Formal Verification Of Symbolic And Connectionist AI: A Way Toward Higher Quality Software

Julien Girard-Satabin, Dorin Doncenco,
Zakaria Chihani

zakaria.chihani@cea.fr

ESSAI 2025
BRATISLAVA - SLOVAKIA
organized by EurAi

Ce travail a bénéficié d'une aide de l'État gérée par l'Agence Nationale de la Recherche au titre de France 2030 portant la référence « ANR-23-PEIA-0006 »

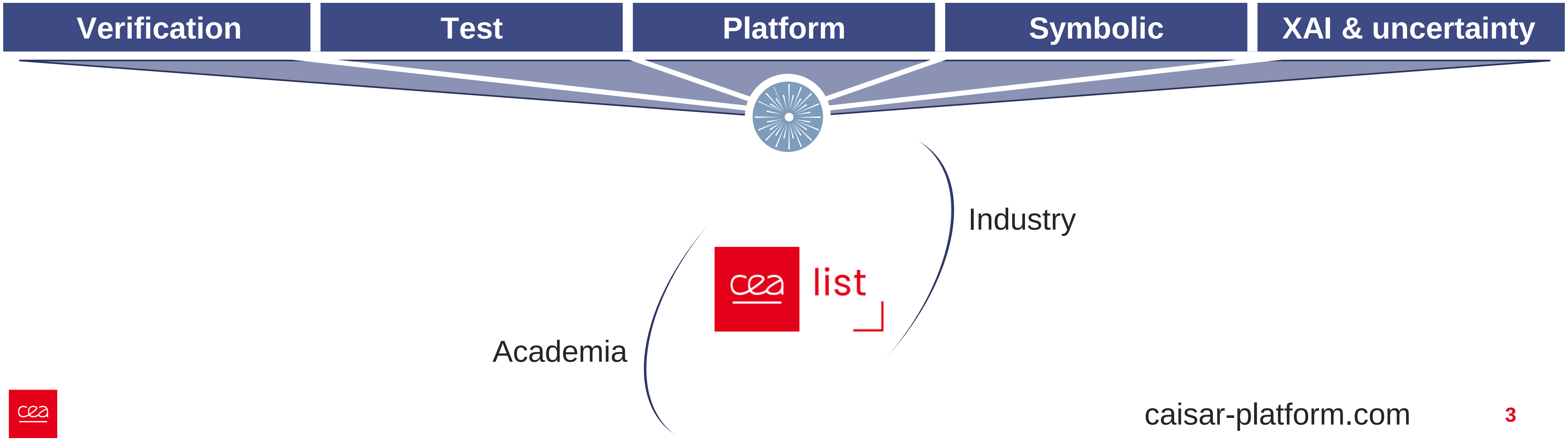
So what are Formal Methods

- Non snobbish definition
  - Math- and logic-based techniques with rigorously established theoretical foundations
  - Used for the specification, development, test and verification of software and hardware

- Why use formal methods ?
  - Non validated software can have dire consequences and mathematical analysis can contribute to the reliability and robustness
  - Some certification standards call for (e.g., DO-178C for avionics) or even **mandate** (e.g., ISO/IEC 15408) the use of formal methods

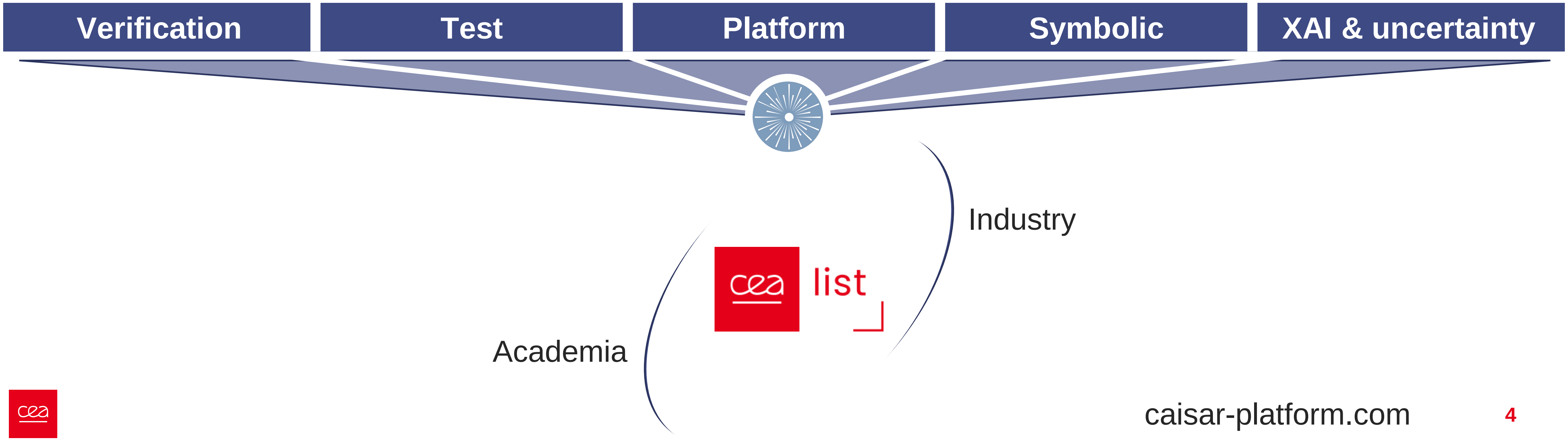
Our lab

- Formal methods for safety and security of software for decades
- Branched into AI trustworthiness in 2017
- About 10 permanent researchers, plus post docs, engineers, PhDs and interns
- Developing tools and methods



This session

- No particular background needed
- At the end the hope is you
  - Know some basic principles of some formal methods
  - Can imagine for what they can be used
  - Want to know more of the technical stuff



Rapid intro to FM

Examples for this talk:

- Property-based testing
- Abstract interpretation
- SMT Solving

So why Formal Methods

A critical system is a system whose failure may cause physical harm, economical losses or damage the environment

Goal: guarantee that the system respects a *safety specification* ϕ

So why Formal Methods

The Ariane 5 reused the inertial reference platform from the Ariane 4, but the Ariane 5's flight path differed considerably from the previous models.

The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to **overflow** and cause a hardware exception.

The subsequent automated analysis of the Ariane code (written in Ada) was the first example of large-scale static code analysis by abstract interpretation.

So why Formal Methods

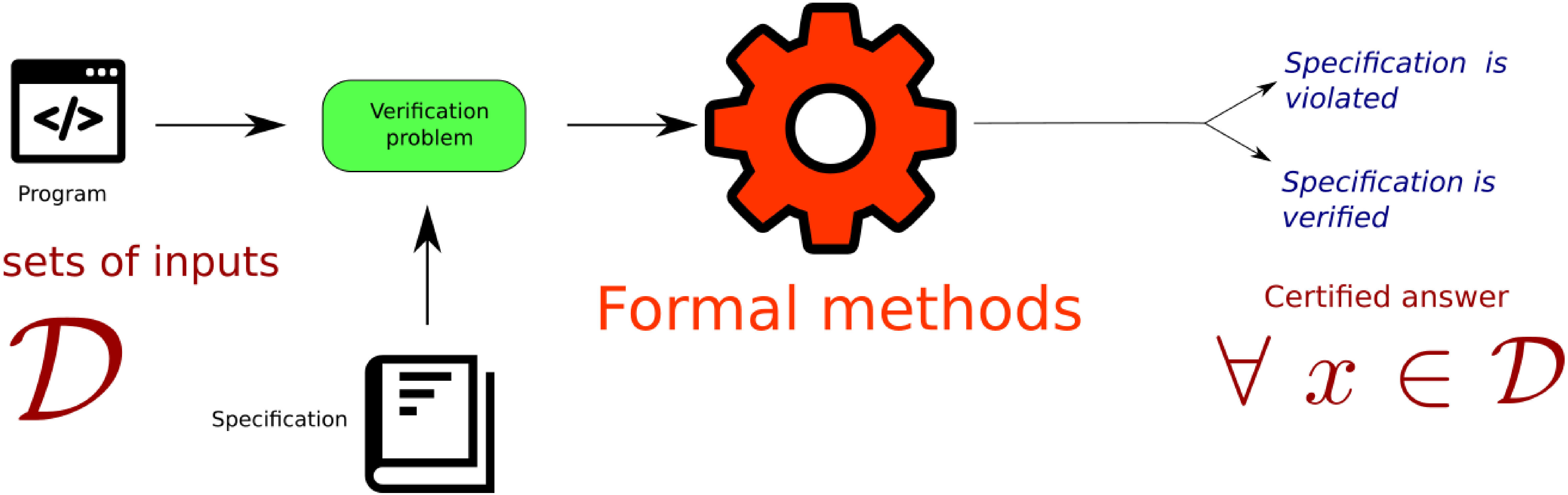
The Ariane 5 reused the inertial reference platform from the Ariane 4, but the Ariane 5's flight path differed considerably from the previous models.

The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to **overflow** and cause a hardware exception.

The subsequent automated analysis of the Ariane code (written in Ada) was the first example of large-scale static code analysis by abstract interpretation.

If you don’t use formal methods, your failure will be in every presentation of every FM conference.

So HOW Formal Methods (usually)



So WHO Formal Methods

Astrée Software

PolySpace TECHNOLOGIES

frama C Software Analyzers

ESTEREL Technologies

CompCert®

bugseng

MODELICA

Microsoft Research

z3

So for WHOM Formal Methods

IRSN
INSTITUT DE RADIOPROTECTION
ET DE SÛRETÉ NUCLÉAIRE

AIRBUS

RATP

IBM

AMD

Schneider Electric

DGA

ALSTOM

eDF

THALES

SIEMENS

NASA

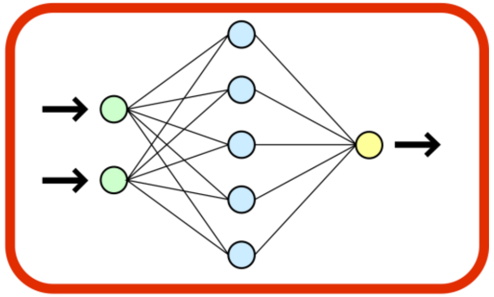
intel

SAFRAN

STMicroelectronics

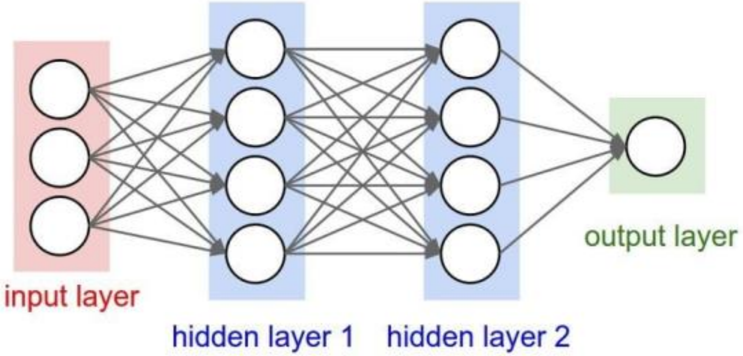
cea

So WHICH Formal Methods

Classical programs	Deep learning
	
Explicit control flow	Generated control flow
Explicit specifications	Implicit specifications
Abstractions and well known concepts	Very few abstractions and reusability
Needs to be robust	Needs to be robust

So WHICH Formal Method for AI

Human written code



Machine learned code

A brief history of wrong predictions

In 1979:

“[P]rogram verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs”

“Social processes and proofs of theorems and programs”, Communications of ACM. By Richard De Millo, Richard Lipton, and Alan Perlis.

A brief history of wrong predictions

In 1979:

"[P]rogram verification is bound to fail. We can't see how it's going to be able to affect anyone's confidence about programs"

"Social processes and proofs of theorems and programs", Communications of ACM.
By Richard De Millo, Richard Lipton, and Alan Perlis.

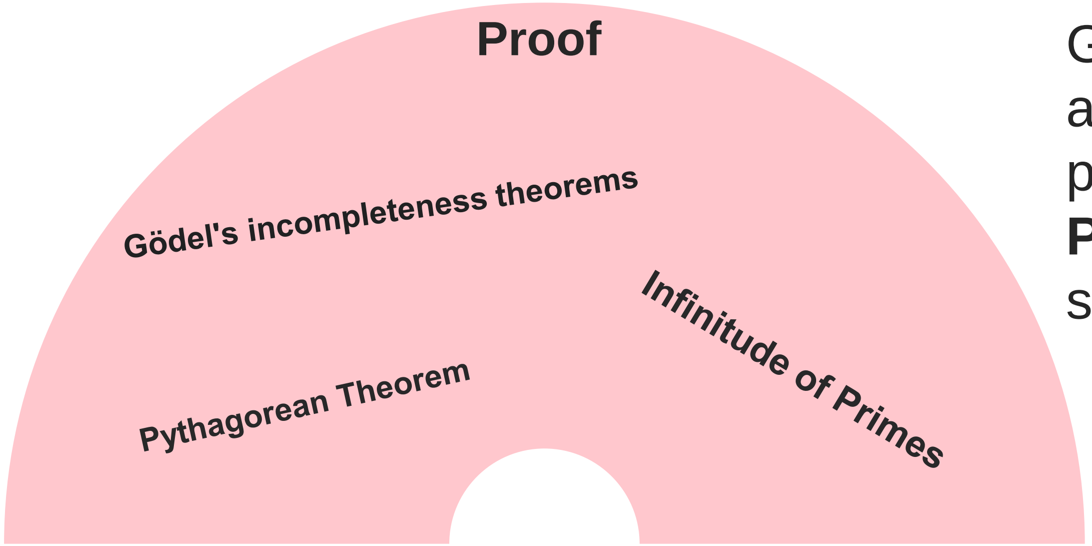
- Distinguished Professor of Computing at the Georgia Tech
- VP and CTO of Hewlett-Packard

- Yale, Berkeley, Princeton, Georgia Tech
- Knuth Prize winner

- ACM, Carnegie Mellon, Yale, Purdue
- The first recipient of the Turing Award

Shifting the social process

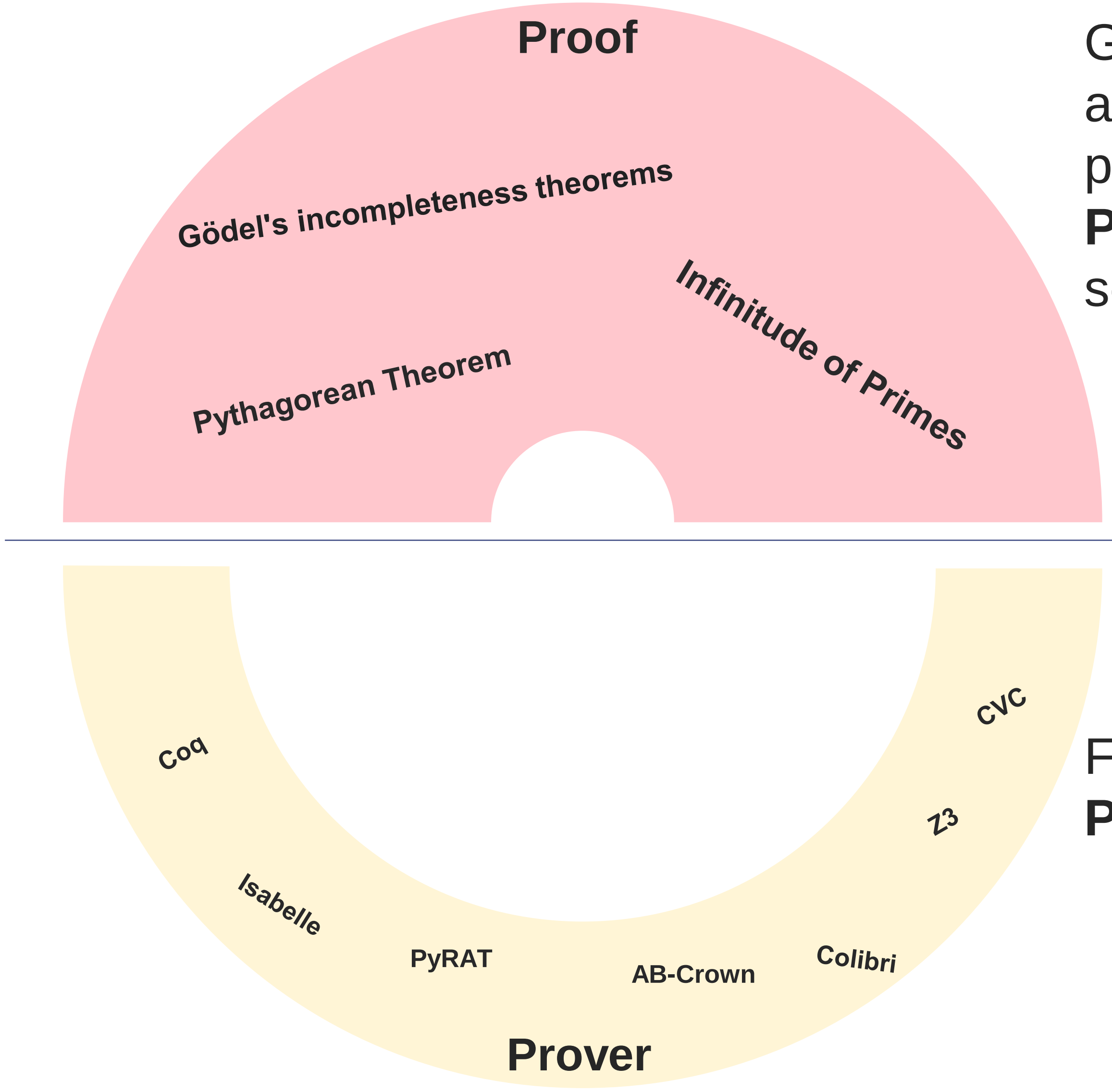
- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring



Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Shifting the social process

- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring

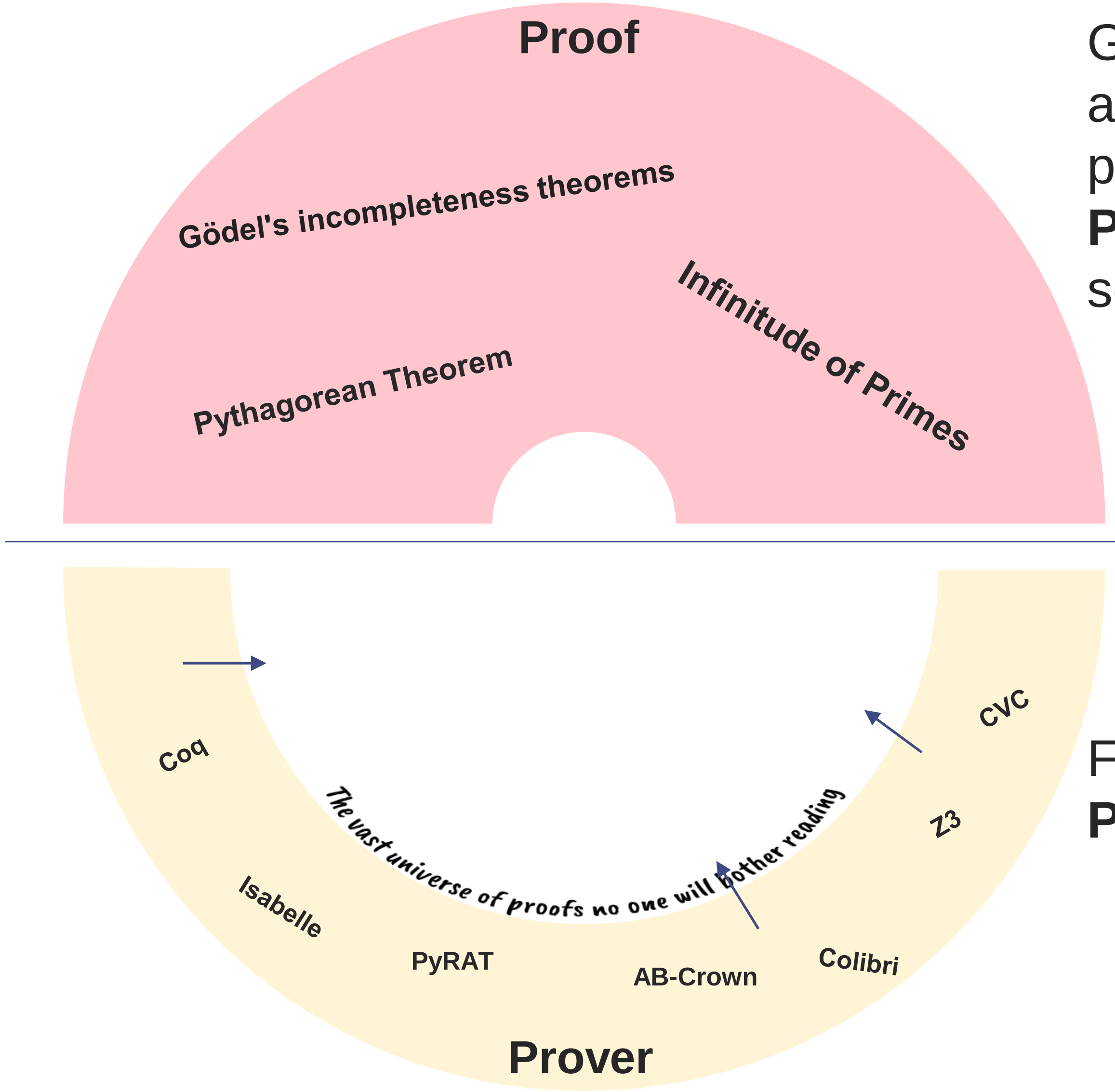


Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Formal methods:
Prover is the focus

