Reliable and Interpretable Artificial Intelligence

Martin Vechev
Department of Computer Science, ETH Zurich

ETH Zürich

How good (robust) is your neural net?

Neural networks are ***not*** robust to input perturbations
(e.g., image rotation / change of lighting)

Misclassifications in neural networks deployed in self-driving cars [1]
In each picture one of the 3 networks makes a mistake…

[1] Pei et. al., DeepXplore: Automated Whitebox Testing of Deep Learning Systems, SOSP 2017

Wanted: **Automated** and **scalable** analysis to verify realistic NNs

Useful in:

- Ensuring correctness of a larger cyber-physical system that uses the NN
- Proving robustness of the NN (beyond just finding adversarial examples)
- Learning interpretable specs of the NN
- Comparing NNs

Problem Statement and Challenges

Neural Network Analysis Problem

Given

- a **neural network N**
- a **property over inputs φ**
- a **property over outputs ψ**

check whether $\forall i \in I. i \vDash \varphi \Longrightarrow N(i) \vDash \psi$ holds

Challenges:

- The property φ over inputs usually captures an **unbounded set of inputs**
- Existing symbolic solutions **do not scale** to large networks (e.g. conv nets)

Key Technical Insight: AI for AI

Deep Neural Nets:

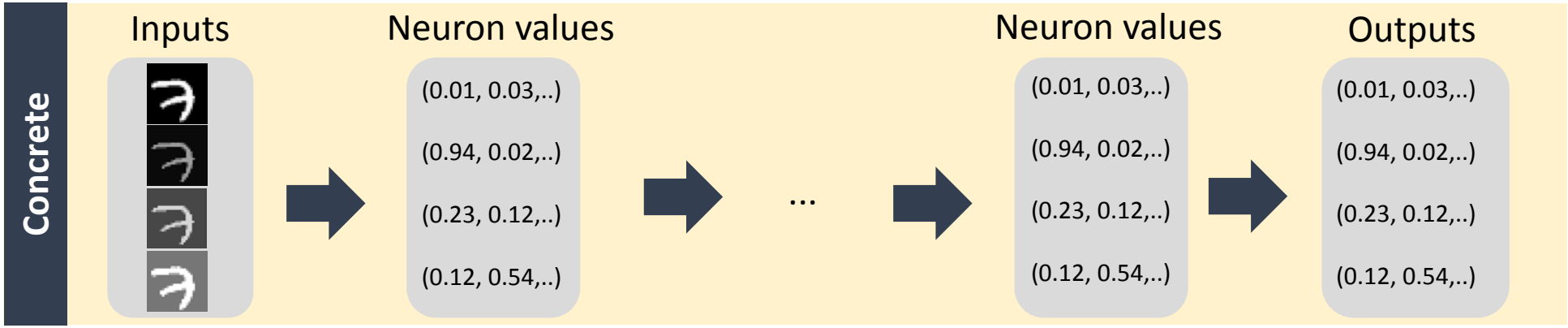
Affine transforms + Restricted non-linearity

+

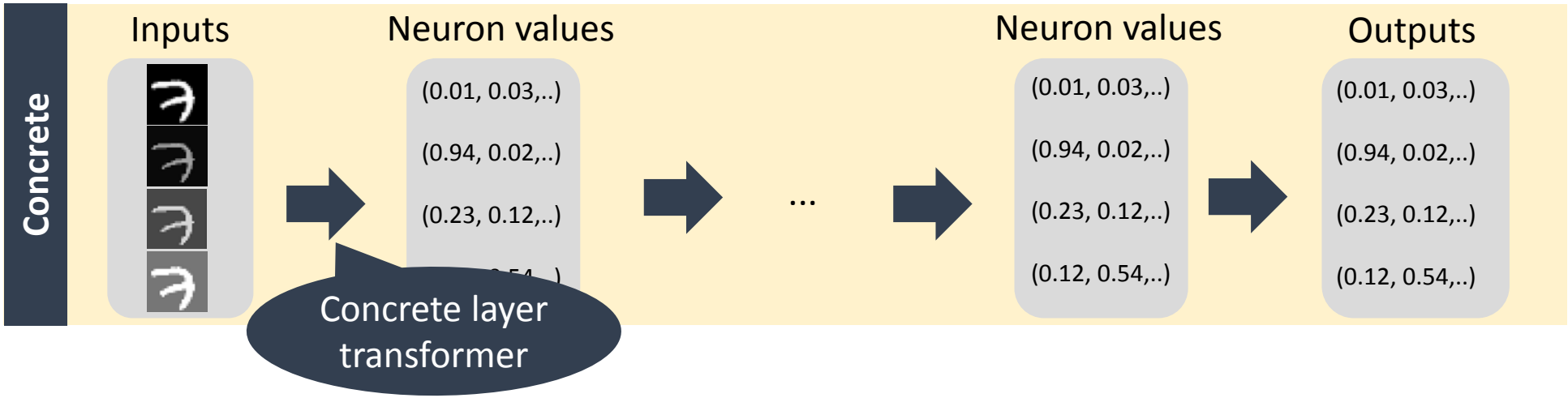
Abstract Interpretation:

