_1 < \ell_0 \ \wedge\ \ell \in \{1, 0, -1\} \text{ then l} \\
&\quad \text{elsif } \ell_1 < \ell_0 \text{ then } -\infty \\
&\quad \text{else } \ell_0, \\
&\quad \text{if } \mathrm{u}_0 < \mathrm{u}_1 \leq \mathrm{u} \ \wedge\ \mathrm{u} \in \{-1, 0, 1\} \text{ then u} \\
&\quad \text{elsif } \mathrm{u}_0 < \mathrm{u}_1 \text{ then } +\infty \\
&\quad \text{else } \mathrm{u}_0]
\end{aligned}
$$

# Improving on $lfp_\nabla \overline{F}$

- Widening gives us an upper approximation $lfp_\nabla \overline{F}$ of $lfp\overline{F}$
- But $\overline{F}(lfp_\nabla \overline{F}) \sqsubseteq lfp_\nabla \overline{F}$ so we can improve approximations by considering the sequence $\overline{F}^n(lfp_\nabla \overline{F})$
- for all i we have

$$lfp\overline{F} \sqsubseteq \overline{F}^n\ (lfp_\nabla \overline{F}) \sqsubseteq \overline{F}(lfp_\nabla \overline{F})$$

- so we can stop anytime with an upper approximation
- Defining a **Narrowing operator** gives us a way to describe

when to stop

# Narrowing operator

An operator $\triangle : L \times L \rightarrow L$ is a *narrowing operator* iff

- $l_2 \sqsubseteq l_1 \;\Rightarrow\; l_2 \sqsubseteq (l_1 \triangle l_2) \sqsubseteq l_1$ for all $l_1, l_2 \in L$, and
- for all descending chains $(l_n)_n$ the sequence $(l_n^{\triangle})_n$ eventually stabilises.

We construct the sequence $Y_n$

$$Y_n = \begin{cases} \mathit{lfp}_{\nabla}(f) & \text{if } n = 0 \\ Y_{n-1} \triangle \overline{F}(Y_{n-1}) & \text{if } n > 0 \end{cases}$$

One can show that:

- $Y_n$ s a descending chain where all elements satisfy $\mathit{lfp}(f) \sqsubseteq Y_n$
- the chain eventually stabilises so $F(Y^{m'}) = Y^{m'}$ for some value $m'$

# Fixpoint approximation with widening and narrowing