Shifting the social process

- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring

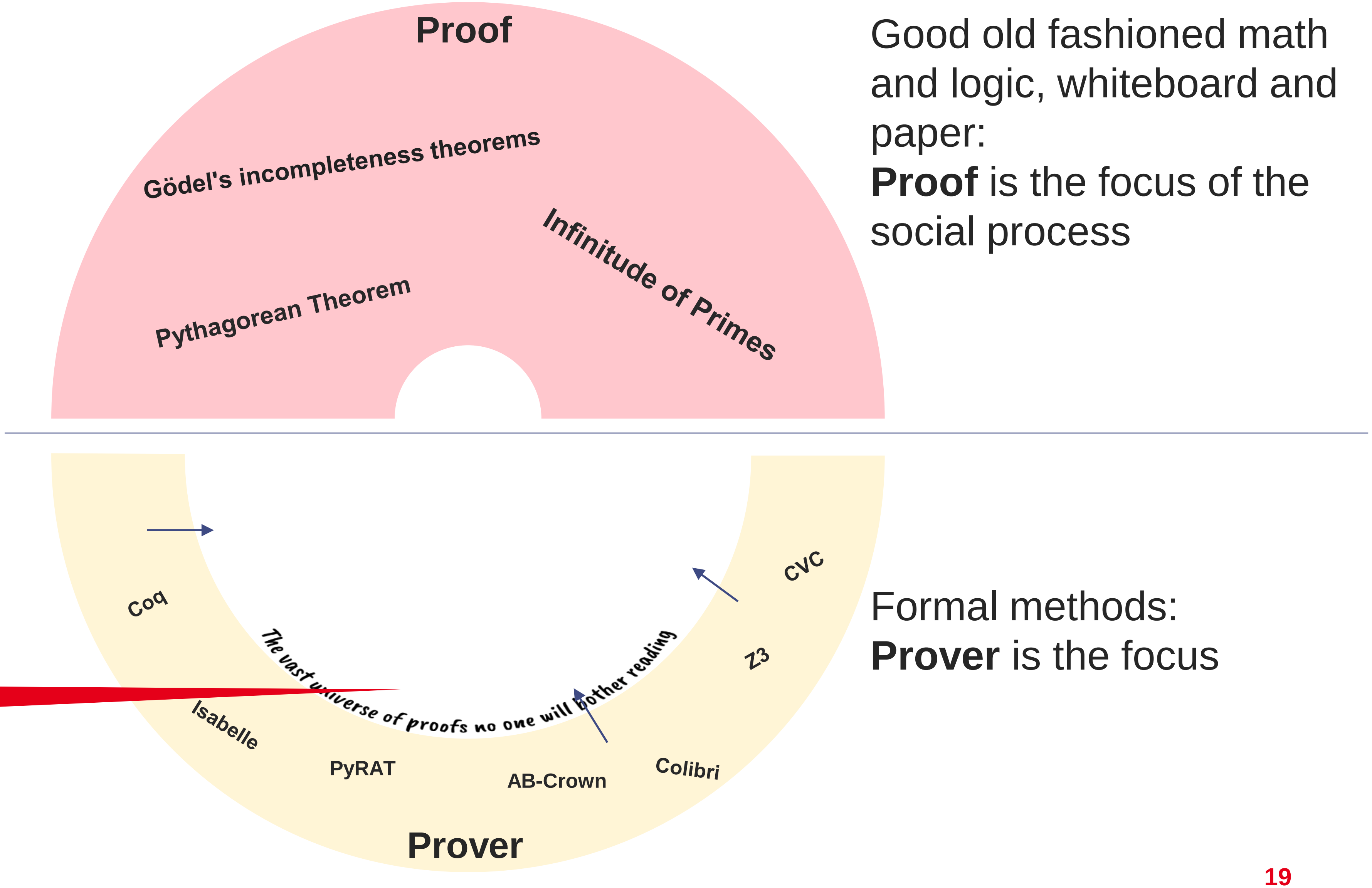


Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Formal methods:
Prover is the focus

Shifting the social process

- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring



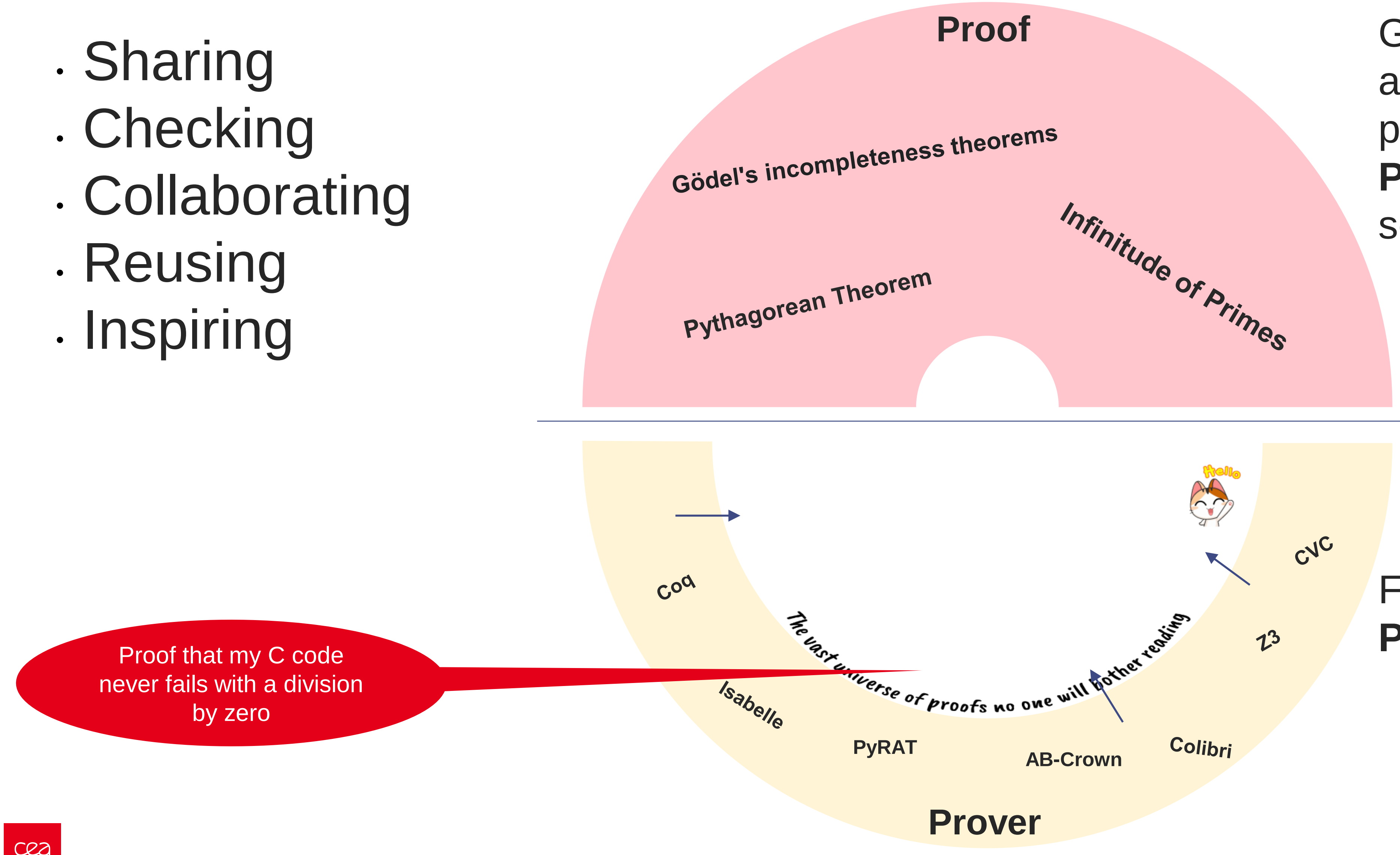
Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Formal methods:
Prover is the focus

Proof that my C code never fails with a division by zero

Shifting the social process

- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring

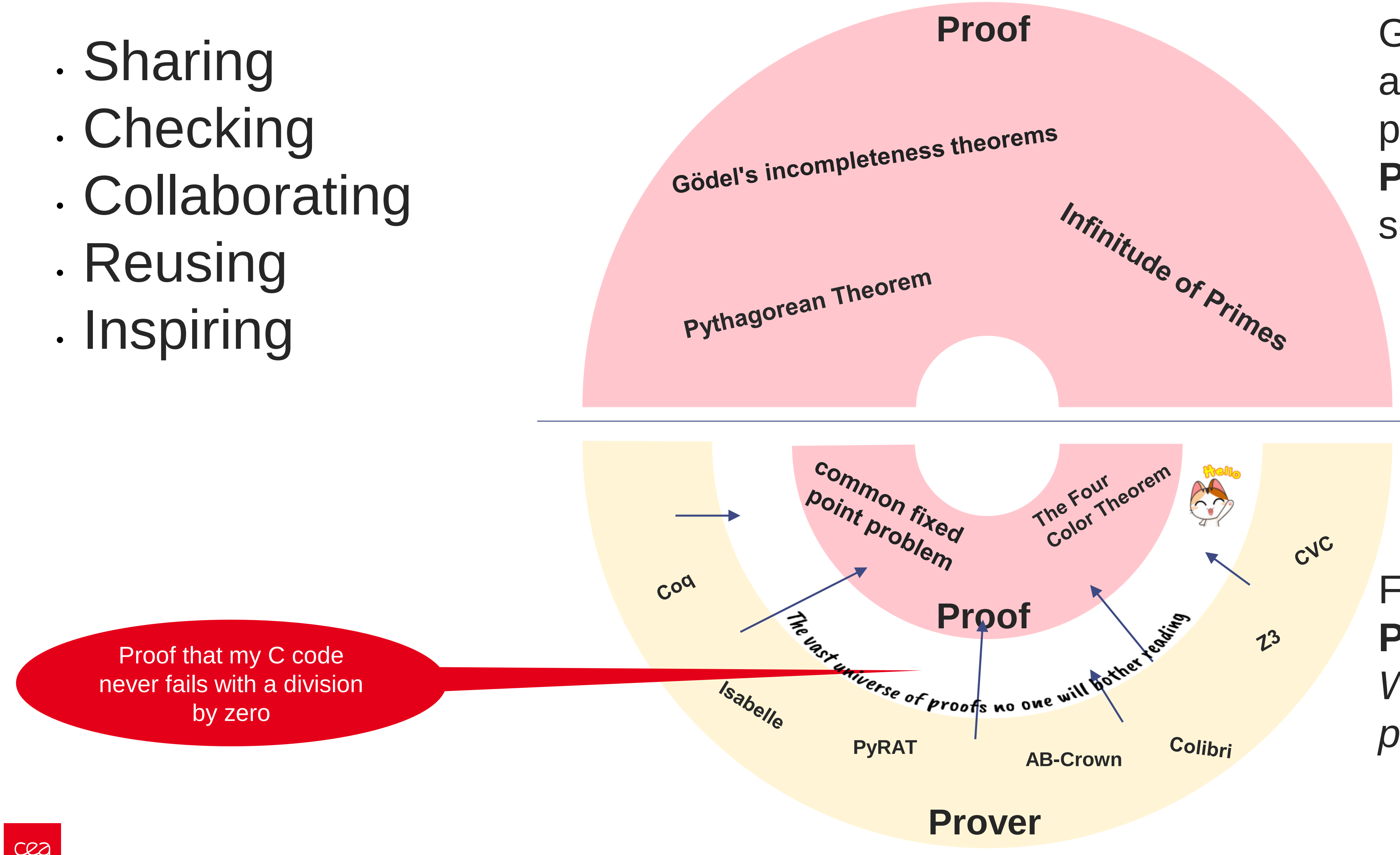


Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Formal methods:
Prover is the focus

Shifting the social process

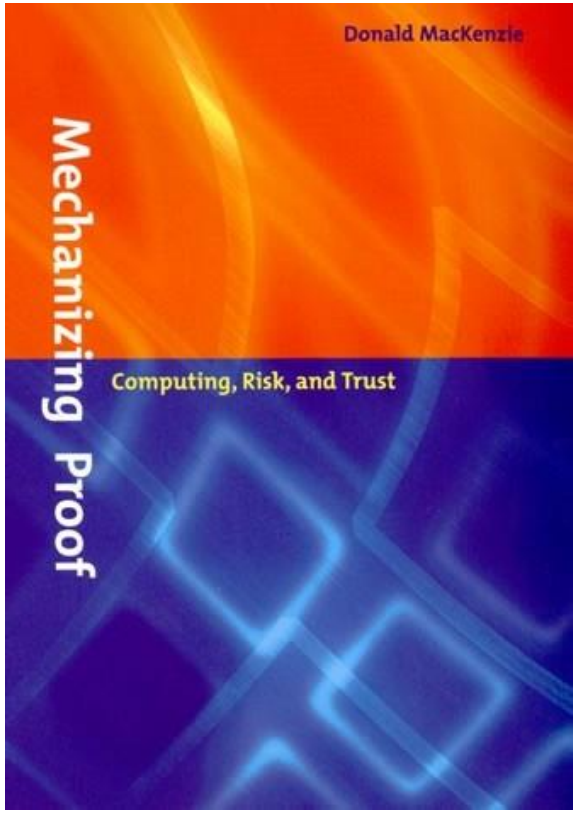
- Sharing
- Checking
- Collaborating
- Reusing
- Inspiring



Good old fashioned math and logic, whiteboard and paper:
Proof is the focus of the social process

Formal methods:
Prover is the focus
With some focus on proofs

Valid scepticism

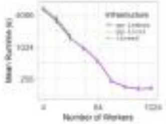


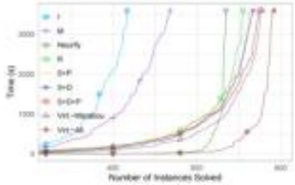
- The first solvers and analyzers were **not** efficient or scalable.
- For example, today's SAT solvers can automatically solve problem instances involving **tens of thousands of variables and millions of constraints**.
- But it wasn't always the case! We needed to invent DPLL, CDCL, Symmetry breaking, two-watched literals, WalkSAT, adaptive branching, random restarts, portfolio, divide-and-conquer, parallel local search…

Restarting Formal Methods for AI

Cambrian explosion: Just in the past few years, more than 20 tools.
Competition for resources: Each paper published increases the scalability.
Cross-fertilization: Good ideas from one tool are implemented in others.
Niche creation: Some solvers are more specialized into particular models and type of properties.
Adaptative Pressure: New models, new architectures, and in general new AI-technologies are born every year and the tools to validate them must keep up.
Domestication: ML practitioners should be made aware of the choices in implementation that can make their models more amenable to FM, so that they can factor this aspect in their decision process.

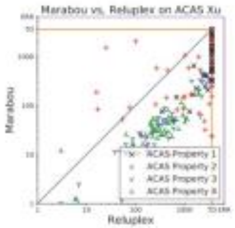
Restarting Formal Methods for AI

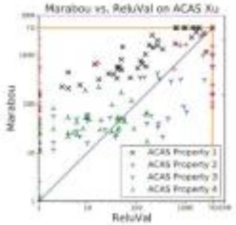




Parallelization Techniques for Verifying Neural Networks

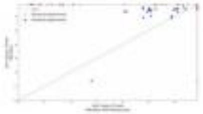
2020- Parallelism





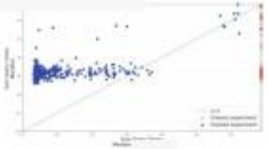
The Marabou Framework for Verification and Analysis of Deep Neural Networks

2019- Dedicated simplex etc.



An Abstraction-Based Framework for Neural Network Verification

2020- Abstractions



	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9

Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks

2017- inception

Restarting Formal Methods for AI

- Artificial Intelligence Safety Engineering (WAISE, at SafeComp)
- AISafety (at IJCAI)
- Safe AI (at AAAI)
- Verification of Neural Networks (VNN, at AAAI or CAV)
- Formal Methods for ML-Enabled Autonomous Systems (FoMLAS, at CAV)
- Machine Learning with Guarantees (ML with Guarantees, at NeurIPS)
- Safe Machine Learning (SafeML, at ICLR)
- Privacy in Machine Learning (PriML, at NeurIPS)
- Security and Safety in Machine Learning Systems (AISecure, at ICLR)
- Dependable and Secure Machine Learning (DSML, at DSN)

General picture

Characterization of (AI) trustworthiness

A three-players game

A three-players game

Developer's side

- What is the architecture of the software, how can it be modified to be more amenable to verification, will these modifications cost too much ?
  (Activation functions of NN, kernel function of SVM, etc.)

Object to certify

- What to verify, how to formally specify it, how is it decomposed in smaller bits?
  (Robustness, metamorphism, behavior specification, etc.)

Properties to verify

Validator's side

- How to verify, what methods fit my problem, can the tools be helped with heuristics?
  (Abstract interpretation, SMT solving, symbolic execution, Constraint programming, etc.)

Methods and tools

The Object

What is the architecture of the software,
how can it be modified to be more amenable to verification,
will these modifications cost too much

The object

Several architecture choices influence what we can do:

- Activation function ? Can we use ReLU ?
- Connectivity ? Fewer connections lead to more scalable verification.
- Number of parameters ? Can we prune the network ?
- The objective function ? Can we modify it so that it does not only learn the goal of the NN but also “some” information to guide the provers’ heuristics ?
- How was it trained ?

The Property

What to verify,
how to *formally* specify it,
how is it decomposed in smaller bits

Types of properties depend on the structure

Formally specifiable (structured data)

- **Semantics** (directly related to the domain) can be described through math and logic
    - The input is a distance, a speed, a physical property (e.g., an observable), a tabular data, (e.g., an age), etc.
- Can be proved on all possible inputs

Types of properties depend on the structure

Formally specifiable (structured data)

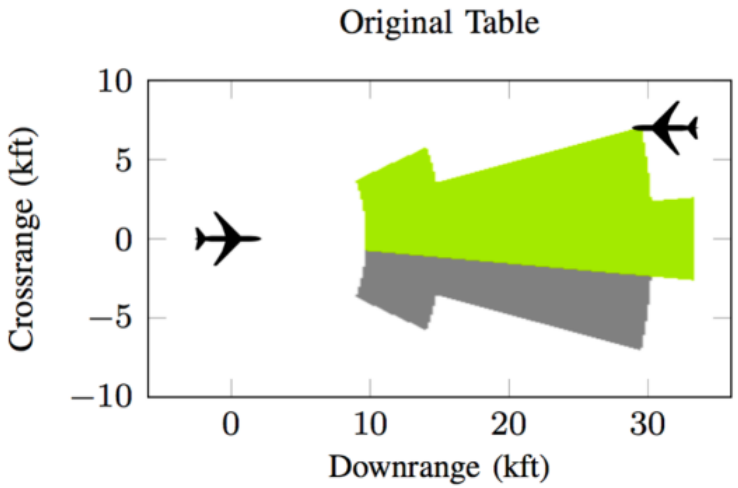
- **Semantics** (directly related to the domain) can be described through math and logic
  - The input is a distance, a speed, a physical property (e.g., an observable), a tabular data, (e.g., an age), etc.
- Can be proved on all possible inputs

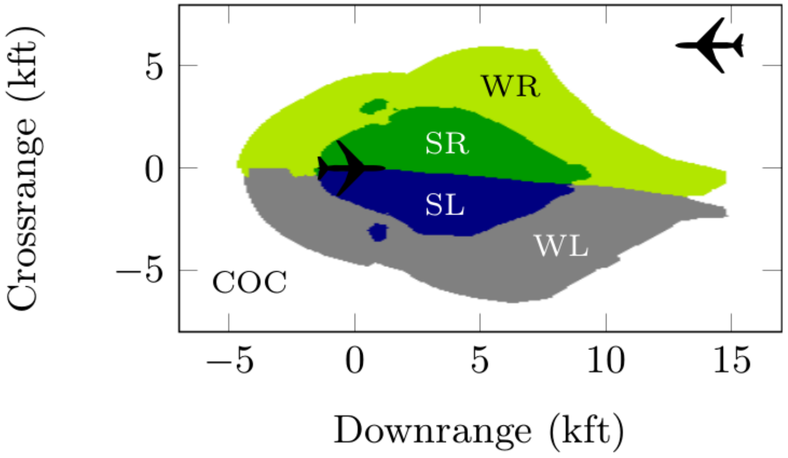
Not directly specifiable (unstructured data)

- Semantics is an **emerging property** (cf *a molecule of water isn’t wet*). e.g. this is a cat. The input is a collection of semanticless values (a pixel) that, together, can form a meaning
- Only abstracted properties can be described through math and logic (e.g. “distance” between two images)
- Cannot realistically be proved on all possible inputs

Structured data

Huge look-up tables replaced with NNs to save space.