Scalable and Precise Numerical Domains

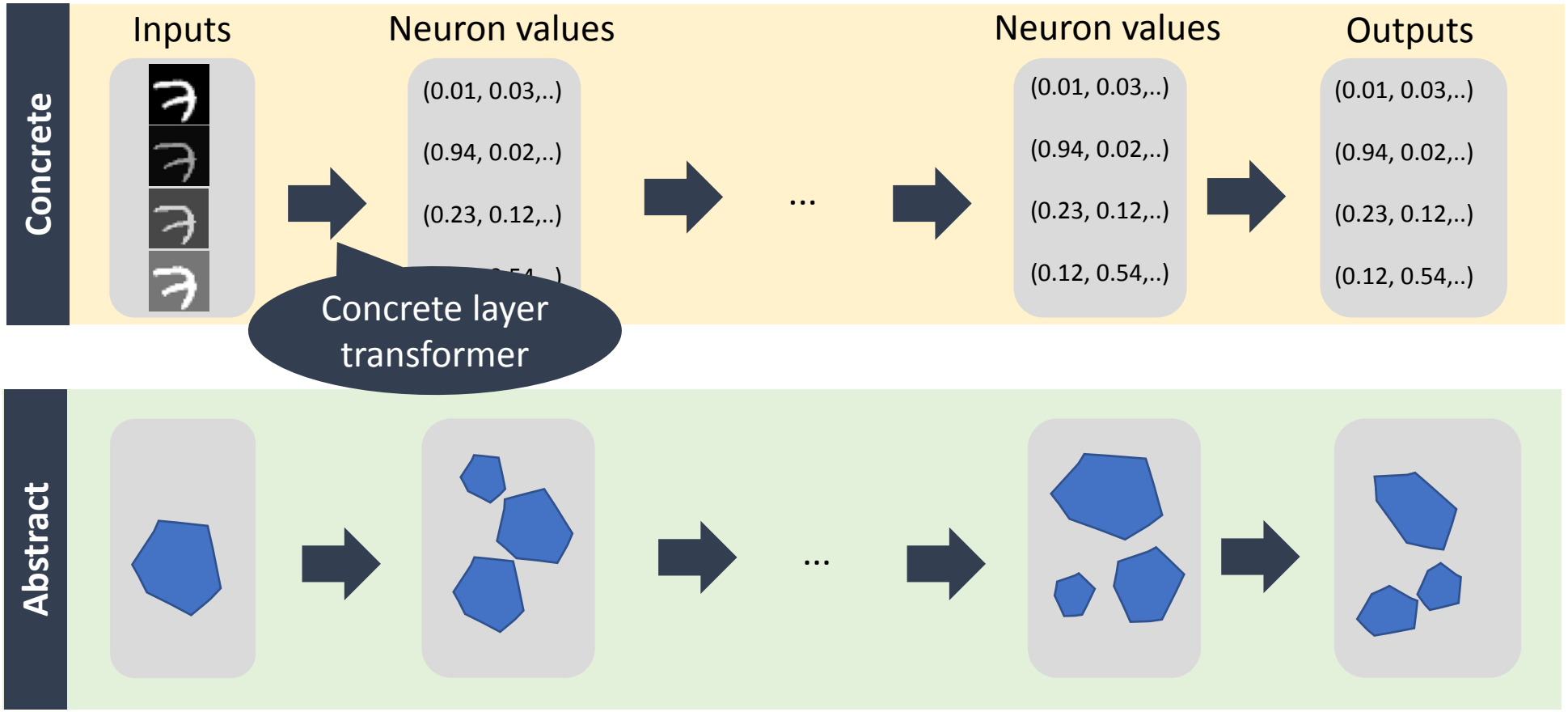
AI²: Abstract Interpretation for NNs



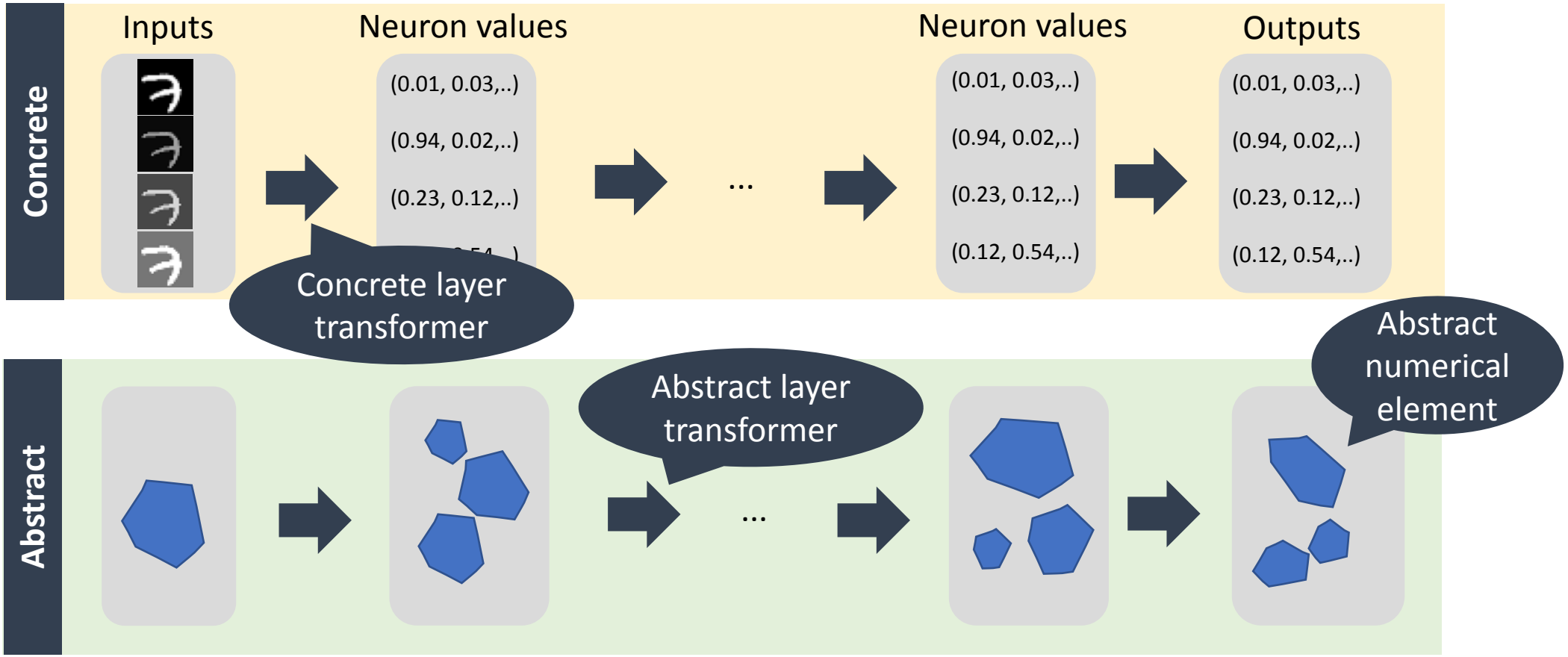
AI²: Abstract Interpretation for NNs



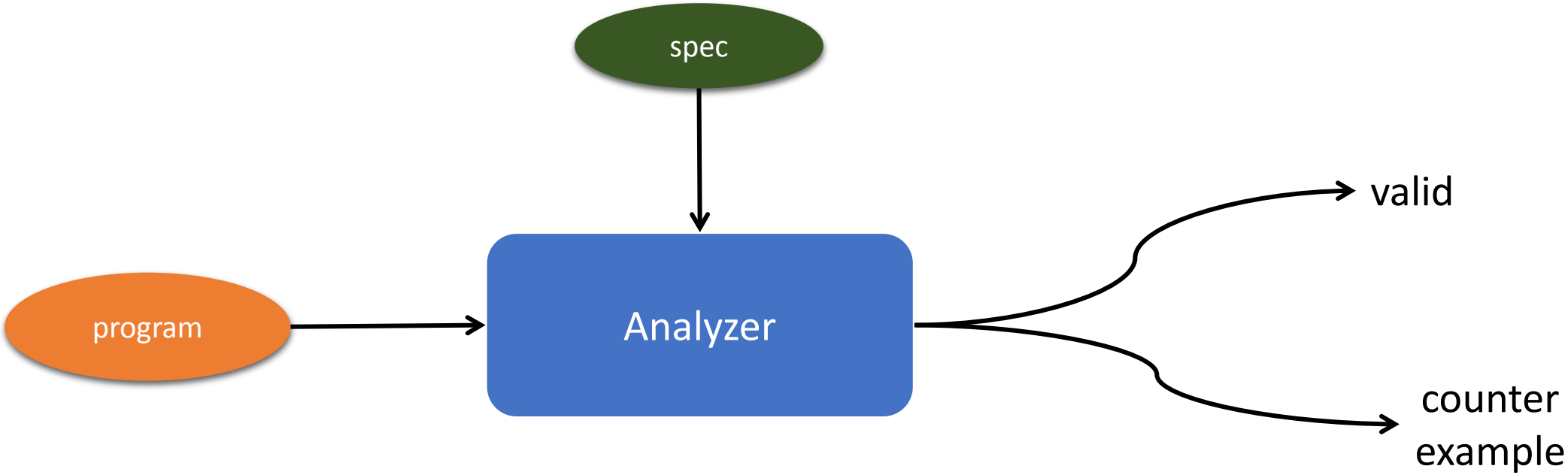
AI²: Abstract Interpretation for NNs

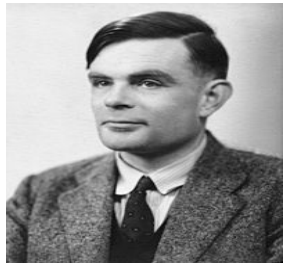


AI²: Abstract Interpretation for NNs



Why Abstractions? (high-level view)





Alan Turing

Minor issue ☺ : general problem is undecidable

Hence: approximation

What is Abstract Interpretation?

We will see a glimpse of A.I., and only the parts we need for our problem.

However, this should be enough to get a working intuition of it, to know the libraries and to know how to apply it.

Abstract Interpretation: a primer

The theory of abstract interpretation is a theory of approximation

Probably one of the most elegant theories in computer science

Patrick Cousot
Inventor of
Abstract Interpretation

- an elegant theoretical framework
- systematic way to build automated analyzers
- a way to think about approximation
- theory invented in late 70s
- started gaining popularity in the 90s
- all commercial tools use some form of A.I.

The principles of approximation are fundamental to reasoning about computation with infinite state spaces.

Abstract Interpretation: a primer

A.I. concerns itself with questions such as:

- What is it that we are approximating?
- What does it mean for the approximation to be optimal (or to approximate)?
- What does it mean for an approximation to be correct?
- How do we actually build a correct and optimal system?
- Can the process of building an analyzer be automated?

Abstract Interpretation Recipe

for building an A.I. engine

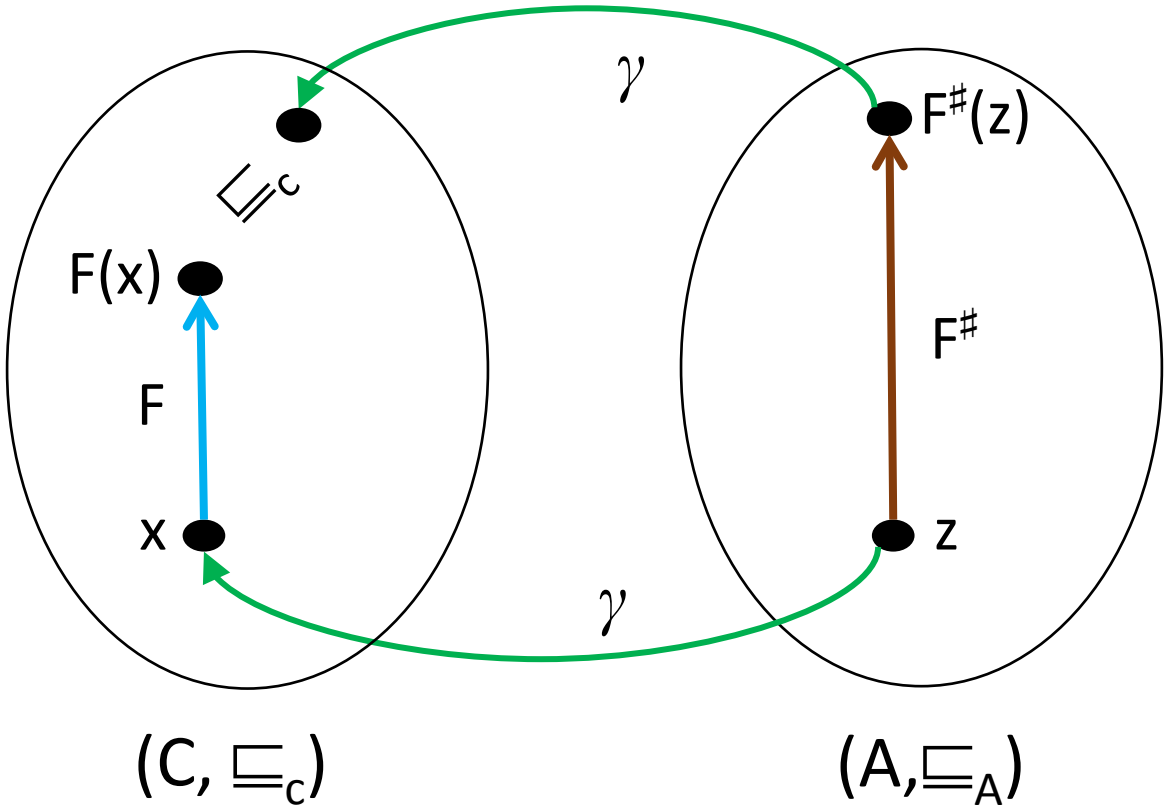
1. come up with an abstract domain
   - select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   - we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   - we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