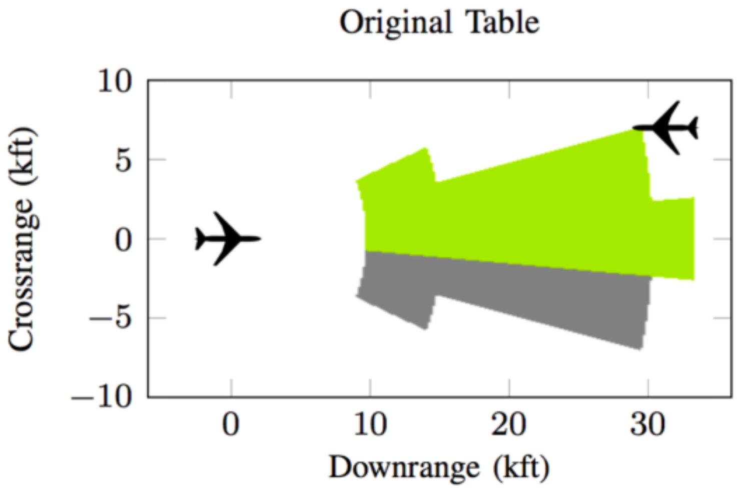


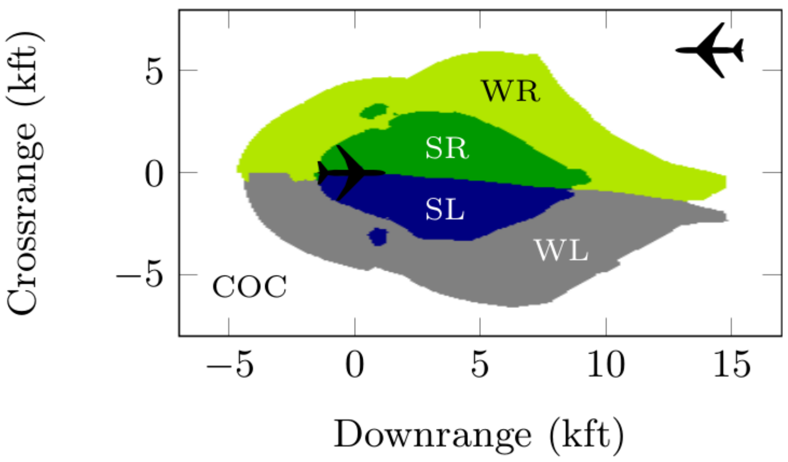
Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks (2017, Katz, Barrett, Dill, Julian, Kochenderfer)

Structured data

Huge look-up tables replaced with NNs to save space.

Description: If the intruder is near and approaching from the left, the network advises "strong right".



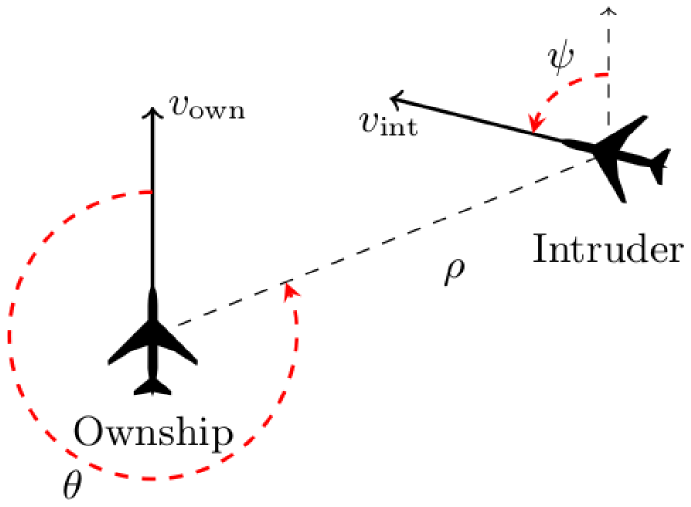


Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks (2017, Katz, Barrett, Dill, Julian, Kochenderfer)

Structured data

Huge look-up tables replaced with NNs to save space.

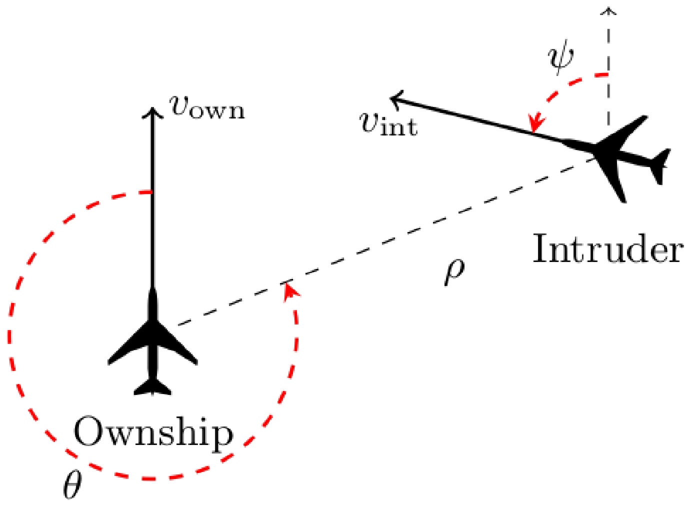
Description: If the intruder is near and approaching from the left, the network advises "strong right".



Structured data

Huge look-up tables replaced with NNs to save space.

Description: If the intruder is near and approaching from the left, the network advises "strong right".



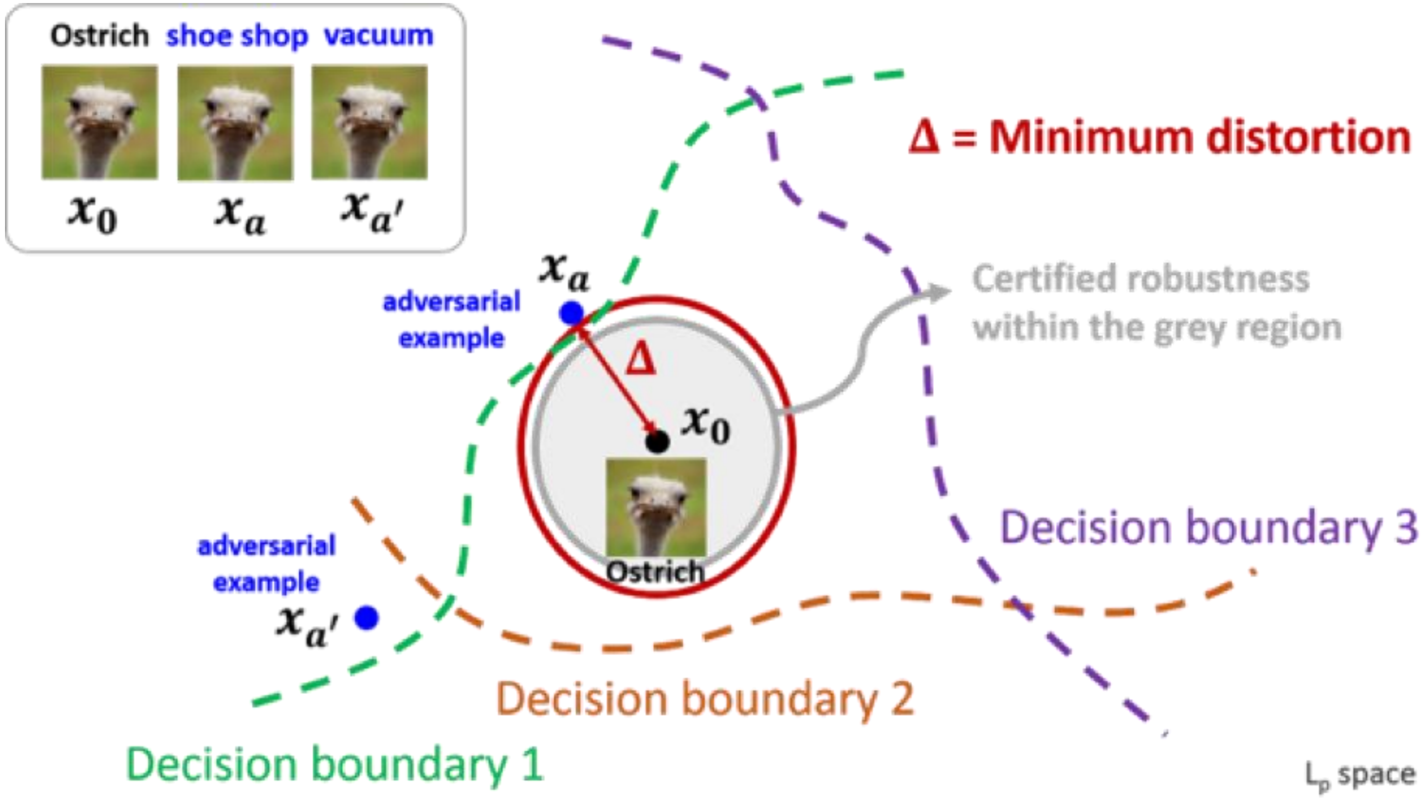
Input constraints: $250 \leq \rho \leq 400$, $0.2 \leq \theta \leq 0.4$, $-3.141592 \leq \psi \leq -3.141592 + 0.005$, $100 \leq v_{\text{own}} \leq 400$, $0 \leq v_{\text{int}} \leq 400$.

Unstructured data

Images, sounds…

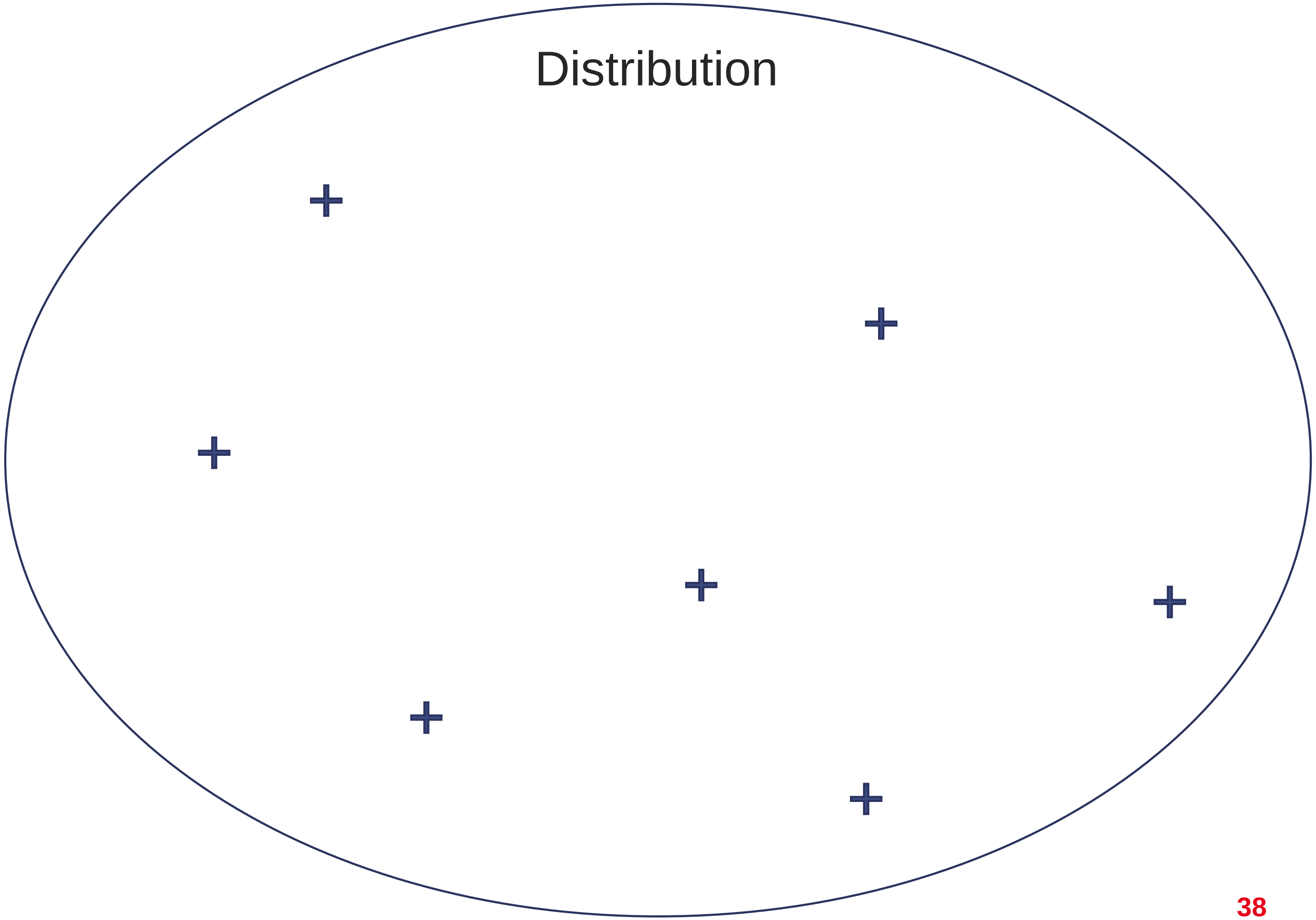
How can you formally specify the desirable property “if pedestrian on the road, then brake”?

But you can have different properties : considering notions such as “input space” and “points that are close to each other”, a property could be “if two inputs are closer than some measure d, they should be classified the same”.



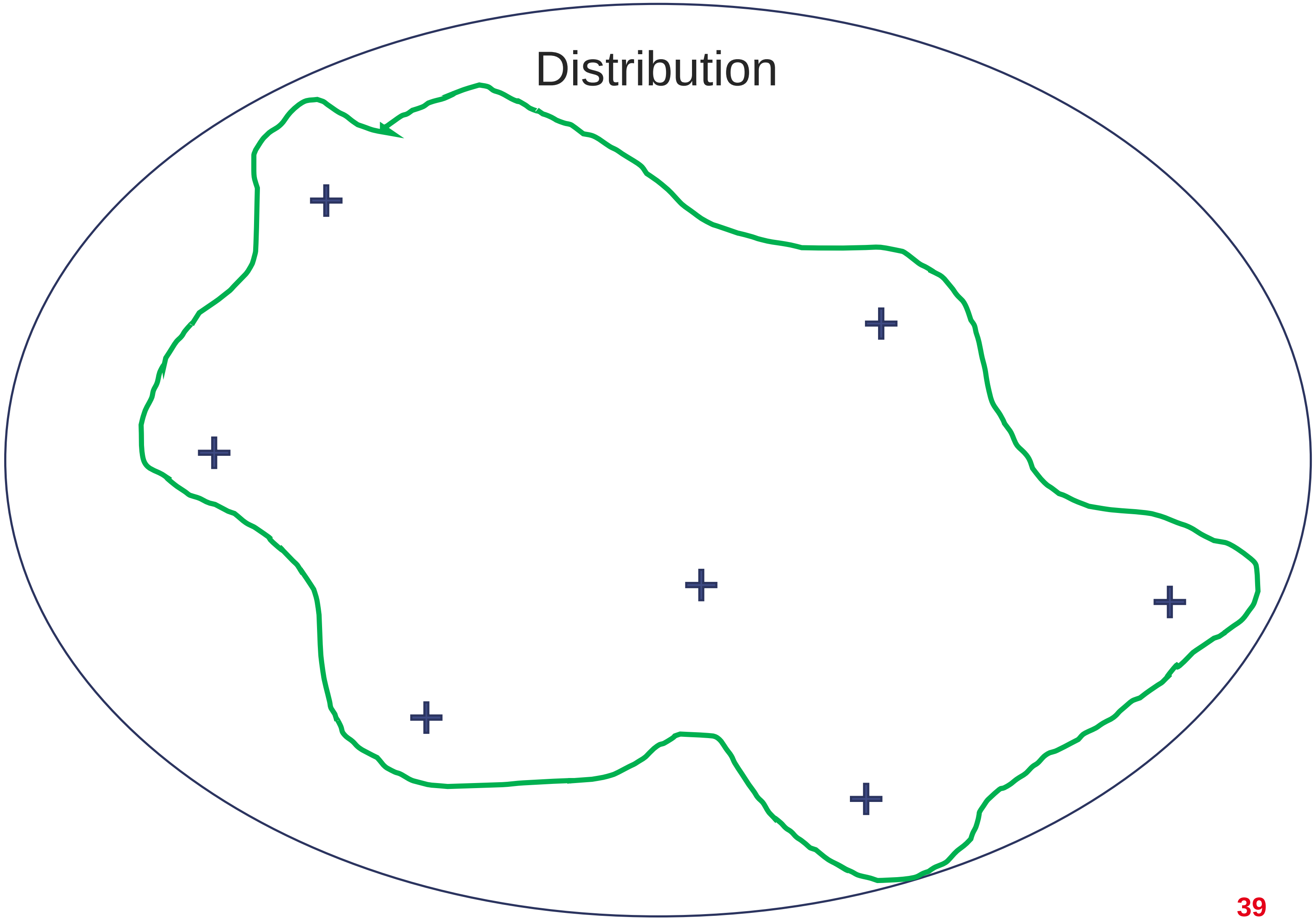
Local robustness or stability

Ideally the selected data to build and validate the model is representative of the intended distribution.



Local robustness or stability

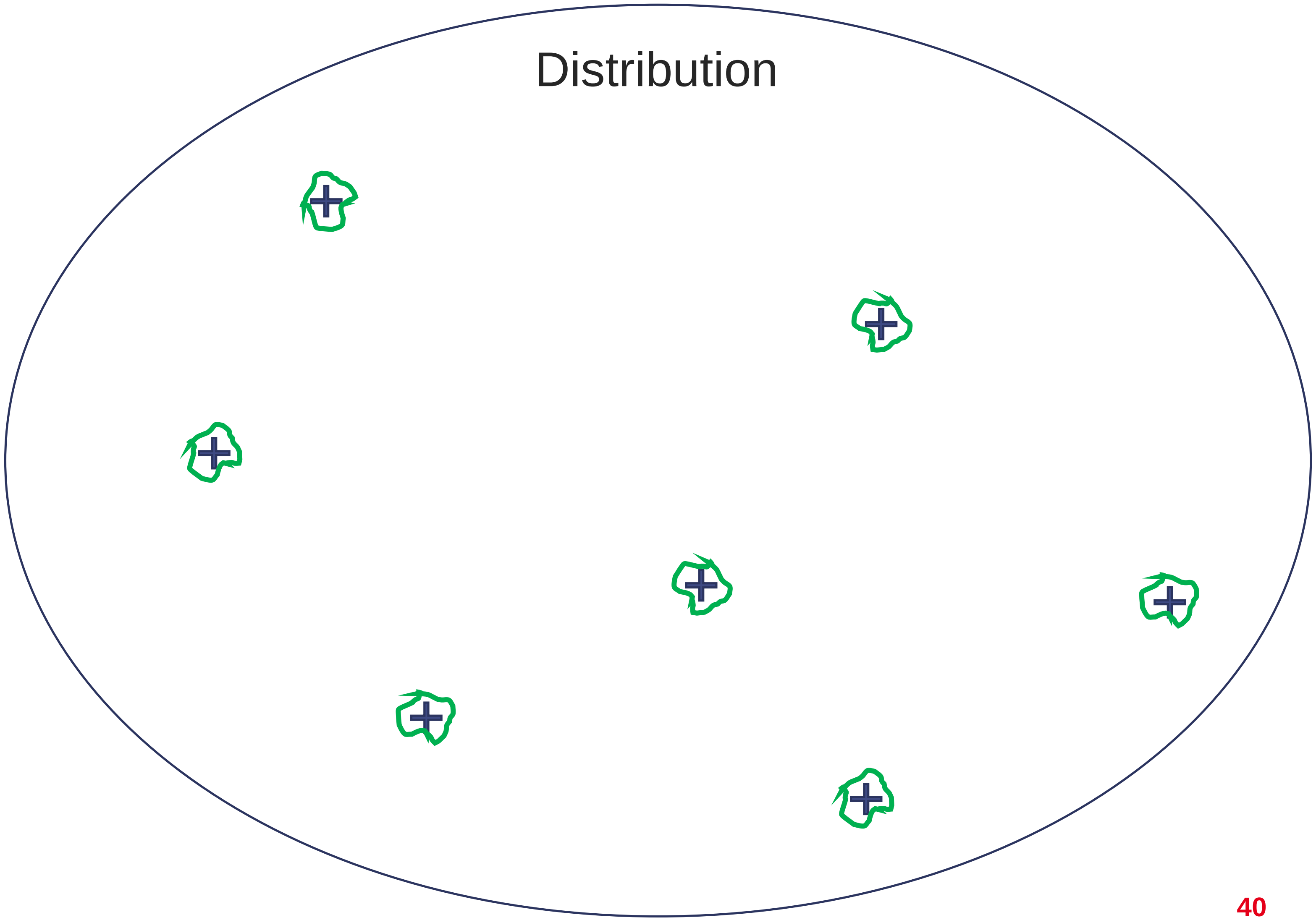
Ideally the selected data to build and validate the model is representative of the intended distribution.



Local robustness or stability

Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

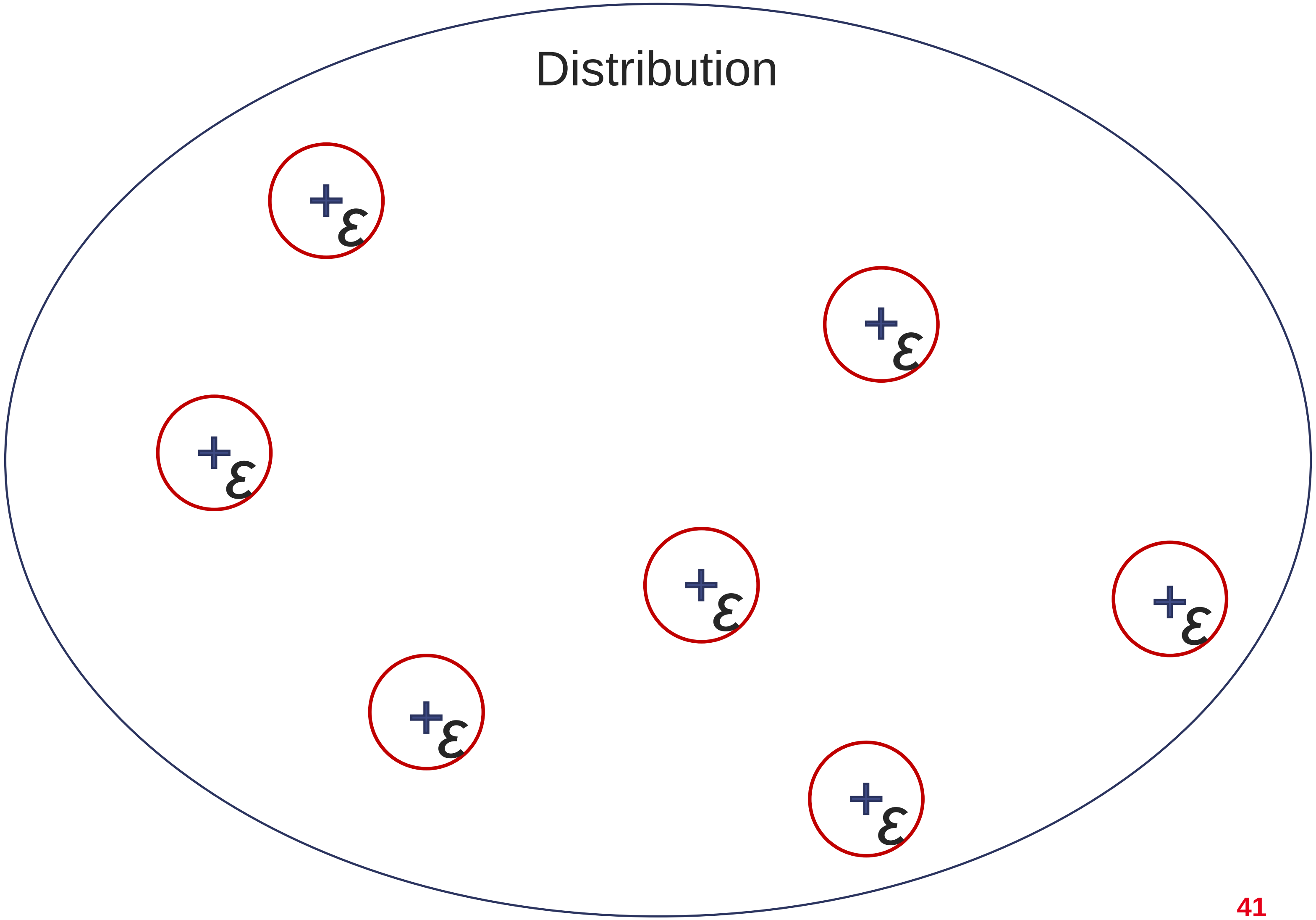


Local robustness or stability

Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?



Local robustness or stability

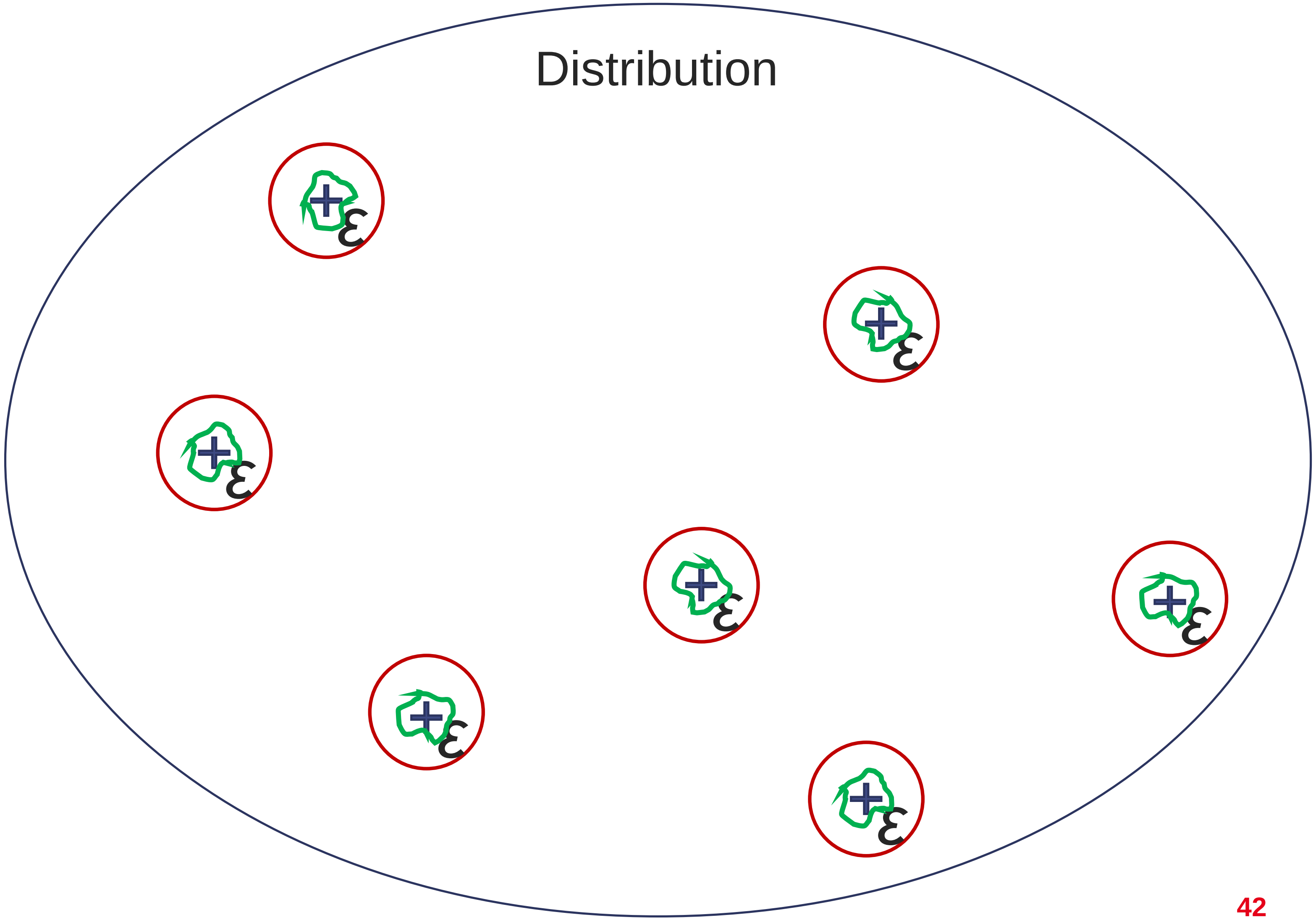
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?

What is it good for?

Can detect this…



Local robustness or stability

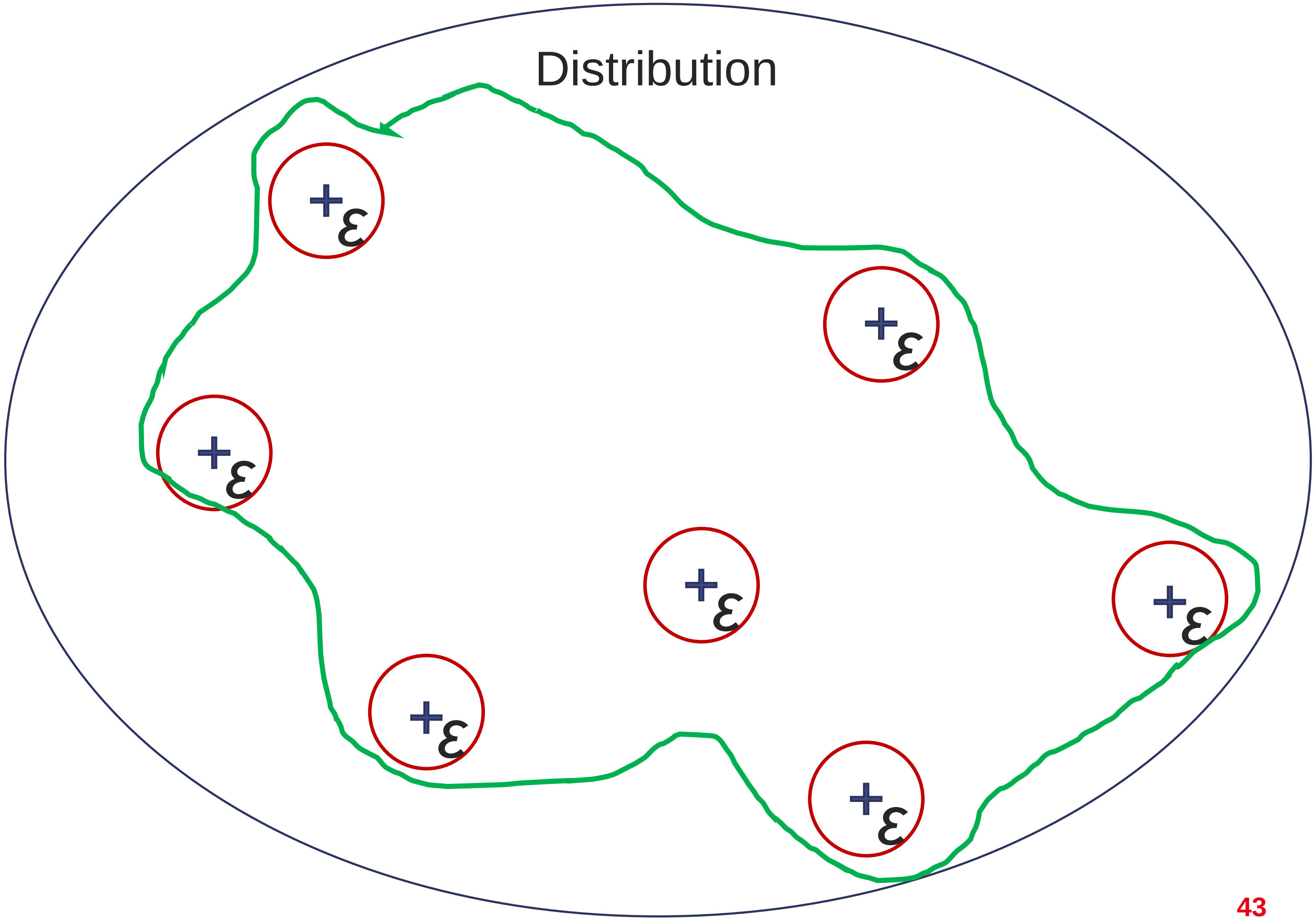
Ideally the selected data to build and validate the model is representative of the intended distribution.

What we want to avoid is a model that only knows what it was shown

How does it behave with the neighborhood of selected data?

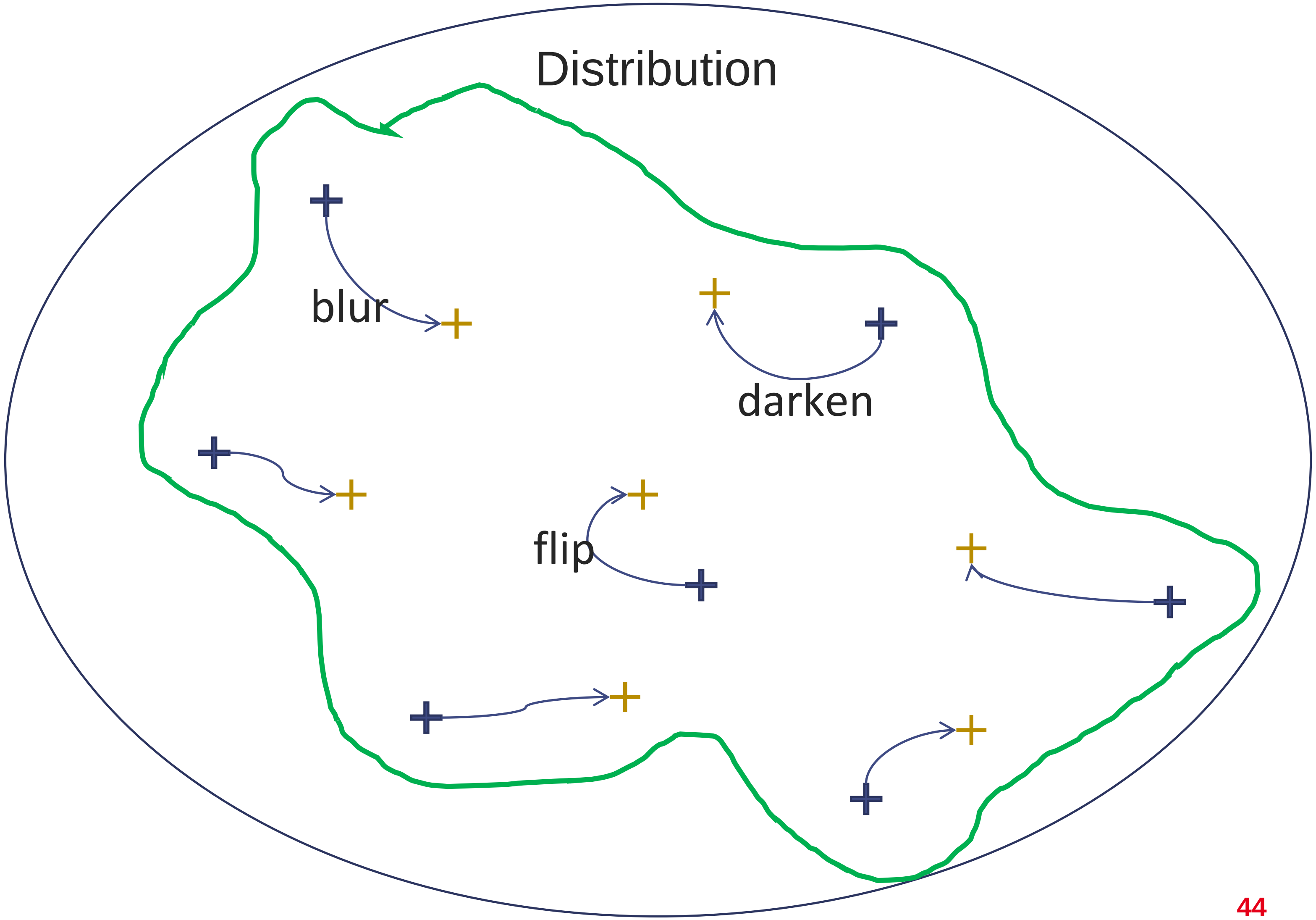
What is it good for?

… But doesn’t imply this



Local robustness or stability

Can also generate other data from acceptable transformations

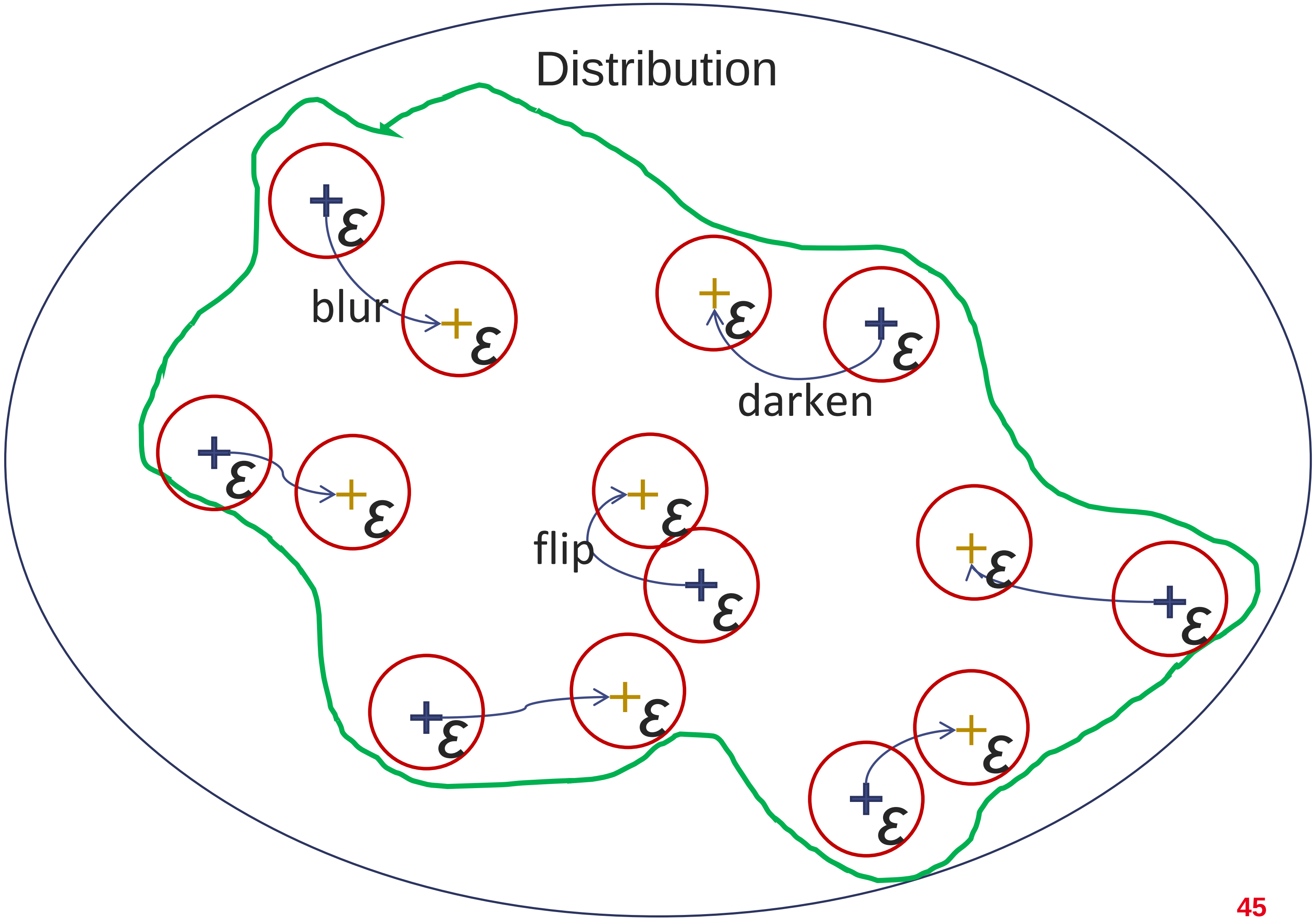


Local robustness or stability

Can also generate other data from acceptable transformations

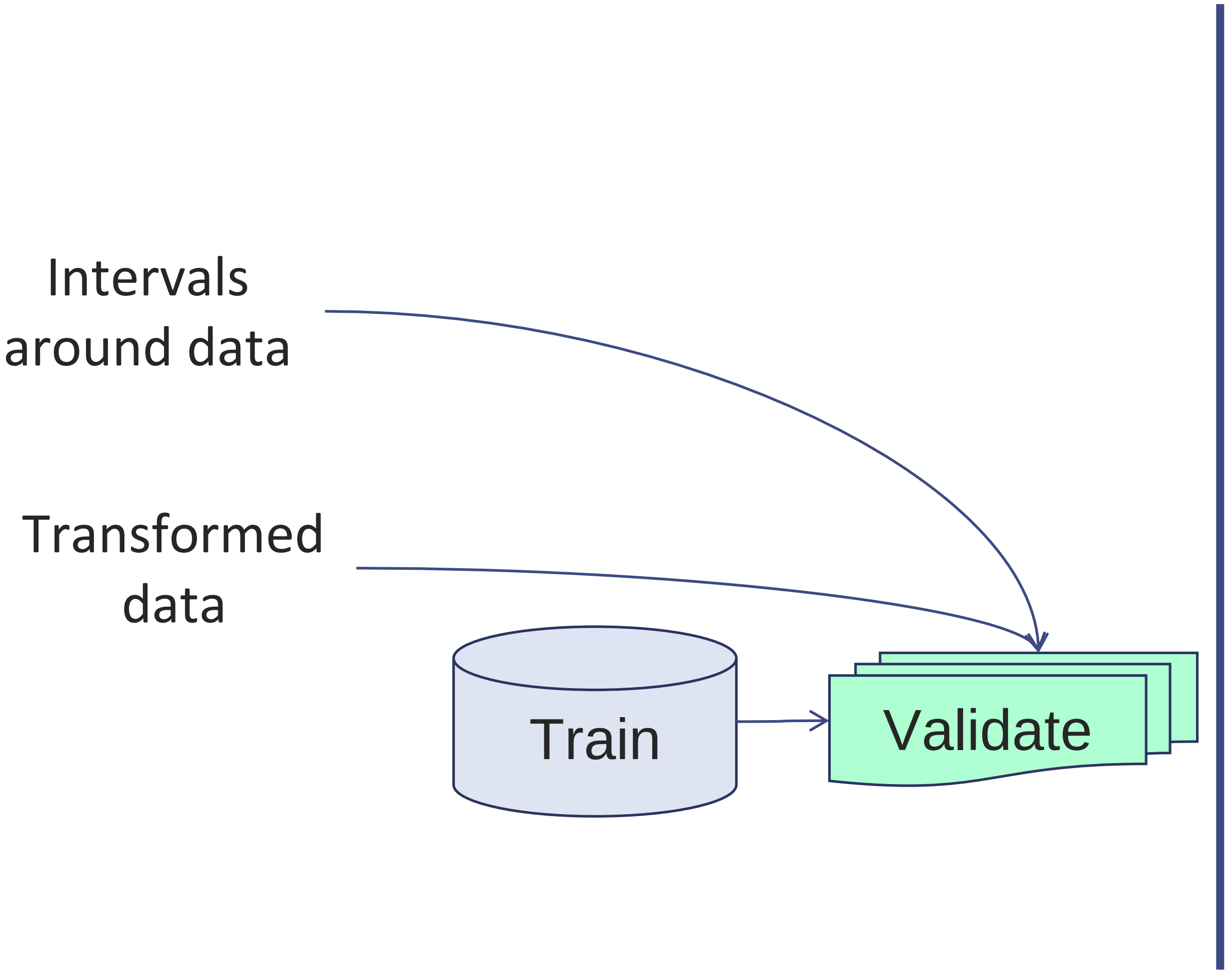
To see if the network can handle a wider distribution.