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

The fixed point is the over-approximation of the program

A.I. cheat sheet



- $(C, \sqsubseteq_c)$ is the concrete lattice. An element x in C is a set of concrete program states.
- $(A,\sqsubseteq_A)$ is the abstract lattice. An element z in A is an abstract element that represents a set of concrete states.
- F is your program. F(x) applies it on set of states x. F is monotone. Least fixed point of F (LFP F) is an element in C that captures all reachable states of F – the set may be infinite or unbounded so we typically cannot compute it.
- $F^\sharp$ is the abstract transformer. $F^\sharp(z)$ applies $F^\sharp$ to abstract element z. $F^\sharp$ should be monotone (see Tarski's theorem)
- γ is the concretization: it defines to which concrete states an abstract element maps to. γ is monotone. It is key to defining what it means for $F^\sharp$ to approximate F.
- We iterate $F^\sharp$ to a fixed point. If $F^\sharp$ approximates F, then its least fixed point (LFP $F^\sharp$) approximates LFP F! We can compute LFP $F^\sharp$!

We just need a little bit of theory to know when we reach a fixed point and what approximation means. Simply a fixed point is not enough! We need to know $F^\sharp$ actually approximates F.

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: $\text{lfp}^{\sqsubseteq} f$ exists

Alfred Tarski

Note: the complete lattice can be of infinite height

Practically: 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 iterative algorithm for computing $\text{lfp}^{\sqsubseteq} f$

Approximating a Function

So we have 2 functions:

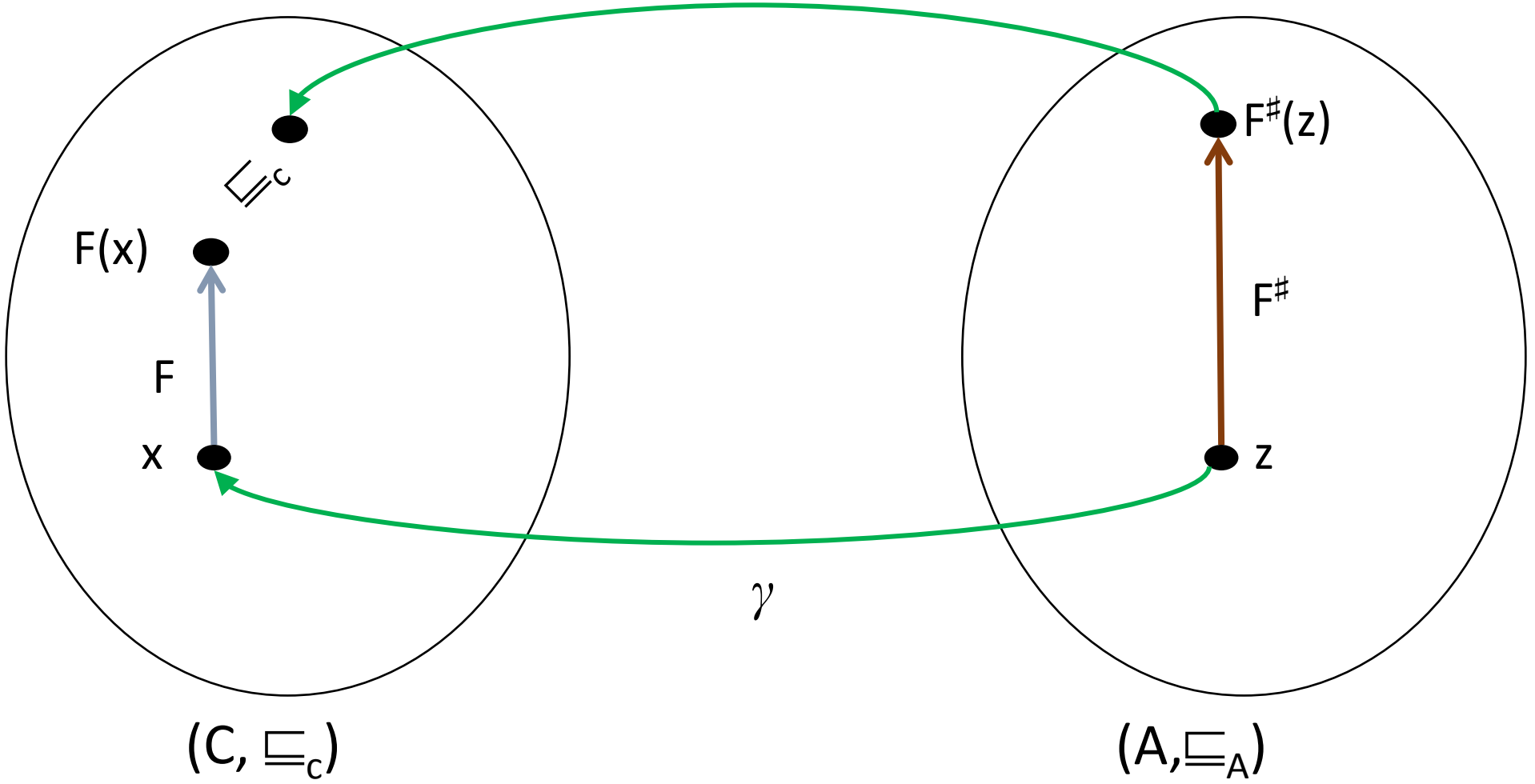
$$F: C \longrightarrow C$$

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

Here is a definition of function approximation:

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

Visualizing the Definition



Key Theorem of A.I.: 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.

Now, let us see how to instantiate the theory.

Abstract Interpretation Recipe

for building an A.I. engine

1. come up with an abstract domain
   - select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   - we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   - we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

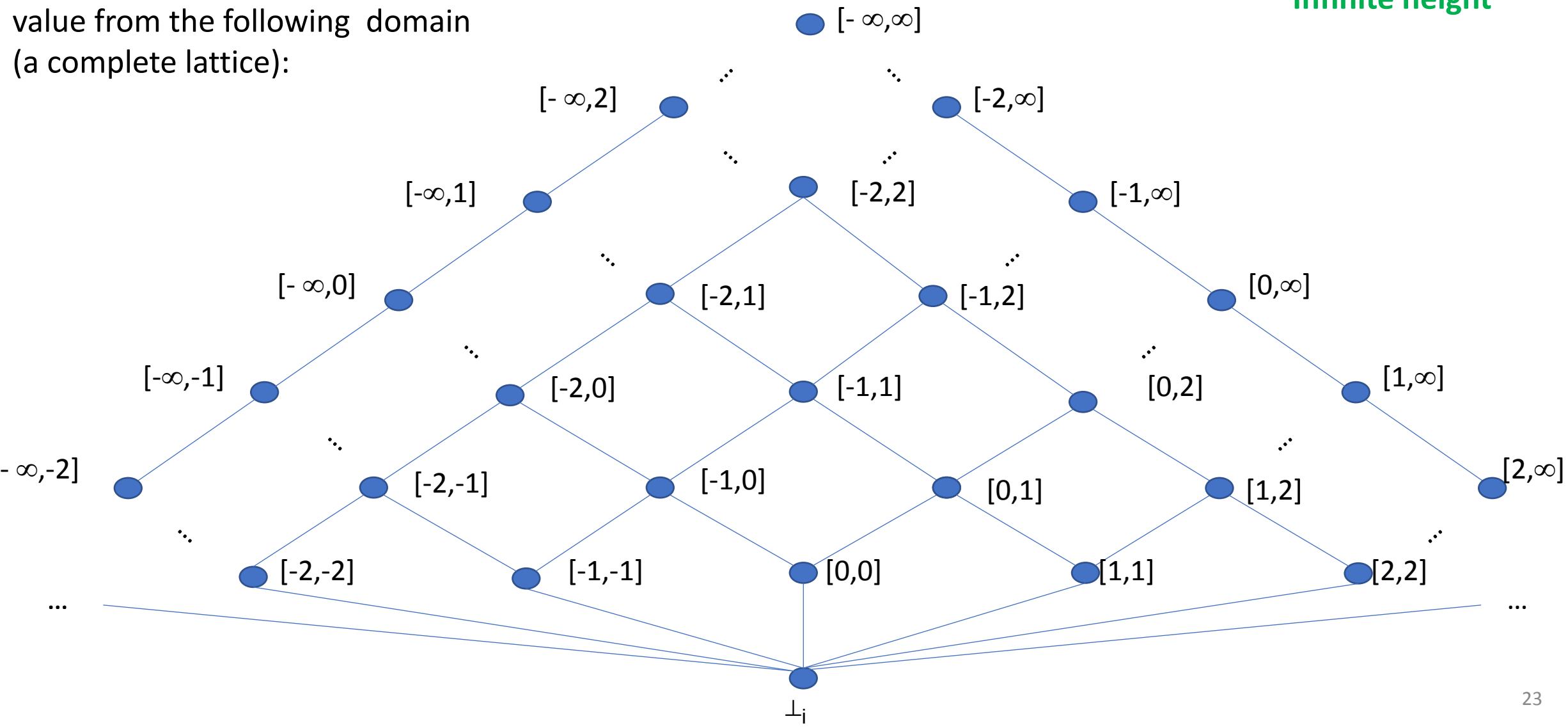
3. iterate abstract transformers over the abstract domain
   - until we reach a fixed point

The fixed point is the over-approximation of the program

Interval Domain

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

Infinite height



Abstract Interpretation Recipe

for building an A.I. engine

1. come up with an abstract domain
   - select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   - we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   - we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

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

The fixed point is the over-approximation of the program

Arithmetic Expressions

If we add $\perp_i$ to any other element, we get $\perp_i$.