And we can combine methods

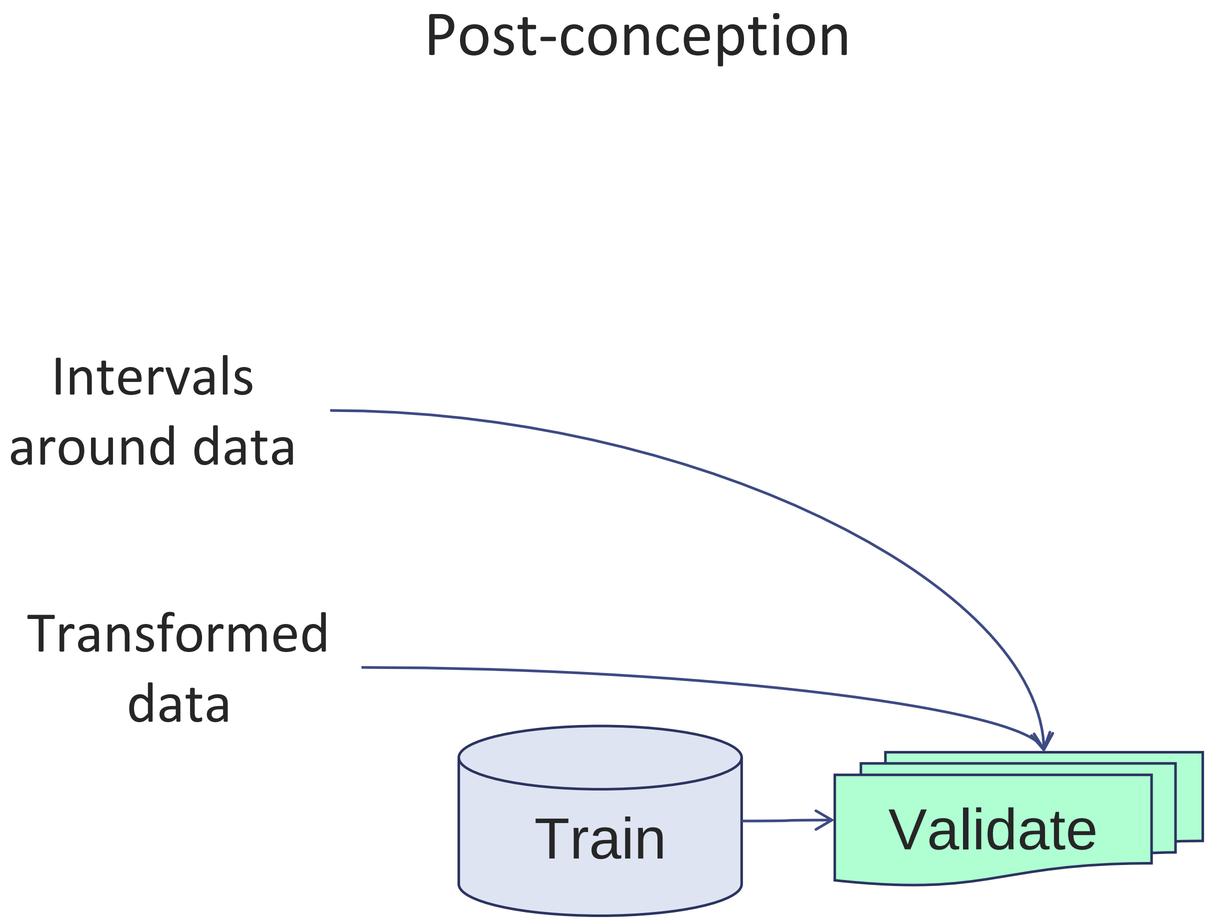


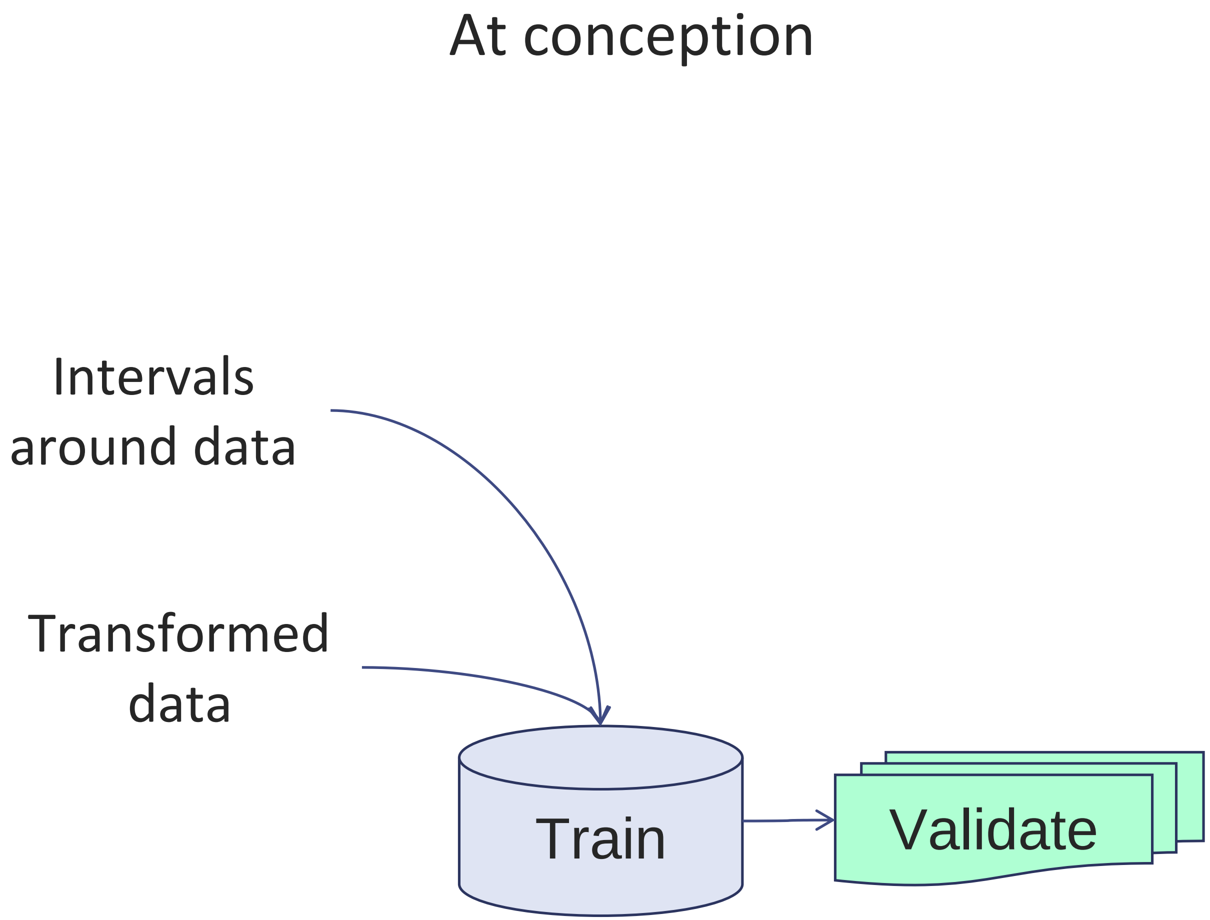
Side note: which phase do we use them ?

Post-conception



Side note: which phase do we use them ?





Rapid intro to FM

Examples for this talk:

- Property-based testing
- Abstract interpretation
- SMT Solving

Metamorphic testing

Ideal setting for testing:

- Collection of inputs
- Corresponding collection of outputs

Metamorphic testing is used when:

- You don’t know the actual **answer** (no oracle)
- But you know what **properties** should be satisfied by the inputs/outputs

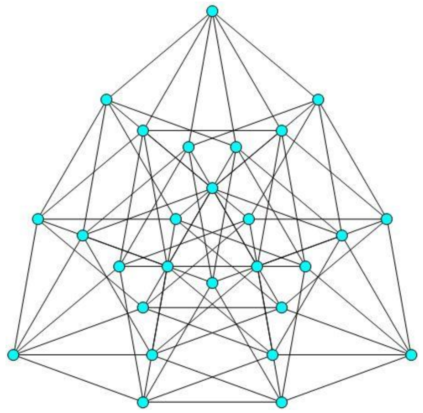
Metamorphic testing

Ideal setting for testing:

- Collection of inputs
- Corresponding collection of outputs

Metamorphic testing is used when:

- You don't know the actual **answer** (no oracle)
- But you know what **properties** should be satisfied by the inputs/outputs



Let
L (V,V) → int
be the length of the shortest path between two vertices, then :

L(a,b) = L(b,a)
For any two points a, b in a graph.

Input **symmetry**
⬍
output **equality**

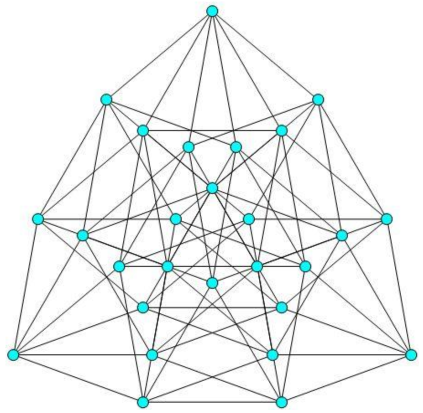
Metamorphic testing

Ideal setting for testing:

- Collection of inputs
- Corresponding collection of outputs

Metamorphic testing is used when:

- You don't know the actual **answer** (no oracle)
- But you know what **properties** should be satisfied by the inputs/outputs



Let
L (list of V) → int
be the length of the shortest path that goes through a collection of vertices, then :

L(c) ≥ L(c')
For any two collections of vertices such that c' ⊆ c.

Input **subset**
↕
output **inequality**

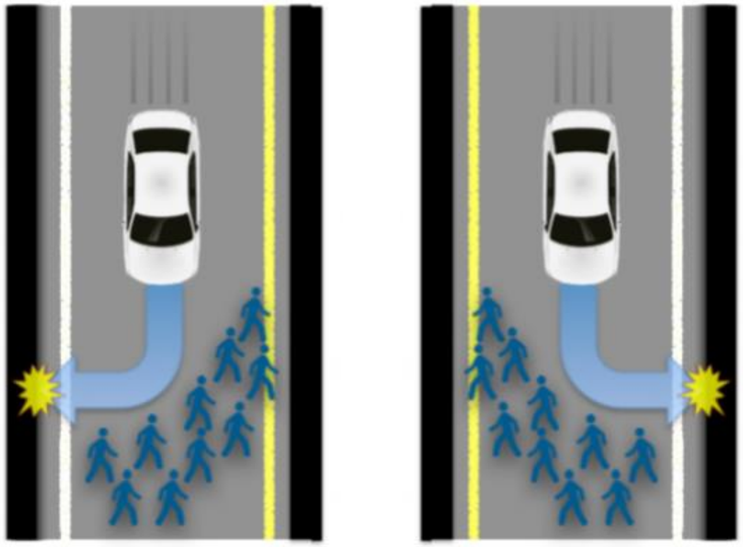
Metamorphic testing applied to AI : AIMOS

AIMOS (Artificial Intelligence Metamorphic Observing Software) is a tool to assess the stability of AI systems using metamorphic testing.

- No need to label data for testing.
- Automates the entire process of applying metamorphic properties on the inputs and outputs of models, comparing them and compiling the results into a stability score.
- Model agnostic (Neural Networks, Support Vector Machines, *etc*.).

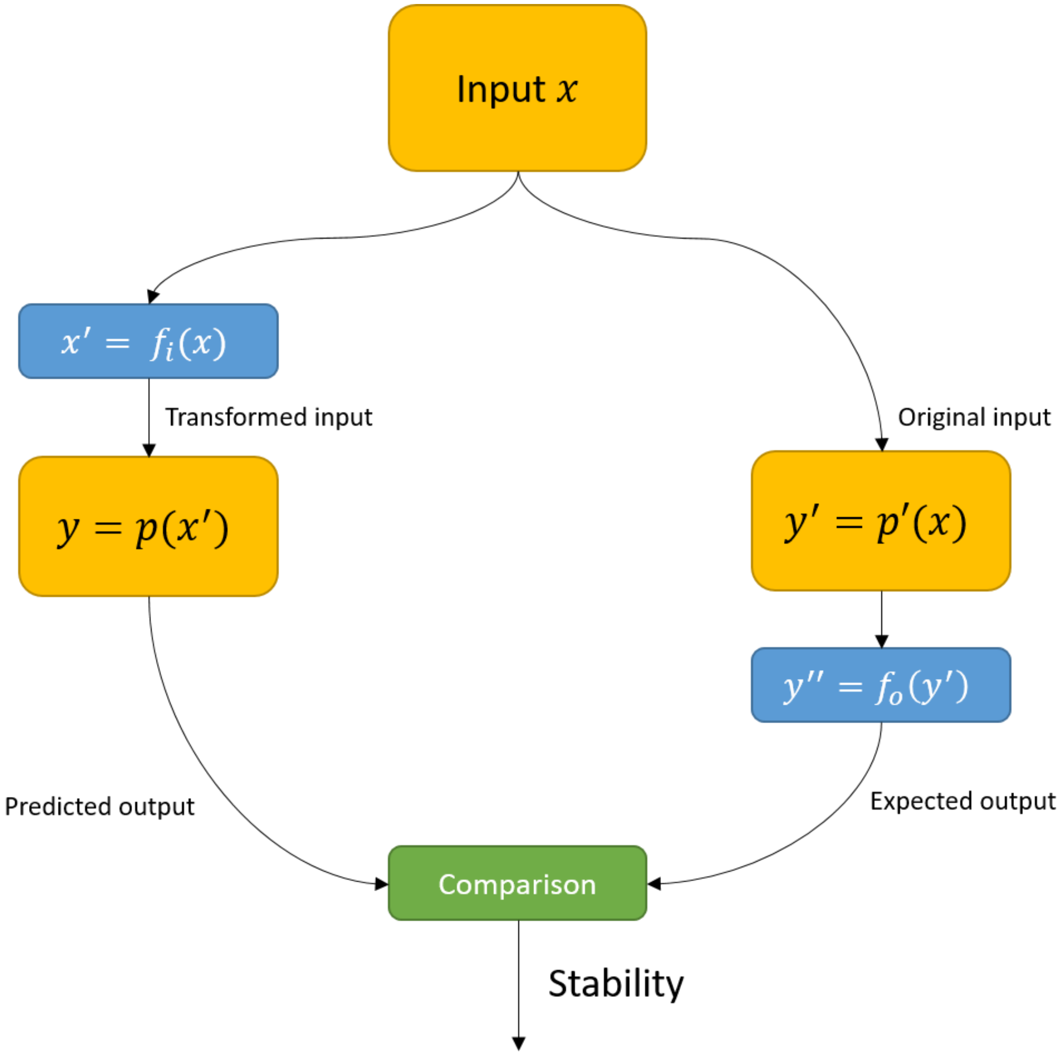
Metamorphic testing applied to AI : AIMOS

Metamorphic testing applied to AI : AIMOS

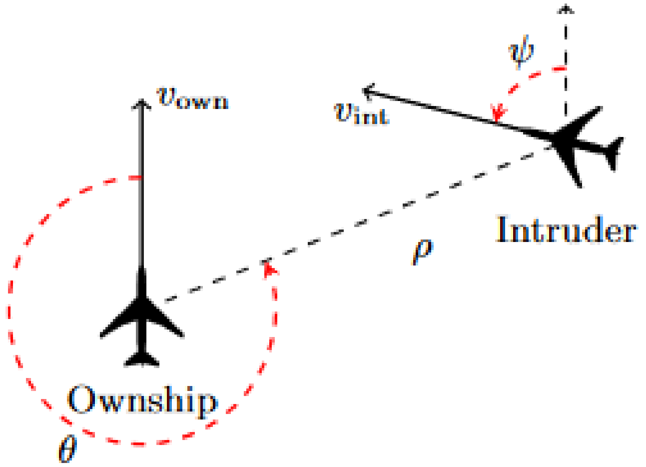


caisar-platform.com

Metamorphic testing applied to AI : AIMOS



Metamorphic testing applied to AI : AIMOS

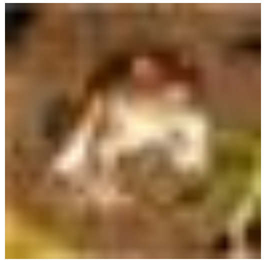


Right Left

Previous advisory	Stability	Stability on restricted space
Clear-of-conflict	97.4%	89.7%
Left	97.6%	95.9%
Right	97.5%	99.6%
Strong left	97.5%	95.3%
Strong right	97.8%	99.8%

Metamorphic testing applied to AI : AIMOS
Easy to use

- Written in Python
- Model agnostic: only the inference functions are needed.
- Built-in support for various frameworks, input formats and model types.
- Built-in classical transformations (rotation, noise, symmetry, *etc.*).



Metamorphic testing applied to AI : AIMOS
Easy to use

- With a configuration file

```yaml
options:
  plot: True
  inputs_path: "inputs"
  transformations:
    - name: "gaussian_blur"
      fn_range: range(1, 10, 2)

models:
  - defaults:
      models_path: "models/model.onnx"
```

Metamorphic testing applied to AI : AIMOS
Easy to use

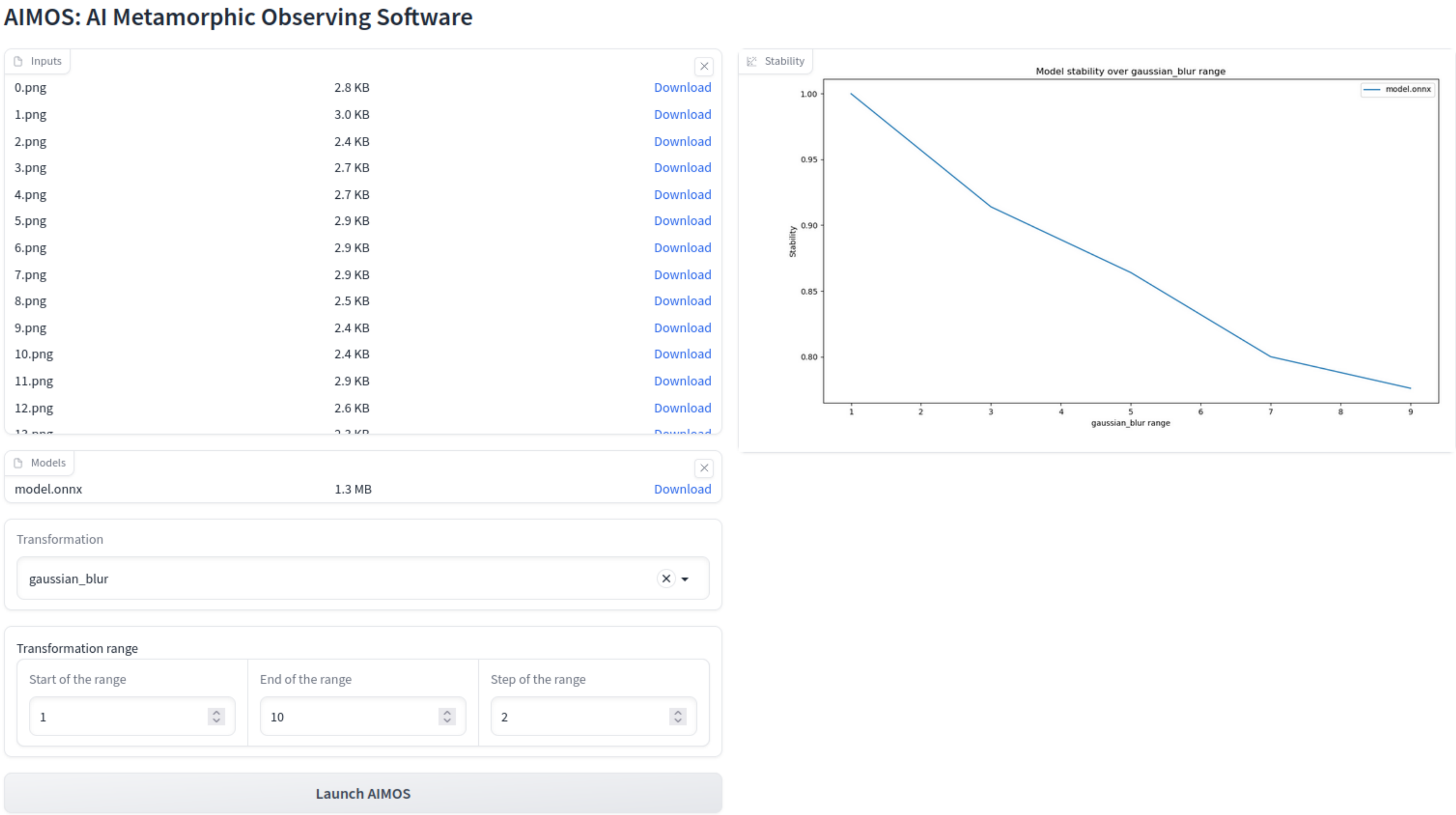
- With a configuration file
- As a Python library

```python
from aimos import core

core.main(
    "./inputs",
    "./models/model.onnx",
    "average_blur",
    fn_range=range(1, 10, 2),
    plot=True,
)
```

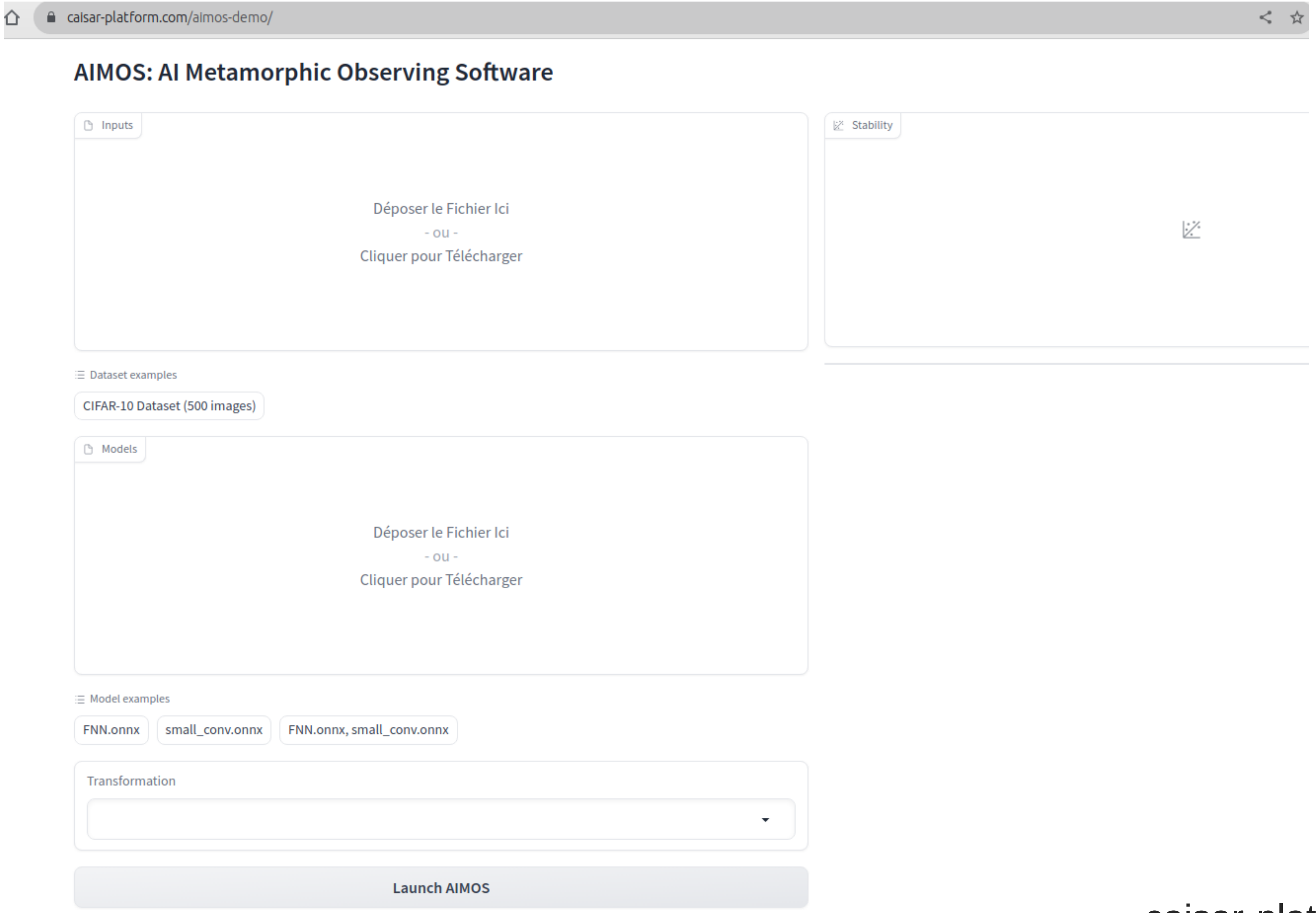
Metamorphic testing applied to AI : AIMOS
Easy to use

- With a configuration file
- As a Python library
- With a Graphical User Interface



caisar-platform.com

Metamorphic testing applied to AI : AIMOS
Easy to use



caisar-platform.com

Metamorphic testing applied to AI : AIMOS
Modular and extensible

Any operation can be replaced with a custom made Python function (loading the model, the inputs, new metrics, *etc.*).

```python
def dead_columns(input, columns=np.uint8([50, 100, 150])):
    """ Adds dead pixel columns to an image. """
    input[:, columns, :] = 0
    return input
```

Metamorphic testing applied to AI : AIMOS

AIMOS is a tool that can be integrated in the verification and validation process of AI-based components.

- Freely available for teaching and research purposes.

- Integrated in CAISAR, an open-source platform for characterizing safety in AI systems.

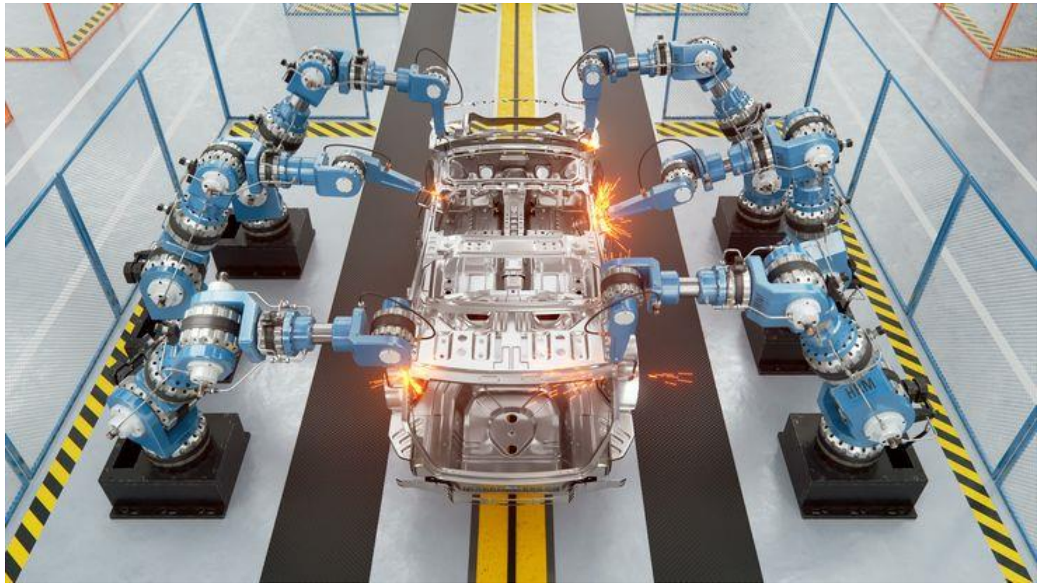
CAISAR

caisar-platform.com

Metamorphic testing applied to AI : AIMOS

The use-case

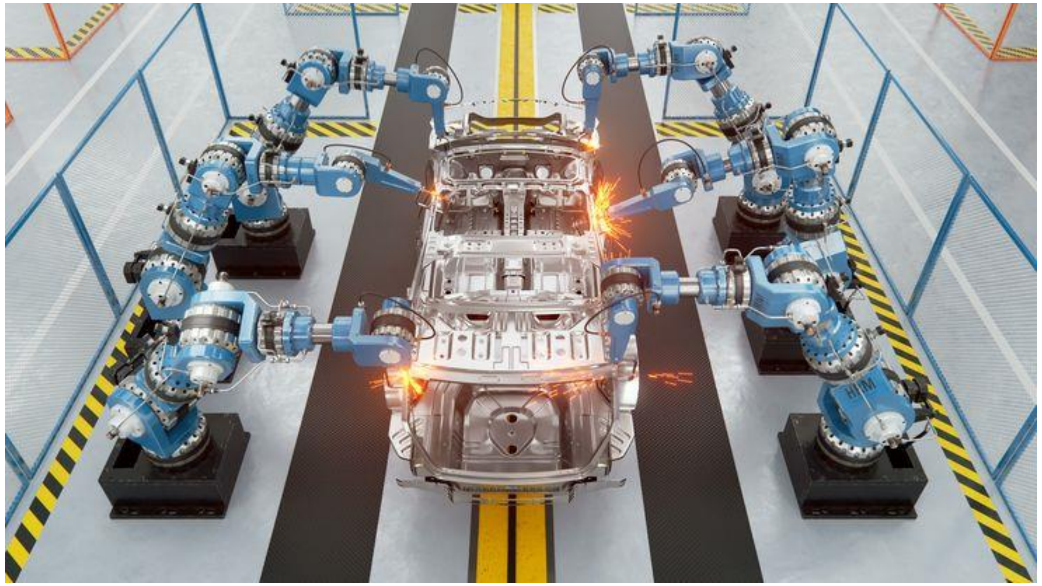
- Welding conveyor belt
- AI analysis for detection of faulty welds
- Notification of human expert



Metamorphic testing applied to AI : AIMOS

The use-case

- Welding conveyor belt
- AI analysis for detection of faulty welds
- Notification of human expert



Metamorphic testing applied to AI : AIMOS

The use-case

- Welding conveyor belt
- AI analysis for detection of faulty welds
- Notification of human expert

- 3 different production lines called C10, C20 and C34 and their corresponding weld.
- 5 AutoML models and 1 internal R&D composit model (NN+SVM) per production line.

Lemesle, A., Varasse, A., Chihani, Z., Tachet, D. (2023). **AIMOS: Metamorphic Testing of AI - An Industrial Application**. In: Guiochet, J., Tonetta, S., Schoitsch, E., Roy, M., Bitsch, F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. SAFECOMP 2023. Lecture Notes in Computer Science, vol 14182. Springer, Cham. https://doi.org/10.1007/978-3-031-40953-0_27

Metamorphic testing applied to AI : AIMOS

The use-case

- Welding conveyor belt
- AI analysis for detection of faulty welds
- Notification of human expert

- 3 different production lines called C10, C20 and C34 and their corresponding weld.
- 5 AutoML models and 1 internal R&D composit model (NN+SVM) per production line.

The environment => ODD => properties

- Day light changes + human workers pass by light sources => Robustness to **varying brigthness**
- Vibrating environment => Robustness to **blurring**

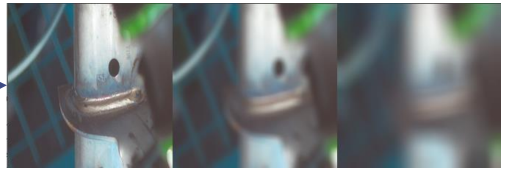
Lemesle, A., Varasse, A., Chihani, Z., Tachet, D. (2023). **AIMOS: Metamorphic Testing of AI - An Industrial Application**. In: Guiochet, J., Tonetta, S., Schoitsch, E., Roy, M., Bitsch, F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. SAFECOMP 2023. Lecture Notes in Computer Science, vol 14182. Springer, Cham. https://doi.org/10.1007/978-3-031-40953-0_27

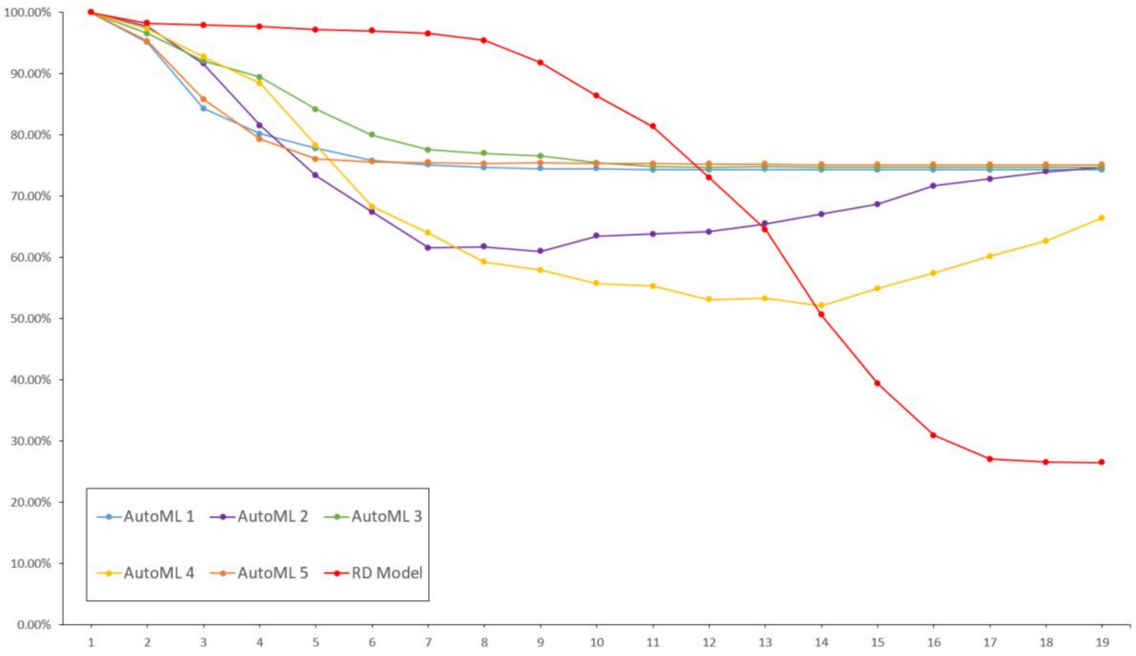
Metamorphic testing applied to AI : AIMOS

Metamorphic properties

AI Models

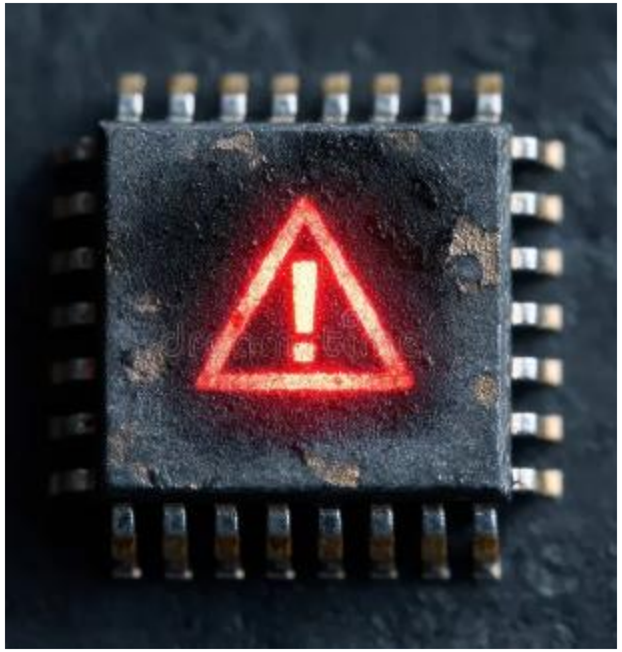
Representative dataset





Side note: think about the use-case

and careful with transfer learning

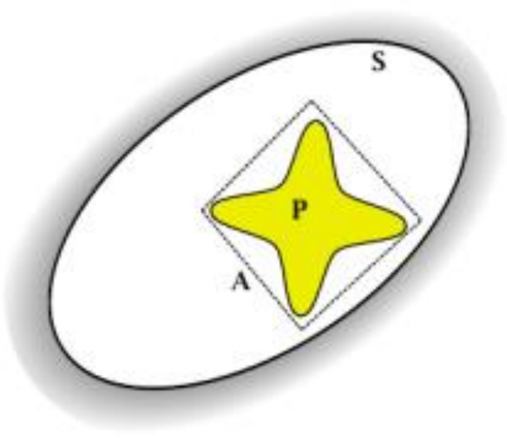


Rapid intro to FM

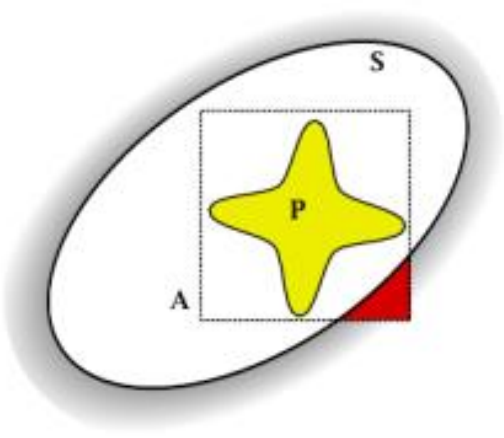
Examples for this talk:

- Property-based testing
- Abstract interpretation
- SMT Solving

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

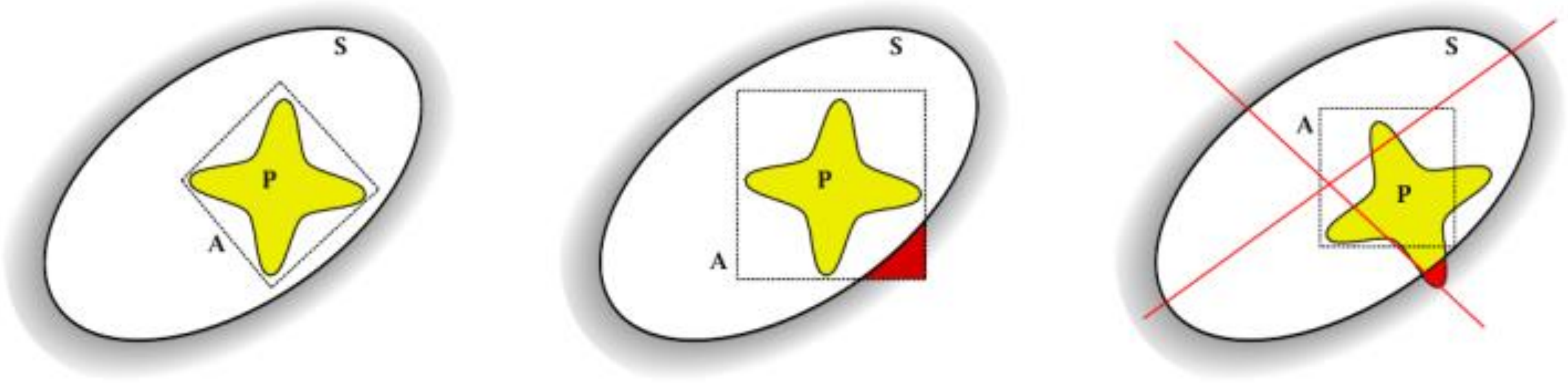


precise analysis
$A \subseteq S \implies P \subseteq S$



false alarm
$A \not\subseteq S$ but $P \subseteq S$

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation



precise analysis	false alarm	unsound analysis
$A \subseteq S \implies P \subseteq S$	$A \not\subseteq S$ but $P \subseteq S$	$A \subseteq S$ but $P \not\subseteq S$

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

We would like to verify a property on the all possible values of inputs x ∈ [a,b] and y ∈ [c,d] in some program.

e.g.:

$$x + y \in [a+c, b+d]$$

Do the same for all operations in the program.

Use other types of domain for more precision (not just intervals).