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

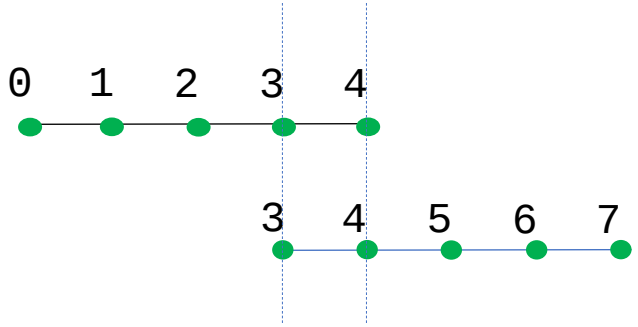
```
[x,y] + [z,q] = [x + z, y + q]
```

what about $*$?

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

$\leq (av_1 , av_2)$

what should $\leq$([0,4], [3,7]) produce ?



one answer is: ([0,3], [3,7]). Is it sound ?

another non-comparable answer is: ([0,4], [4,7]). Is it sound ?

Can you find a more precise answer ?

$$\leq (av_1 , av_2)$$

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

$$\leq ([0,4], [3,7]) = ([0,4] \sqcap_i [-\infty,7], \ [0, \infty] \sqcap_i [3, 7]) = ([0,4], [3,7])$$

Abstract Interpretation Recipe

for building a static analyzer

1. come up with an abstract domain
   - select based on the type of properties you want to prove

2. define abstract semantics for the programming language w.r.t. to the abstract domain from step 1.
   - we need to define the abstract transformers, that is, the effect of statement / expression on the abstract domain
   - we need to prove that the abstract semantics are sound w.r.t concrete semantics of the programming language

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

The fixed point is the over-approximation of the program

We typically start the iteration from $\perp$ meaning nothing is reachable.

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:
}
```

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

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

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

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

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

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

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

Iterate 1

Initially, the values are T as the variables can be anything

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

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

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

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

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

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

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

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

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

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

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

Iterate 3

```
foo (int i) {

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

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

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

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

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

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

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

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

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

Iterate 4

```
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

```
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 → [-∞,∞], y → [-∞,∞], i → [-∞,∞]

2: x → [5,5], y → [-∞,∞], i → [-∞,∞]

3: x → [5,5], y → [7,7], i → [-∞,∞]

4: x → [5,5], y → [7,7], 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 → [7,7], i → [-∞, -1]

Iterate 7: we jumped to ∞

```
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:
}
```

Step 1: we did a join: [7,7] ⊔ [8,8] = [7,8]

Step 2: we saw that end point of interval increased, so we bumped it to ∞

1: x → [-∞,∞], y → [-∞,∞], i → [-∞,∞]

2: x → [5,5], y → [-∞,∞], i → [-∞,∞]

3: x → [5,5], y → [7, ∞], i → [-∞,∞]

4: x → [5,5], y → [7,7], 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 → [7,7], i → [-∞, -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, ∞], i → [-∞,∞]

4: x → [5,5], y → [7, ∞], 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, ∞], 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 → [-∞,∞], y → [-∞,∞], i → [-∞,∞]

2: x → [5,5], y → [-∞,∞], i → [-∞,∞]

3: x → [5,5], y → [7, ∞], i → [-∞,∞]

4: x → [5,5], y → [7, ∞], i → [0,∞]

5: x → [5,5], y → [8, ∞], i → [0,∞]

6: x → [5,5], y → [8,8], i → [-1,∞]

7: x → [5,5], y → [7, ∞], i → [-∞, -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, ∞], i → [-∞,∞]

4: x → [5,5], y → [7, ∞], i → [0,∞]

5: x → [5,5], y → [8, ∞], i → [0,∞]

6: x → [5,5], y → [8, ∞], i → [-1,∞]

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

Iterate 11: a fixed point is reached

```
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, ∞], i → [-∞,∞]

4: x → [5,5], y → [7, ∞], i → [0,∞]

5: x → [5,5], y → [8, ∞], i → [0,∞]

6: x → [5,5], y → [8, ∞], i → [-1,∞]

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

We saw a glimpse of A.I., but enough to get a working intuition with it.

Abstract Interpretation is a rich area with many branches and applications.

A particular branch we use when analyzing neural networks is numerical domains (relating variables).

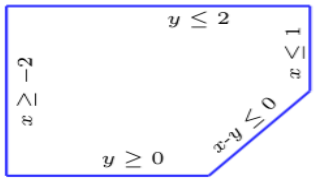
Transformers of these domains can be very tricky to implement efficiently and correctly!

Good analyzer is a combination of careful Math (e.g., polyhedral computation) + Efficient algorithms and coding.

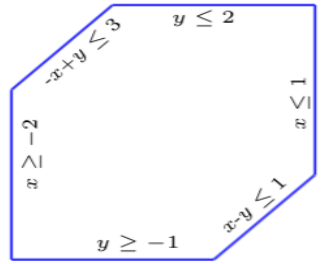
To use A.I., we typically use existing libraries which implement all transformers and operators we need, e.g.,:

→ ELINA (ETH): http://elina.ethz.ch/
→ Apron (ENS): http://apron.cri.ensmp.fr/library/

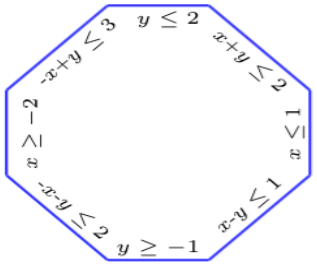
Numerical Abstract Domains



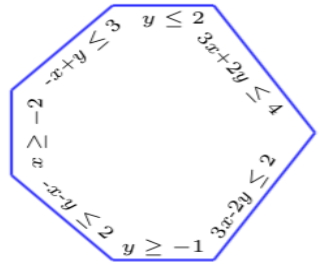
Pentagons $O(n^2)$



Zones $O(n^3)$



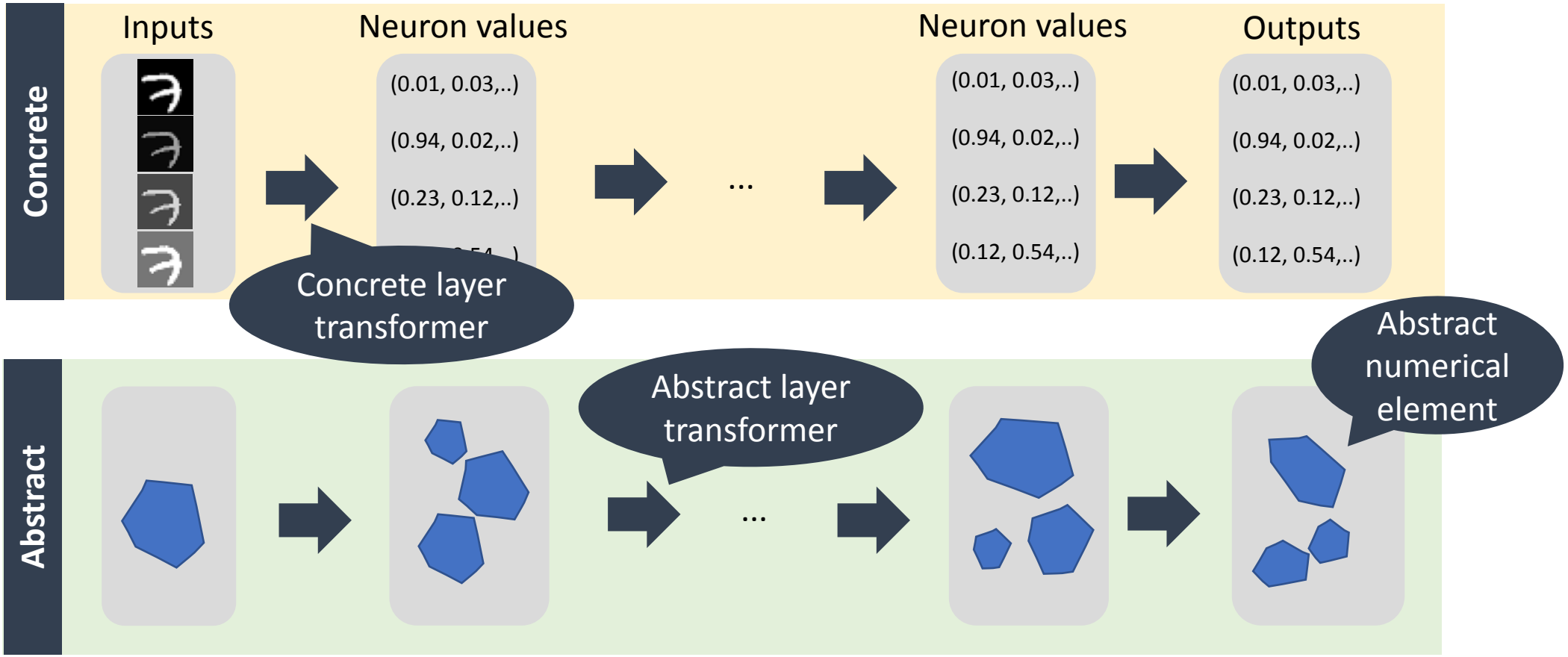
Octagons $O(n^3)$



Polyhedra $O(2^n)$

We will use these libraries when analyzing robustness of neural networks…

AI²: Abstract Interpretation for NNs



Zonotope Abstract Domain

A numerical domain that is **exact** for linear opeartions. Used to analyze numerical programs.

Here, each variable (in our case, **abstract neuron**) is captured in an **affine form**.

We can think of this domain as a more extensive version of the Interval domain, we still talk about a single variable, but the variables can also be related a little bit through parameters.

Zonotope Abstract Domain

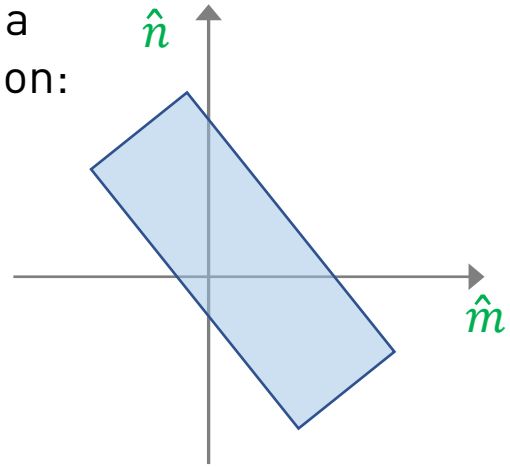
If we have two (concrete) neurons n and m,
then the abstract neurons will look like:

$$\hat{n} = {a_0}^n + \sum_{i=1}^{k} {a_i}^n \epsilon_i$$