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

Property: "f(y)=100 → Critical vibration frequency "

```
f (int y){
 int x;                    ..................
 x = 3 * (y²+1);           ..........
 if x > 100 then           .........
    x = x + 10;            ..........
 else                      ...................
    x = x - 2;             ...........
 return x;                 ..............
}
```

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

Property: “f(y)=100 → Critical vibration frequency ”

Concret

f (int y){	
int x;	..-2,-1,0,1,2,..
x = 3 * (y^2+1);	3,6,9,..
if x > 100 then	102,105,108,..
x = x + 10;	**112,115,118**,..
else	3,6,9..93,96,99
x = x - 2;	**1,4,7,..91,94,97**
return x;	1,4,..94,97,112,115..
}	

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

Property: "f(y)=100 → Critical vibration frequency "

Code		Concret	Intervals
f (int y){			
int x;	....	..-2,-1,0,1,2,..	-∞,+∞
x = 3 * (y²+1);	....	3,6,9,..	3,+∞
if x > 100 then	....	102,105,108,..	102,+∞
x = x + 10;	....	**112,115,118**,..	**112,+∞**
else	....	3,6,9..93,96,99	3,99
x = x - 2;	....	**1,4,7,..91,94,97**	**1,97**
return x;	....	1,4,..94,97,112,115..	1,+∞
}			

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

Property: "f(y)=100 → Critical vibration frequency "

	Concret	Intervals	modulo
f (int y){			
int x;	..-2,-1,0,1,2,..	-∞,+∞	0%1
x = 3 * (y^2+1);	3,6,9,..	3,+∞	0%3
if x > 100 then	102,105,108,..	102,+∞	0%3
x = x + 10;	**112,115,118**,..	**112,+∞**	1%3
else	3,6,9..93,96,99	3,99	0%3
x = x - 2;	**1,4,7,..91,94,97**	**1,97**	1%3
return x;	1,4,..94,97,112,115..	1,+∞	1%3
}			

PyRAT: Python Reachability Assessment Tool Based on Abstract Interpretation

<u>Property: “f(y)=100 → Critical vibration frequency ”</u>

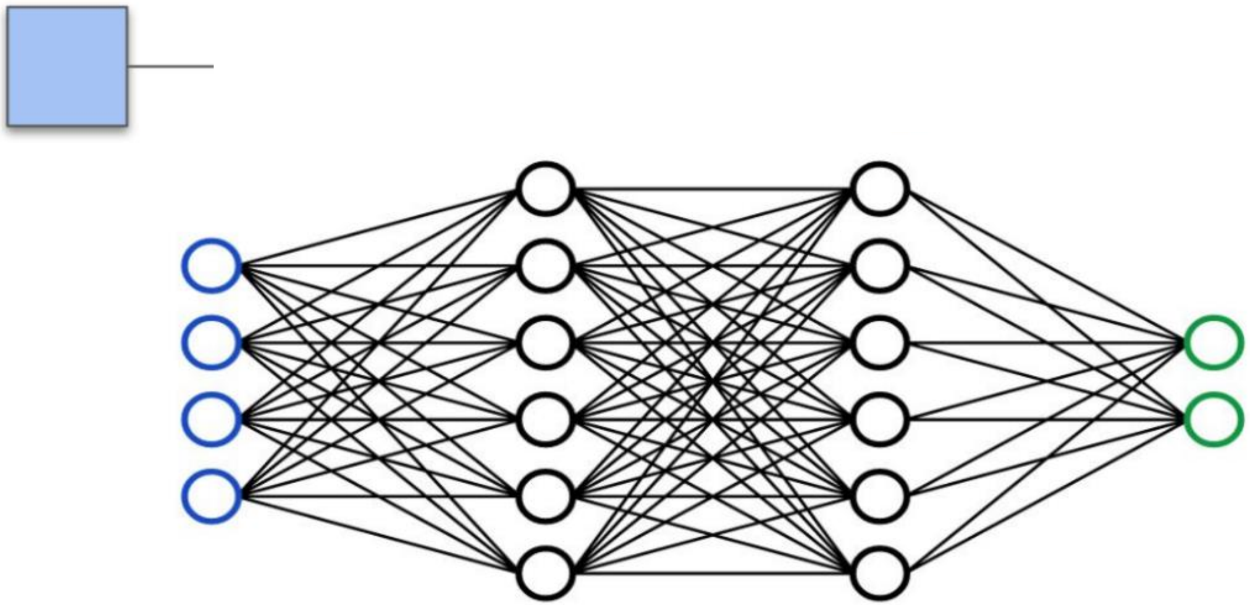
	Concret	Intervals	modulo	Union of intervals
f (int y){				
int x;	..-2,-1,0,1,2,..	-∞,+∞	0%1	-∞,+∞
x = 3 * (y^2+1);	3,6,9,..	3,+∞	0%3	3,+∞
if x > 100 then	102,105,108,..	102,+∞	0%3	102,+∞
x = x + 10;	**112,115,118**,..	**112,+∞**	1%3	**112,+∞**
else	3,6,9..93,96,99	3,99	0%3	3,99
x = x - 2;	**1,4,7,..91,94,97**	**1,97**	1%3	**1,97**
return x;	1,4,..94,97,112,115..	1,+∞	1%3	*[1,97]∪ [112,+∞[*
}				

Conservative over-approximation: the concretization of the abstract domains contains reality

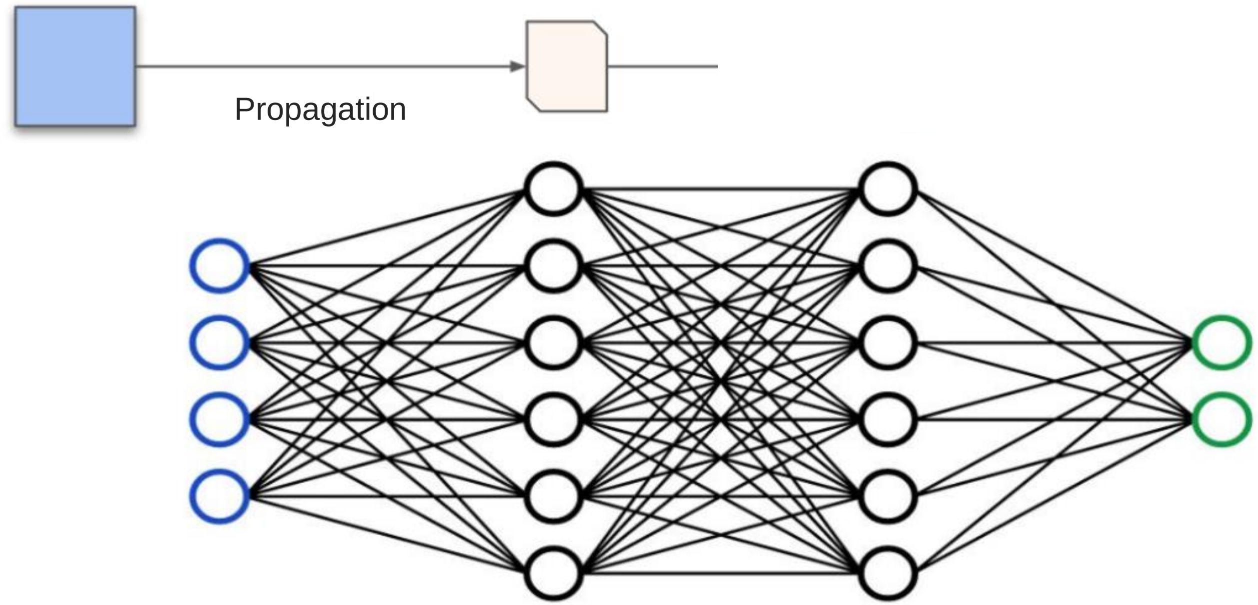
The inverse is not necessarily true. 1,4..94,97,**100,103,106,109**,112,115..

PyRAT: Python Reachability Assessment Tool
Application to NN

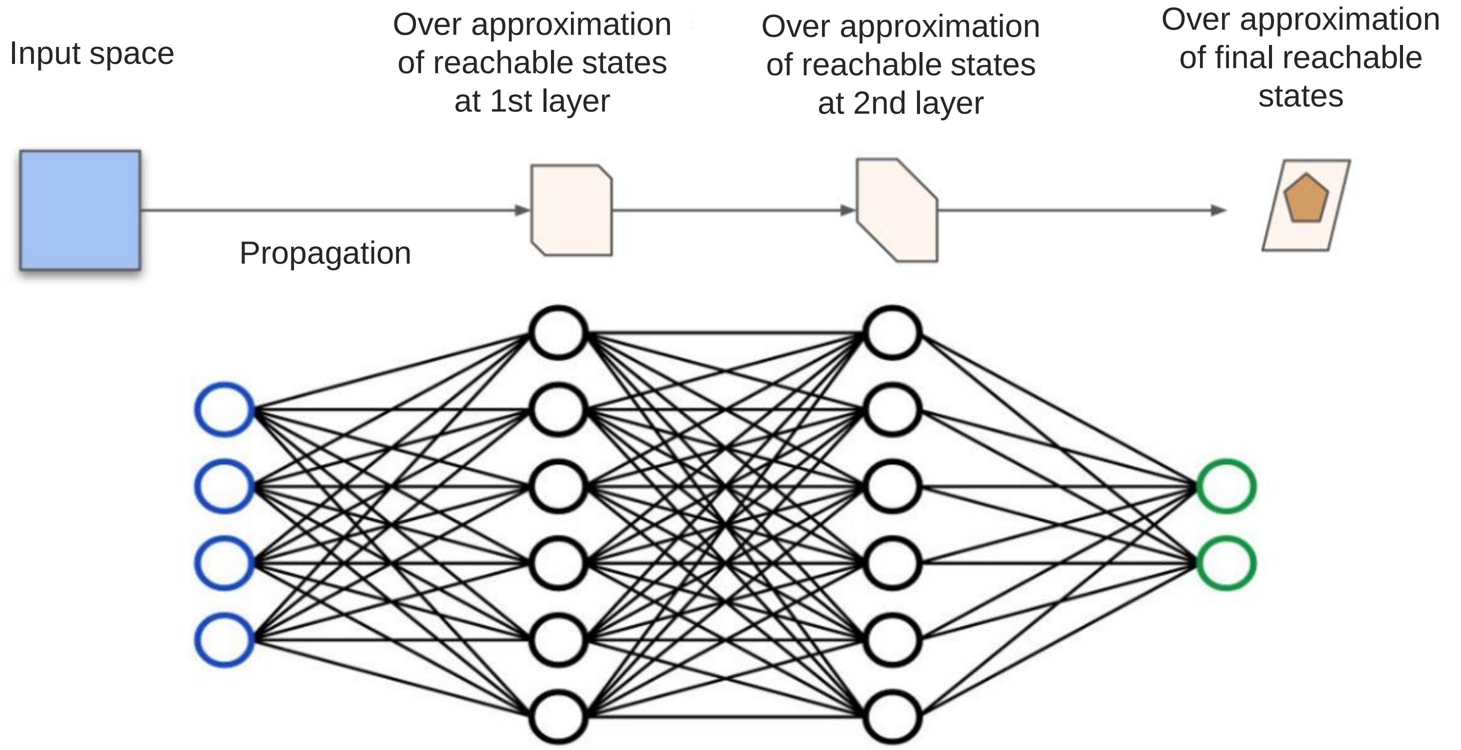
Input space



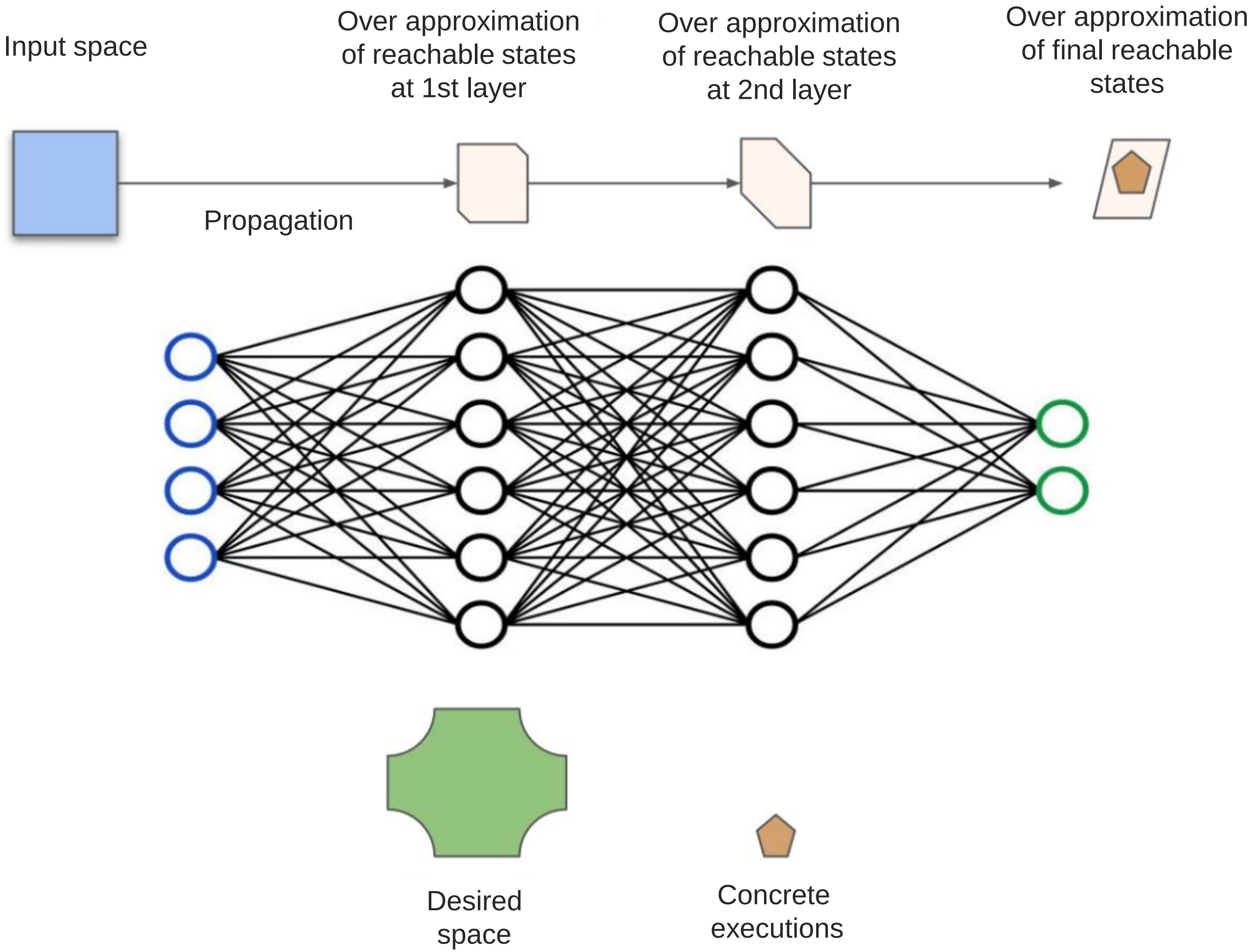
PyRAT: Python Reachability Assessment Tool
Application to NN

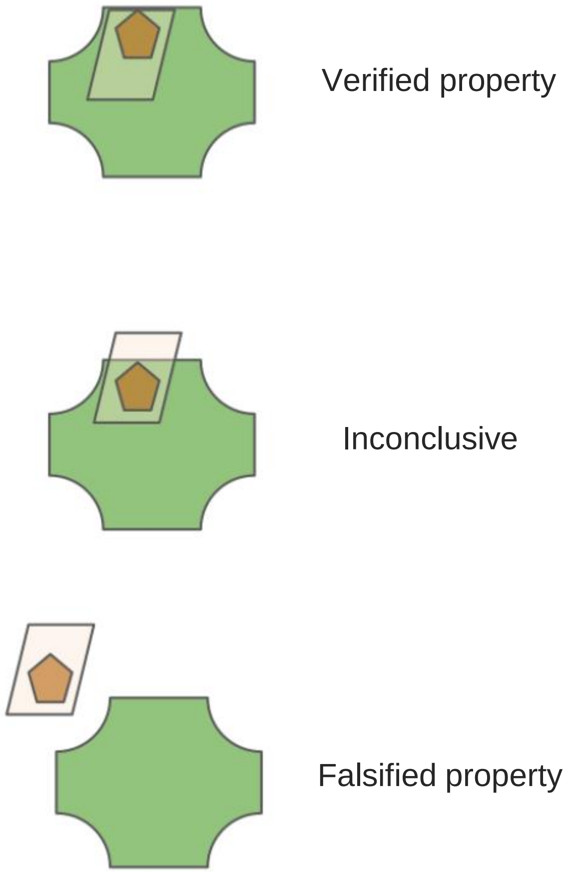


PyRAT: Python Reachability Assessment Tool
Application to NN

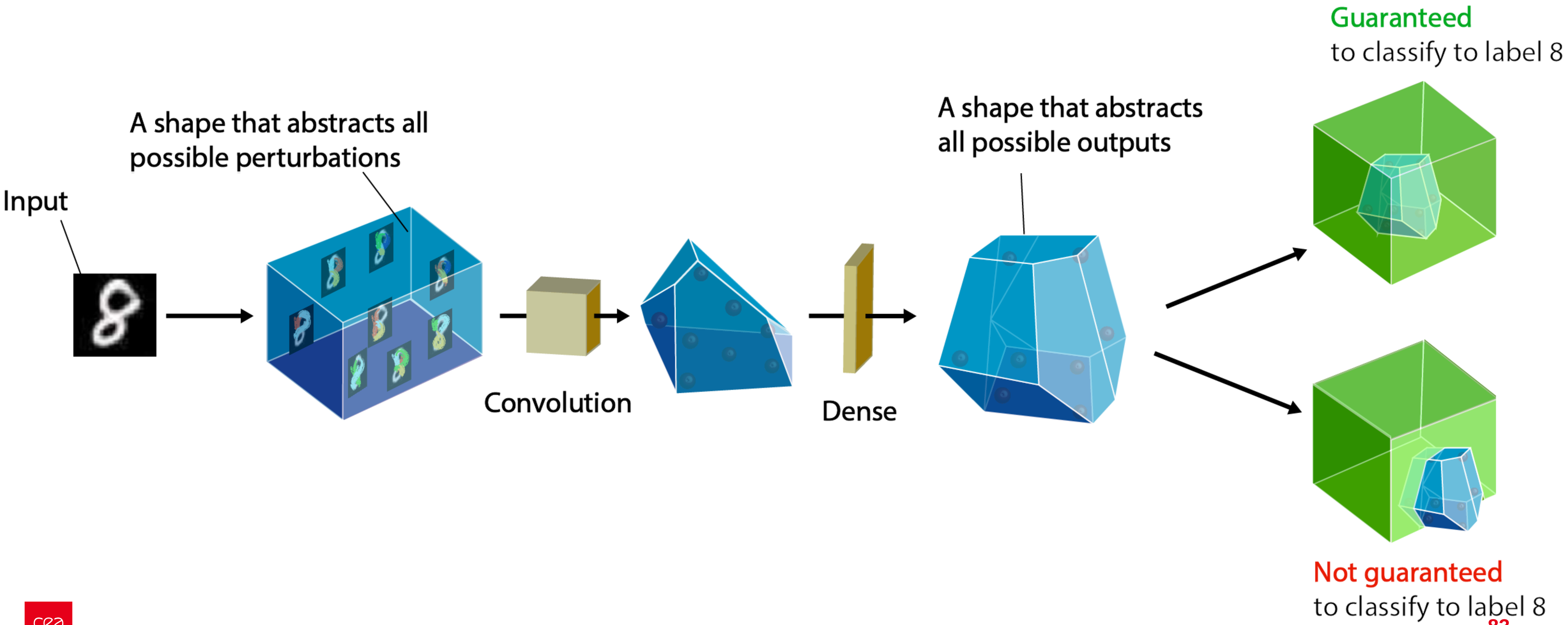


PyRAT: Python Reachability Assessment Tool
Application to NN





PyRAT: Python Reachability Assessment Tool
Application to NN



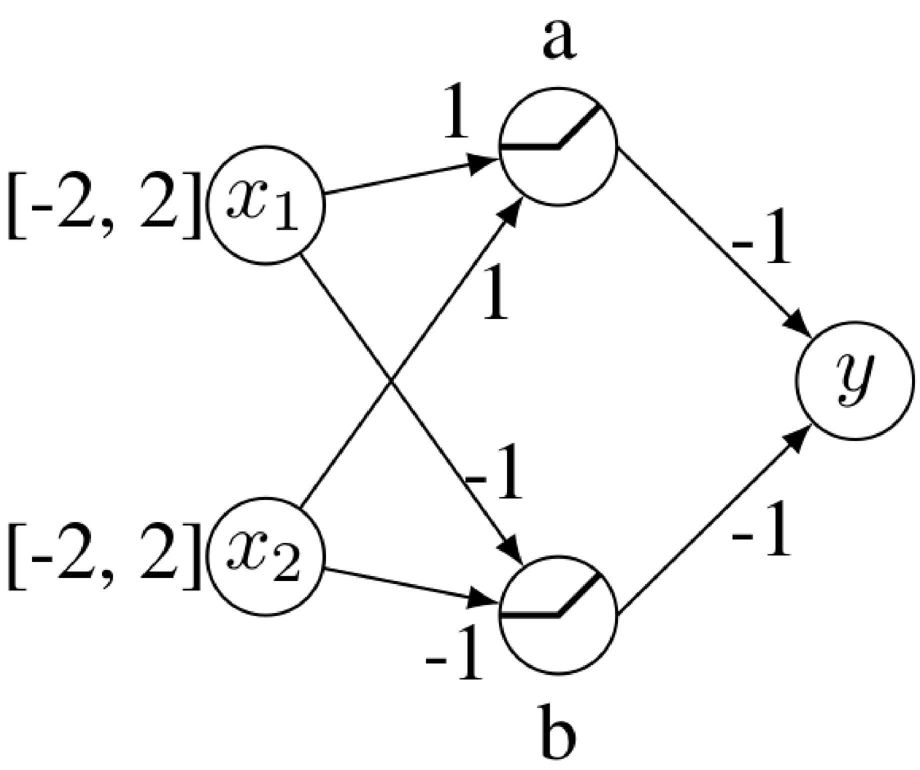
PyRAT: Python Reachability Assessment Tool
Application to NN

$$-2 \leq x_1 \leq 2$$
$$-2 \leq x_2 \leq 2$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y = -a_{\text{out}} - b_{\text{out}}$$



$$\boxed{\textbf{Prove that } y > -5}$$

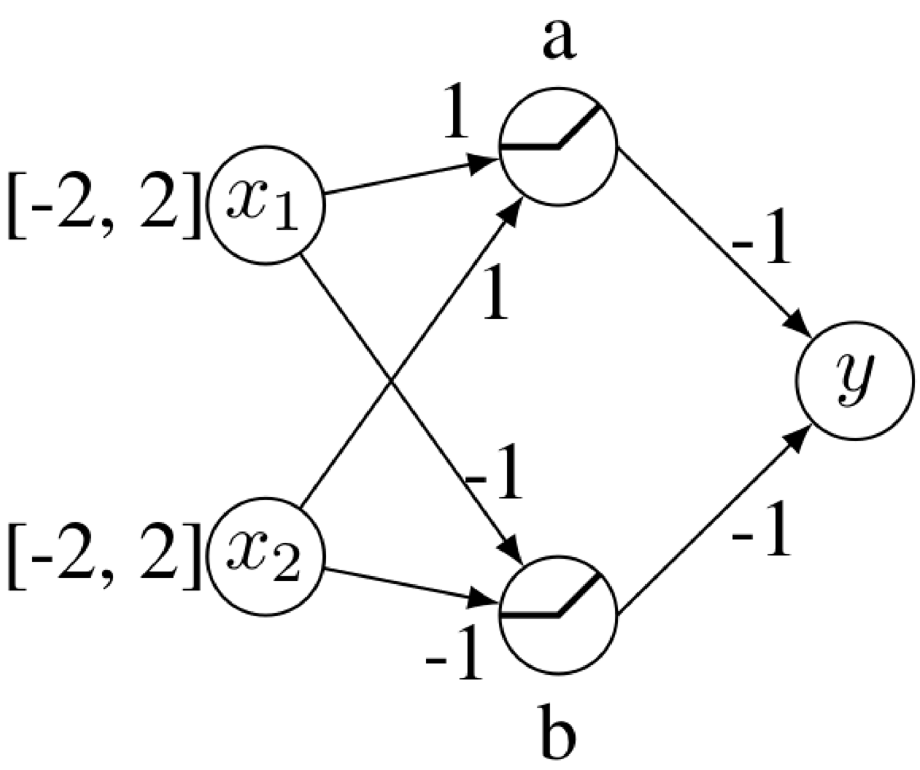
PyRAT: Python Reachability Assessment Tool
Application to NN

$$-2 \leq x_1 \leq 2$$
$$-2 \leq x_2 \leq 2$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y = -a_{\text{out}} - b_{\text{out}}$$