$$\hat{m} = {a_0}^m + \sum_{i=1}^{k} {a_i}^m \epsilon_i$$

The **meaning** (γ) is a polytope centered around ${a_0}^n$ and ${a_0}^m$

Example of a concretization:



Zonotope Abstract Domain

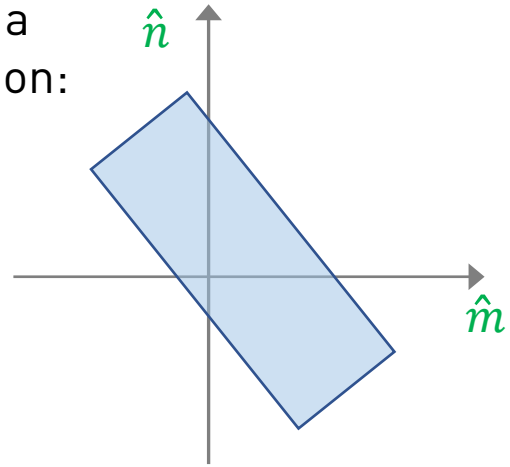
If we have two (concrete) neurons n and m,
then the abstract neurons will look like:

$$\hat{n} = {a_0}^n + \sum_{i=1}^{k} {a_i}^n \epsilon_i$$

$$\hat{m} = {a_0}^m + \sum_{i=1}^{k} {a_i}^m \epsilon_i$$

The **meaning** (γ) is a polytope centered around ${a_0}^n$ and ${a_0}^m$

Example of a concretization:



ϵ_i : noise terms ranging [-1,1] shared between abstract neurons

${a_i}^n$: real number that controls magnitude of noise

Closed under affine transforms, e.g., $\hat{n} + \hat{m}$

Not closed under joins and meets, e.g.,: $\hat{n} \sqcup \hat{m}$, $\hat{n} \sqcap \hat{m}$

Concretization + centering

Centered means there is a center point called C, where from any point X in the polytope, we can obtain a flipped point Y of X, where Y = 2C-X, and Y is in the polytope and X and Y are equal distance from C.

For instance, ψ below is centered around C = (1,0).

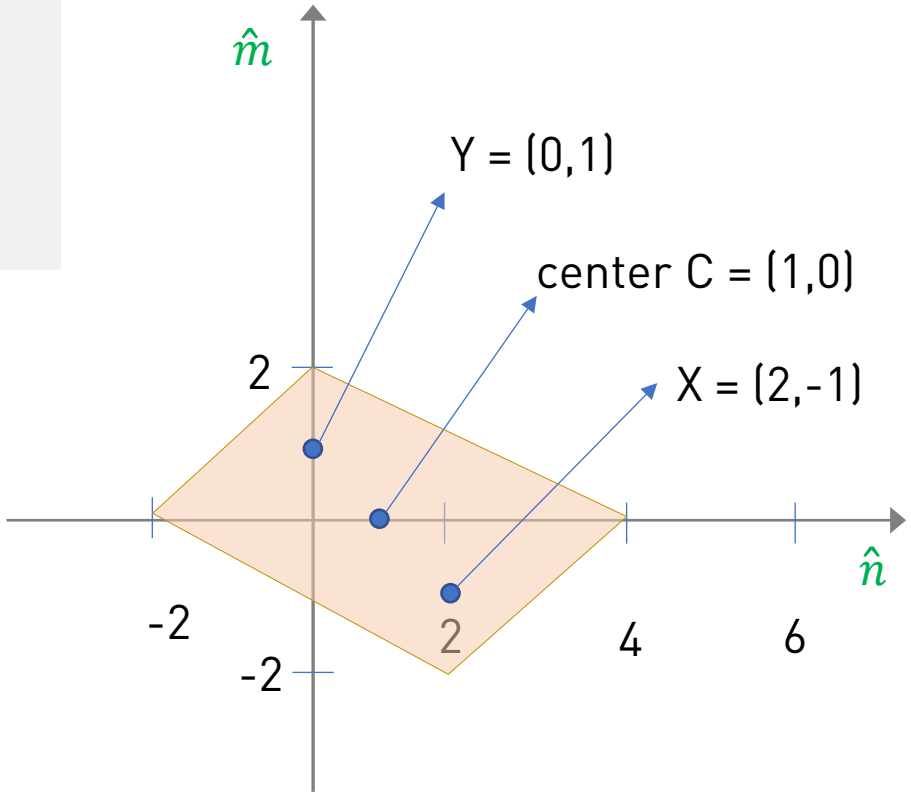
For example, a point X = (2,-1) can be flipped to obtain

a point Y = 2C – X = (0,1)

ψ =

$$\hat{n} = 1 - 2\epsilon_1 + \epsilon_2$$
$$\hat{m} = 0 + \epsilon_1 + \epsilon_2$$

$\gamma(\psi)$ is:



Example of Join ⊔

ψ_1 =

$$\hat{n} = 3 + \epsilon_1 + 2\,\epsilon_2$$
$$\hat{m} = 0 + \epsilon_1 + \epsilon_2$$

⊔

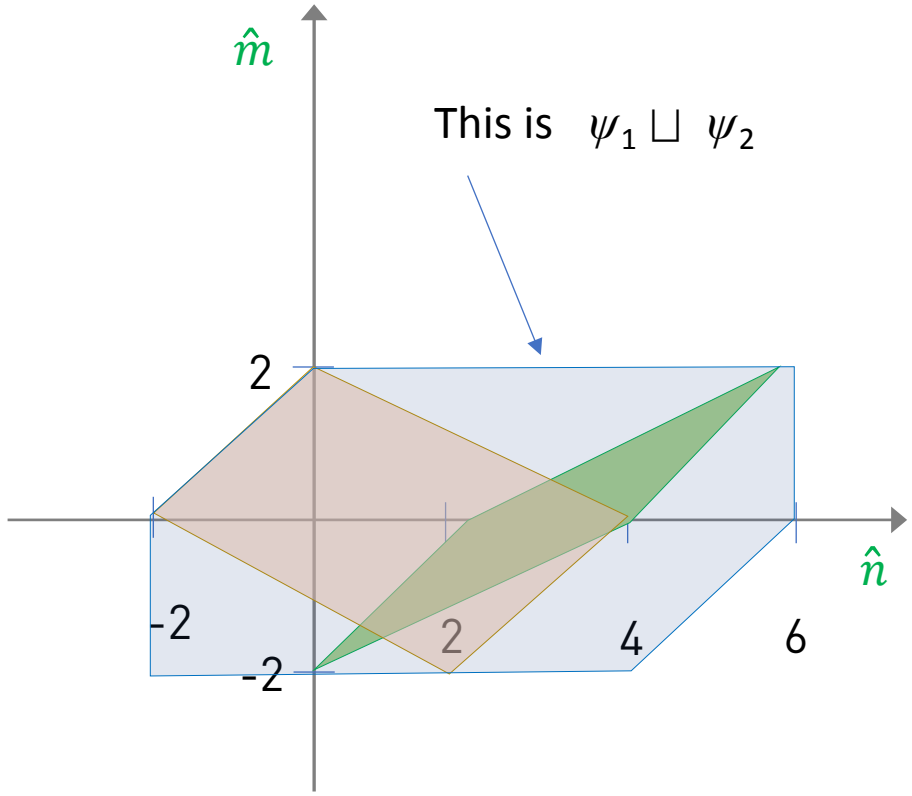
ψ_2 =

$$\hat{n} = 1 - 2\epsilon_1 + \epsilon_2$$
$$\hat{m} = 0 + \epsilon_1 + \epsilon_2$$

=

$$\hat{n} = 2 + \epsilon_2 + 3\epsilon_u$$
$$\hat{m} = 0 + \epsilon_1 + \epsilon_2$$

New error term is introduced



Zonotope Abstract Domain Operations

Multiplication by a constant real-valued constant C :

$$(a_0{}^n + \sum_{i=1}^{k} a_i{}^n \epsilon_i) \quad * \quad C = (C * a_0{}^n + \sum_{i=1}^{k} C * a_i{}^n \epsilon_i)$$

Adding two variables is done component-wise (abstract transformer is exact) :

$$(a_0{}^n + \sum_{i=1}^{k} a_i{}^n \epsilon_i) \quad + \quad (a_0{}^m + \sum_{i=1}^{k} a_i{}^m \epsilon_i) = \quad (a_0{}^n + a_0{}^m) + \sum_{i=1}^{k} (a_i{}^n + a_i{}^m) * \epsilon_i$$

Zonotope Abstract Domain Operations

Multiplication of two variables is non-linear, so an approximation is computed.
Here is one abstract transformer (not very precise, but often implemented):

$$(a_0{}^n + \sum_{i=1}^{k} a_i{}^n \, \epsilon_i) \; * \; (a_0{}^m + \sum_{i=1}^{k} a_i{}^m \, \epsilon_i) =$$

$$(a_0{}^n * a_0{}^m) + \sum_{i=1}^{k} (a_i{}^n * a_0{}^m + a_i{}^m * a_0{}^n) * \epsilon_i + \sum_{i=1}^{k} \sum_{j=1}^{k} a_i{}^m * a_j{}^n * \overbrace{\epsilon_i * \epsilon_j}$$

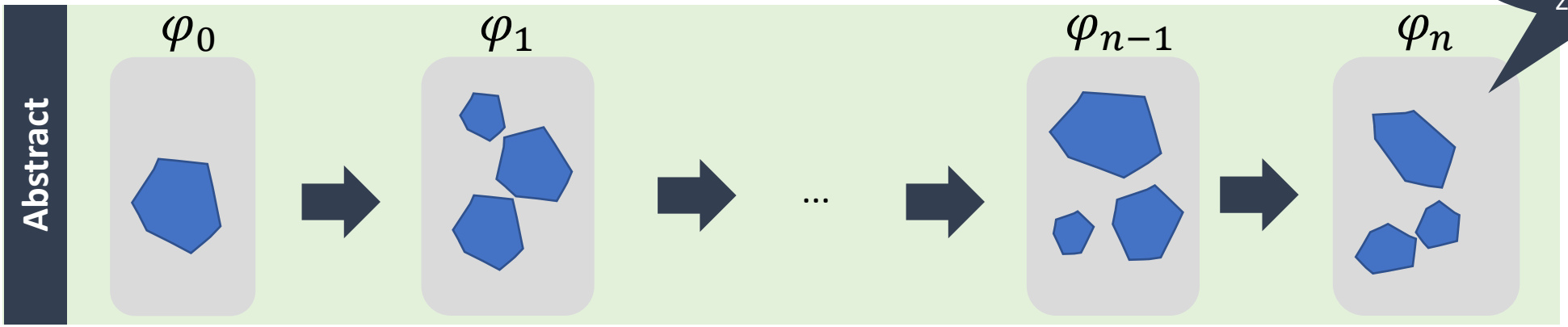
Again, this becomes a fresh variable $\epsilon_{i,j}$

$$\epsilon_{i,j} \in [-1,1] \; if \;\; i \neq j$$

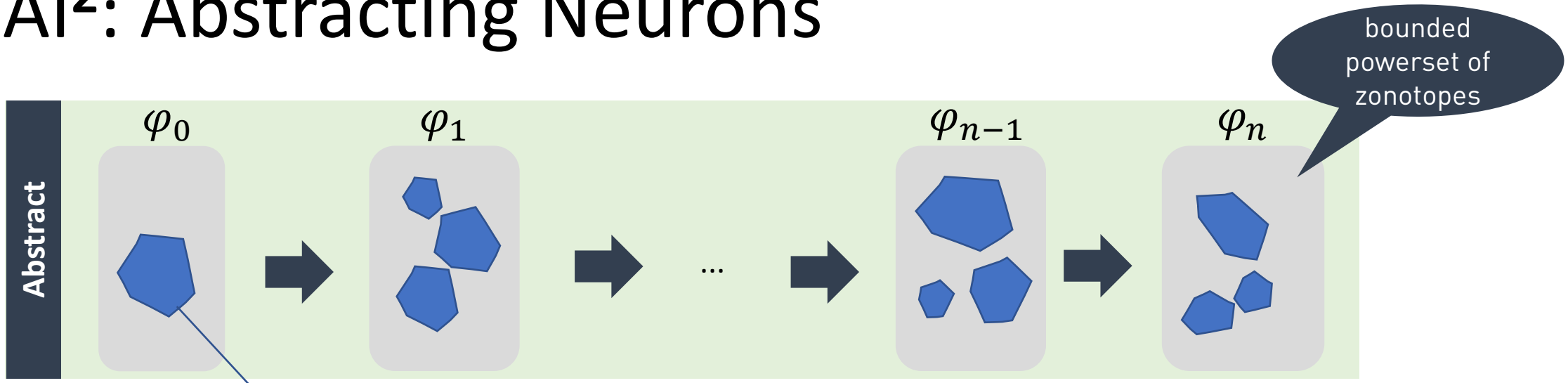
$$\epsilon_{i,j} \in [0,1] \; if \;\; i = j$$

Join and Meet are a little more elaborate, and the domain is not closed under these.

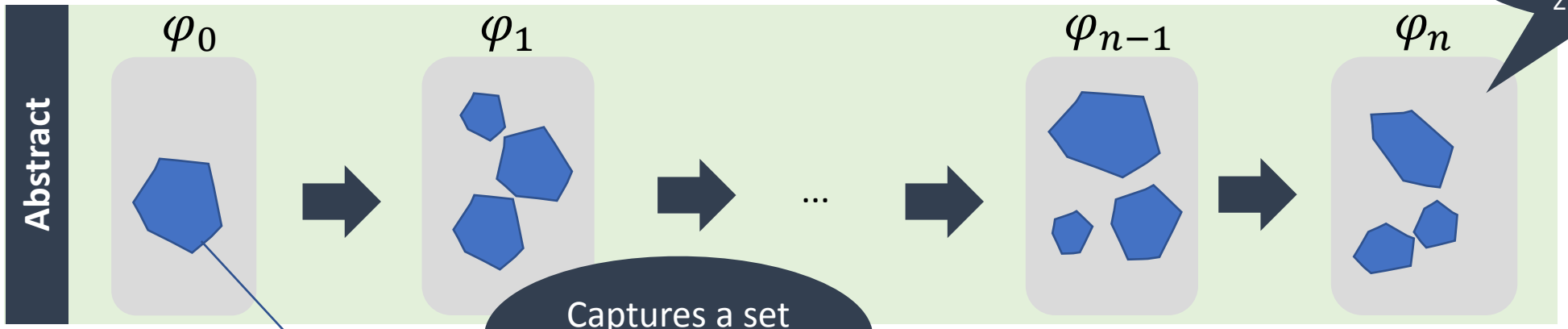
AI²: Abstracting Neurons



AI²: Abstracting Neurons



AI²: Abstracting Neurons

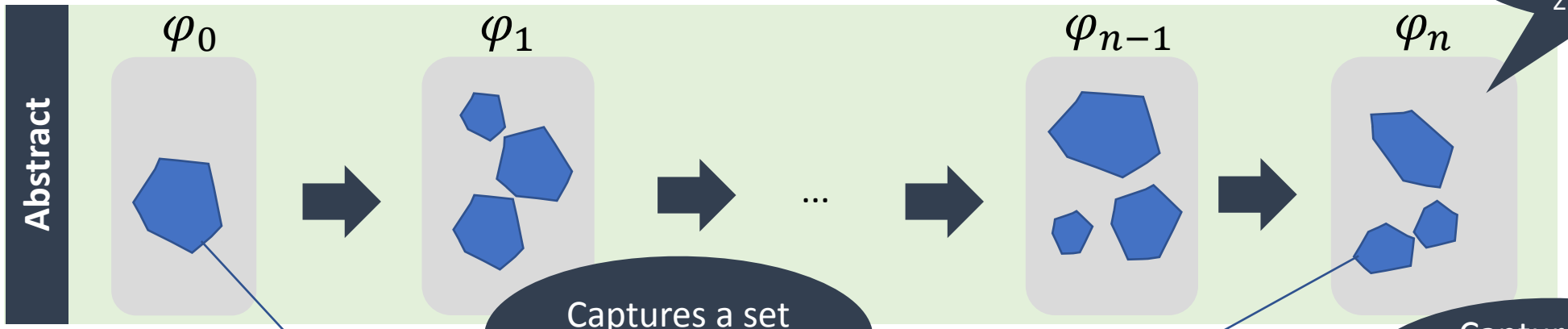


Captures a set of images

Robustness specification φ_0

$x_0 = 0$
$x_1 = 0.975 + 0.025\epsilon_1$
$x_2 = 0.125$
...
$x_{784} = 0.938 + 0.062\epsilon_{784}$
$\forall i. \epsilon_i \in [0,1]$

AI²: Abstracting Neurons



Captures a set of images

Captures all possible output vectors

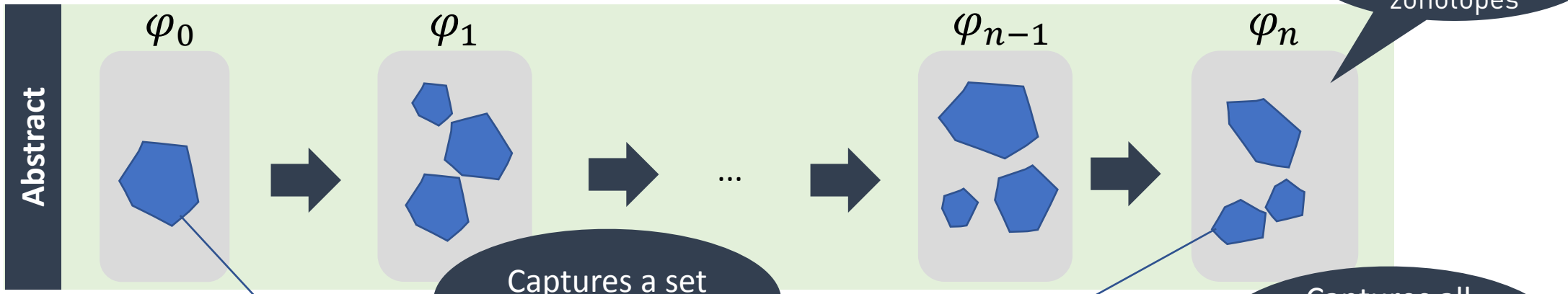
Robustness specification φ_0

$x_0 = 0$
$x_1 = 0.975 + 0.025\epsilon_1$
$x_2 = 0.125$
...
$x_{784} = 0.938 + 0.062\epsilon_{784}$
$\forall i. \epsilon_i \in [0,1]$

Output constraint φ_n

$x_0 = 0$
$x_1 = 2.60 + 0.015\epsilon_0 + 0.023\epsilon_1 + 5.181\epsilon_2 + \cdots$
$x_2 = 4.63 - 0.005\epsilon_0 - 0.006\epsilon_1 + 0.023\epsilon_1 + \cdots$
...
$x_9 = 0.12 - 0.125\epsilon_0 + 0.102\epsilon_1 + 3.012\epsilon_1 + \cdots$
$\forall i. \epsilon_i \in [0,1]$

AI²: Abstracting Neurons



Captures a set of images

Captures all possible output vectors

Robustness specification φ_0

$x_0 = 0$
$x_1 = 0.975 + 0.025\epsilon_1$
$x_2 = 0.125$
...
$x_{784} = 0.938 + 0.062\epsilon_{784}$
$\forall i.\, \epsilon_i \in [0,1]$

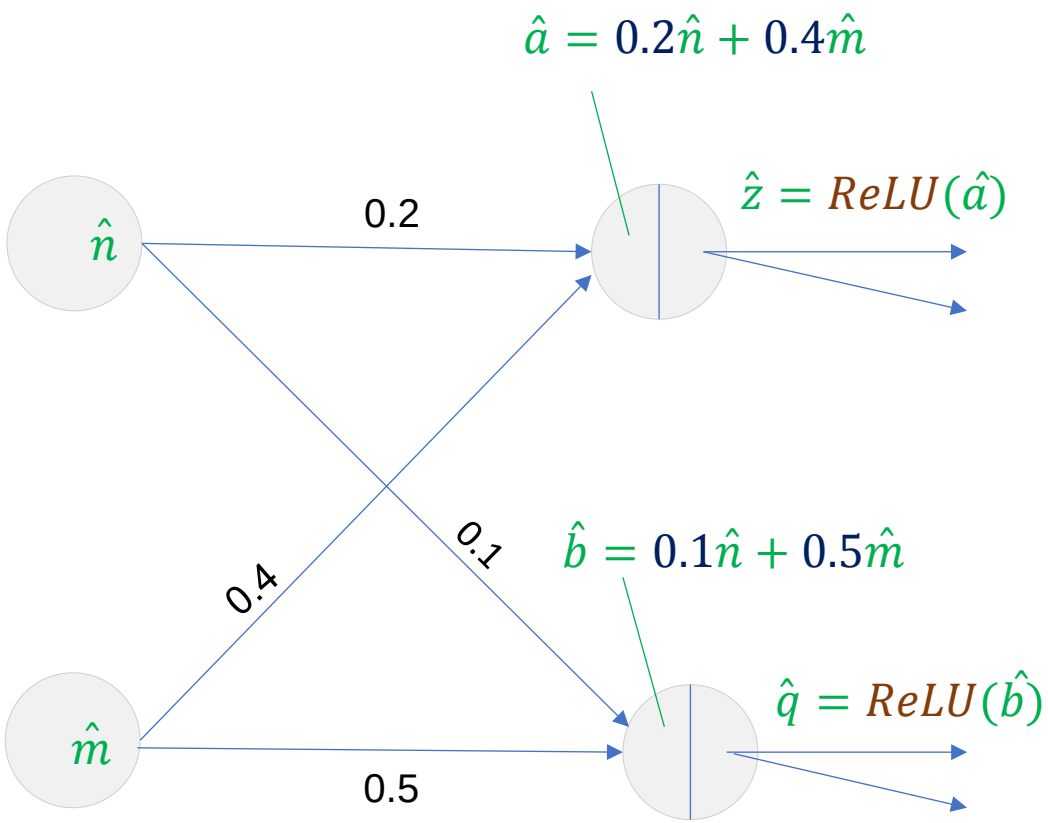
Output constraint φ_n

$x_0 = 0$
$x_1 = 2.60 + 0.015\epsilon_0 + 0.023\epsilon_1 + 5.181\epsilon_2 + \cdots$
$x_2 = 4.63 - 0.005\epsilon_0 - 0.006\epsilon_1 + 0.023\epsilon_2 + \cdots$
...
$x_9 = 0.12 - 0.125\epsilon_0 + 0.102\epsilon_1 + 3.012\epsilon_2 + \cdots$
$\forall i.\, \epsilon_i \in [0,1]$

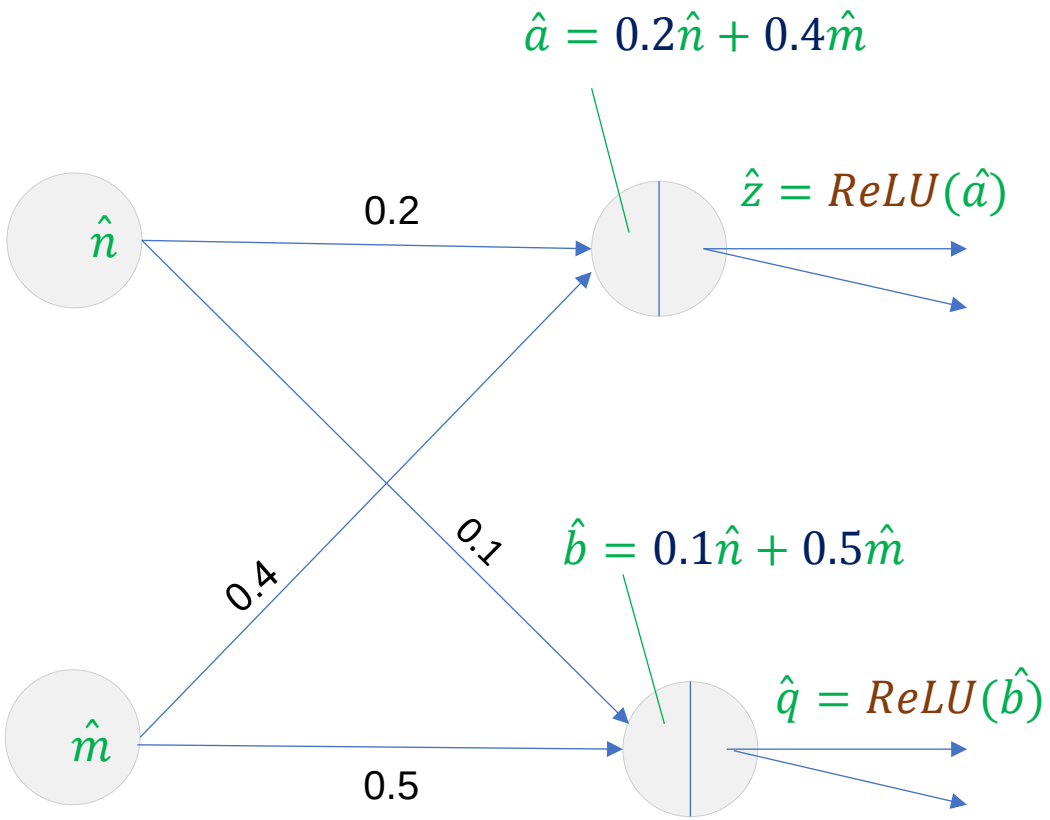
Label i is possible iff: $\varphi_n \sqcap \{\forall j.\, x_i \geq x_j\} \neq \bot$

(board)

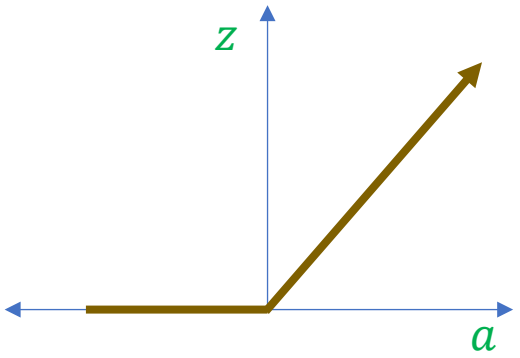
Abstract Neuron Transformers



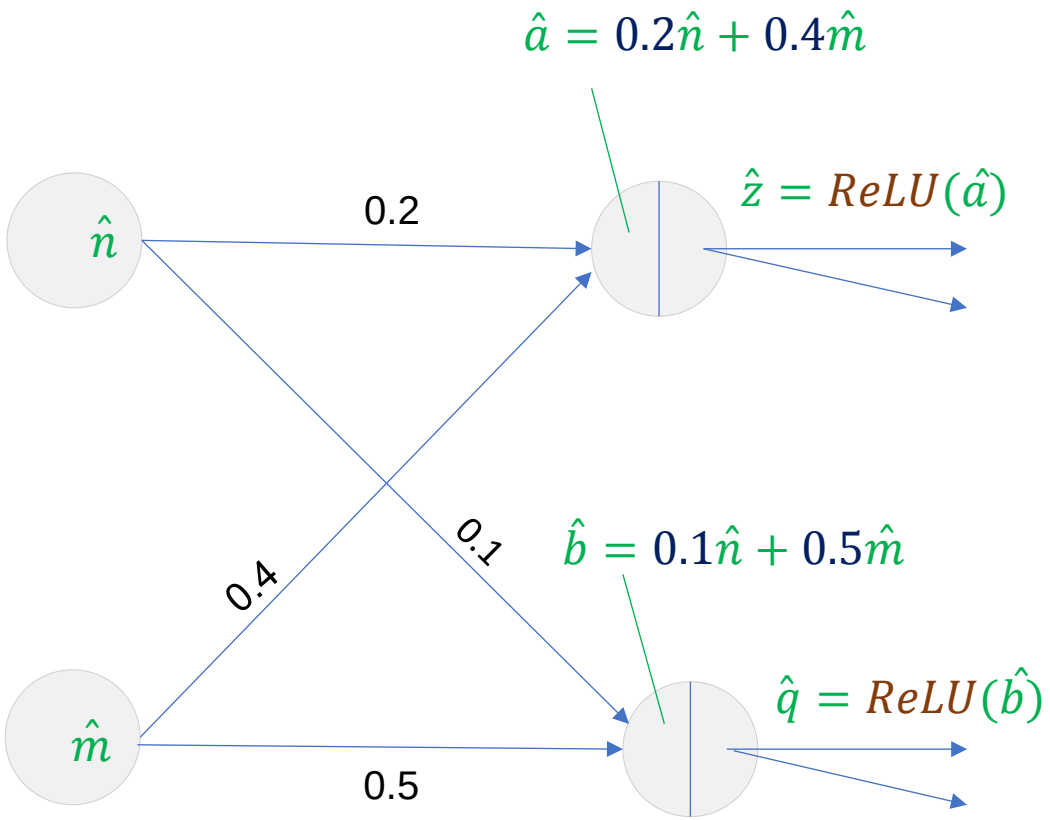
Abstract Neuron Transformers



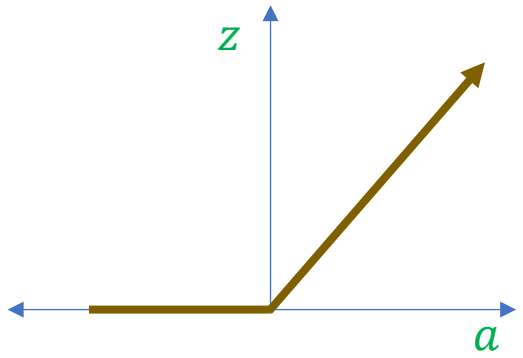
Activation function: $z = ReLU(a) = \max(0, a)$



Abstract Neuron Transformers



Activation function: $z = ReLU(a) = \max(0, a)$



ReLU abstract transformer:

$$f^{\#}_{ReLU} = f^{\#}_{k} \circ \cdots \circ f^{\#}_{1}$$

$$f^{\#}_{i}(\psi) = (\psi \sqcap \{x_i \geq 0\}) \sqcup \psi_0$$

$$\psi_0 = \begin{cases} [\![x_i = 0]\!](\psi) & \text{if } (\psi \sqcap \{x_i < 0\}) \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

The AI² System

Supports neural networks with:

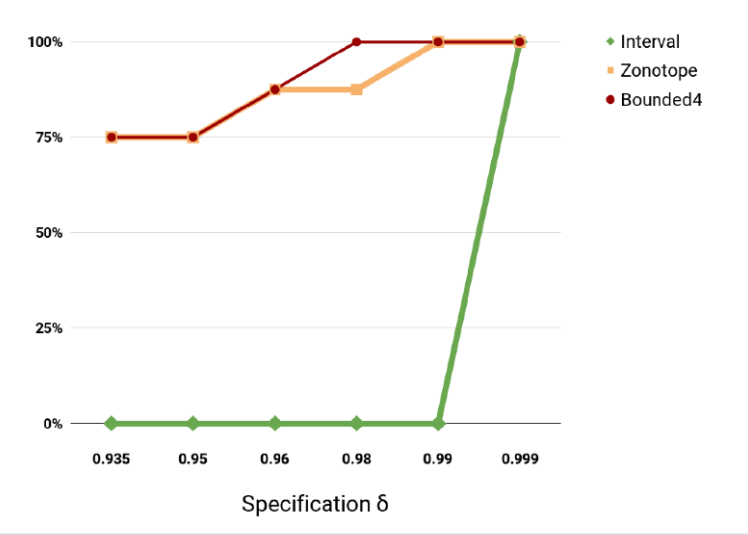
Layers: Fully-connected, convolutional, max-pooling, flattening
Activation functions: ReLU

Supported numerical domains:

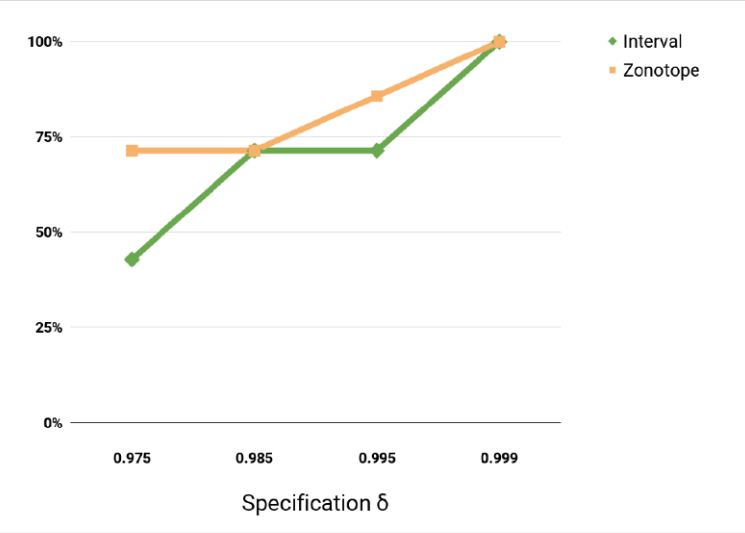
Intervals, Zonotopes, Polyhedra, Bounded powerset domain

Experimental Results

MNIST ConvNet 6 layers, 15K neurons



CIFAR-10 ConvNet 6 layers, 57K neurons



Summary

- Brief introduction to abstract interpretation (approximating functions)
- Example of A.I. on intervals
- Proving robustness of neural networks with A.I. and Zonotope domain
- Open problems: addition activation functions (sigmoid), different kinds of networks, faster Zonotope implementation