$a_{\text{in}} =$ [-2,2] + [-2,2] = [-4,4]

$b_{\text{in}} =$ - [-2,2] - [-2,2] =[-4,4]

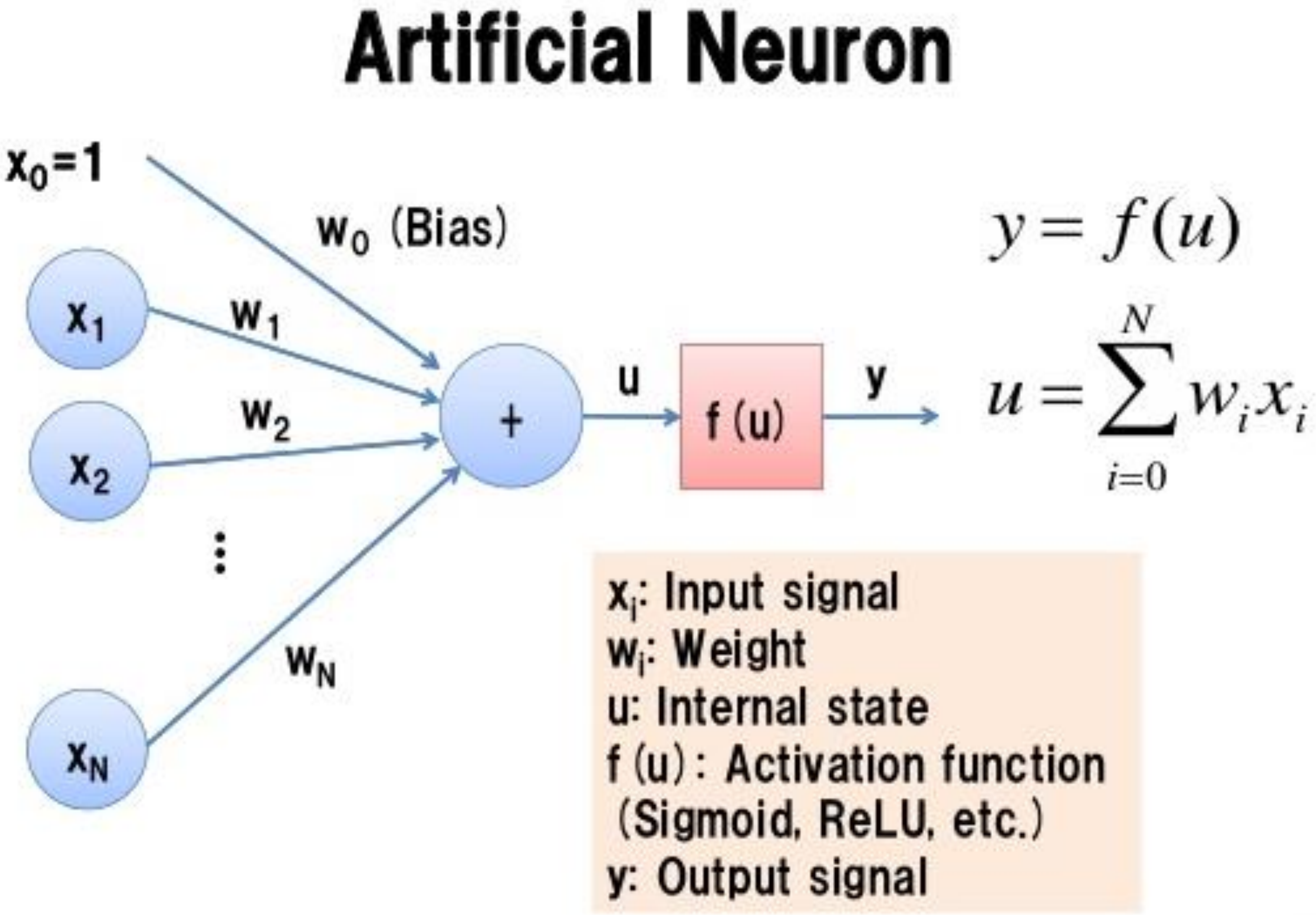
$a_{\text{out}} =$

$b_{\text{out}} =$

$y =$

Prove that $y > -5$

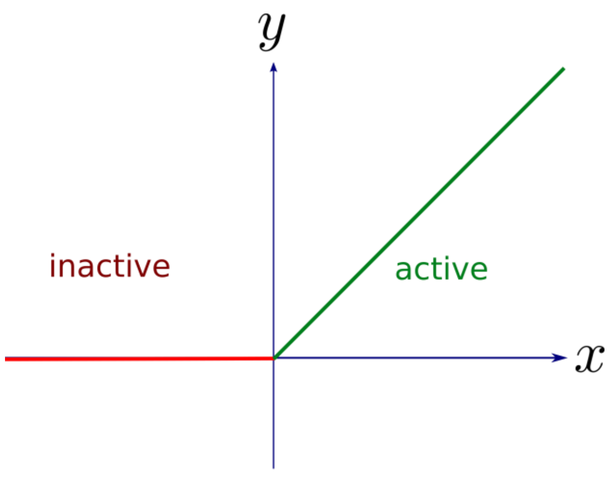
PyRAT: Python Reachability Assessment Tool
Application to NN



PyRAT: Python Reachability Assessment Tool
Application to NN

$$a_{\text{out}} = \max(a_{\text{in}}, 0)$$

	a_{in}	a_{out}
$a_{\text{in}} = x_1 + x_2$	[-4,4]	
if a_{in} > 0 then	]0,4]	
$a_{\text{out}} = a_{\text{in}}$		]0,4]
else	[-4,0]	
$a_{\text{out}} = 0$		[0,0]
		[0,4]



$ReLU : x \to max(x, 0)$

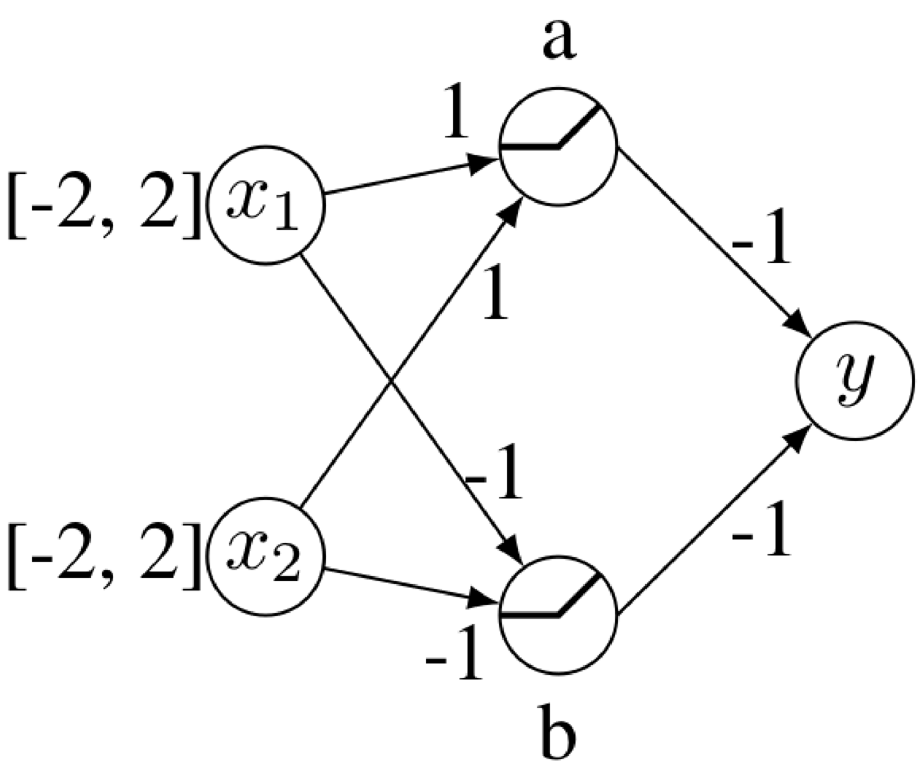
PyRAT: Python Reachability Assessment Tool
Application to NN

$$-2 \leq x_1 \leq 2$$
$$-2 \leq x_2 \leq 2$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y = -a_{\text{out}} - b_{\text{out}}$$



$$a_{\text{in}} = \text{[-2,2] + [-2,2] = [-4,4]}$$

$$b_{\text{in}} = \text{- [-2,2] - [-2,2] =[-4,4]}$$

$$a_{\text{out}} =$$

$$b_{\text{out}} =$$

$$y =$$

$\boxed{\textbf{Prove that } y > -5}$

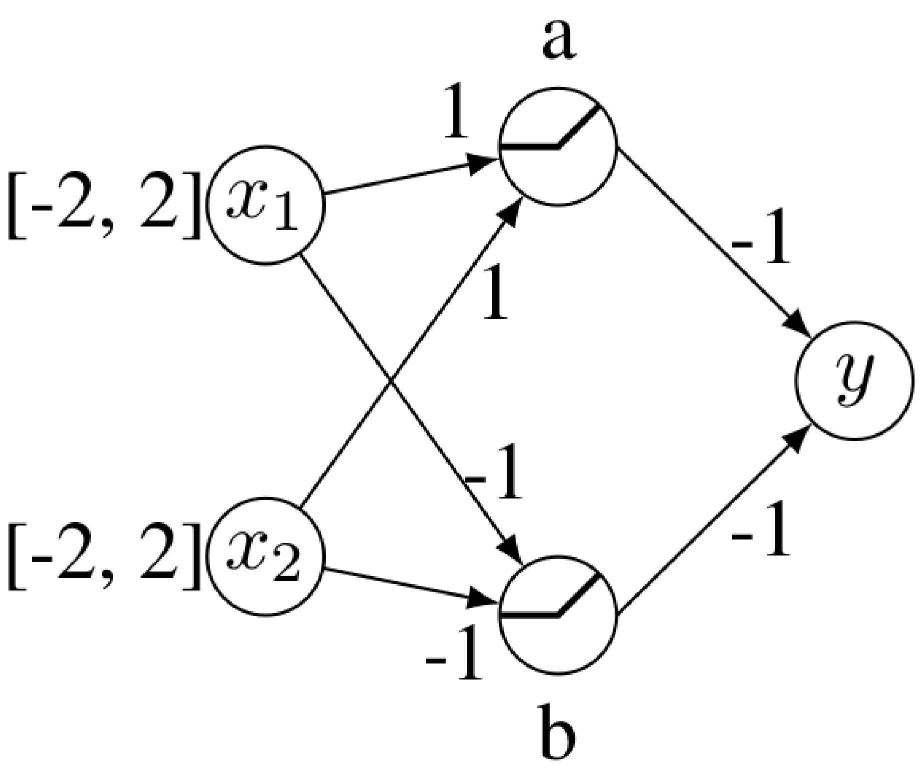
PyRAT: Python Reachability Assessment Tool
Application to NN

$$-2 \leq x_1 \leq 2$$
$$-2 \leq x_2 \leq 2$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y = -a_{\text{out}} - b_{\text{out}}$$



$a_{\text{in}} =$ [-2,2] + [-2,2] = [-4,4]

$b_{\text{in}} =$ - [-2,2] - [-2,2] =[-4,4]

$a_{\text{out}} =$ [0,4]

$b_{\text{out}} =$ [0,4]

$y =$

$\boxed{\textbf{Prove that } y > -5}$

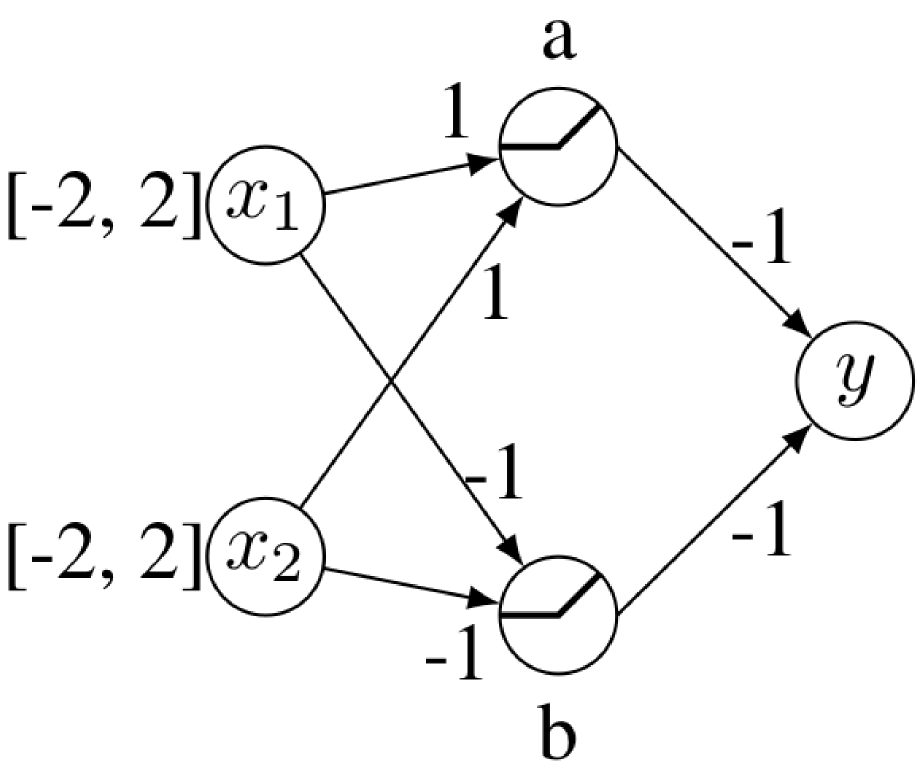
PyRAT: Python Reachability Assessment Tool
Application to NN

$$-2 \leq x_1 \leq 2$$
$$-2 \leq x_2 \leq 2$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y = -a_{\text{out}} - b_{\text{out}}$$



$a_{\text{in}} =$ [-2,2] + [-2,2] = [-4,4]

$b_{\text{in}} =$ - [-2,2] - [-2,2] =[-4,4]

$a_{\text{out}} =$ [0,4]

$b_{\text{out}} =$ [0,4]

$y =$ [-8,0]

$\boxed{\textbf{Prove that } y > -5}$

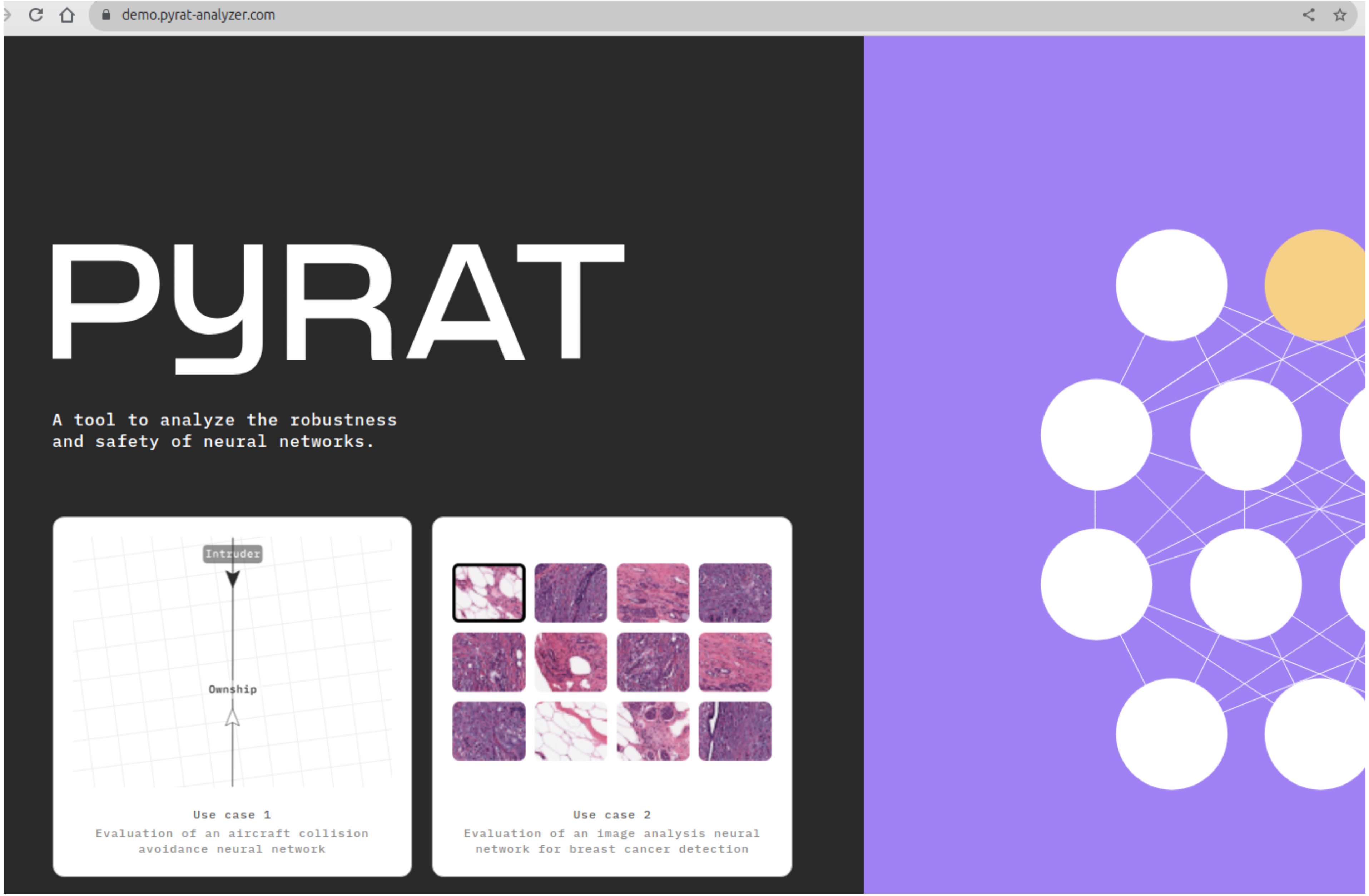
PyRAT: Python Reachability Assessment Tool

- 2nd at VNNComp 2024
- Written in Python with PyTorch and Numpy backend
- Supports common layers and architecture in ONNX, Keras/Tensorflow and PyTorch
- Different abstract domains implemented: Box, Zonotopes, Constrained Zonotopes, ...
- Integrated in CAISAR, an open-source platform for characterizing safety in AI systems.

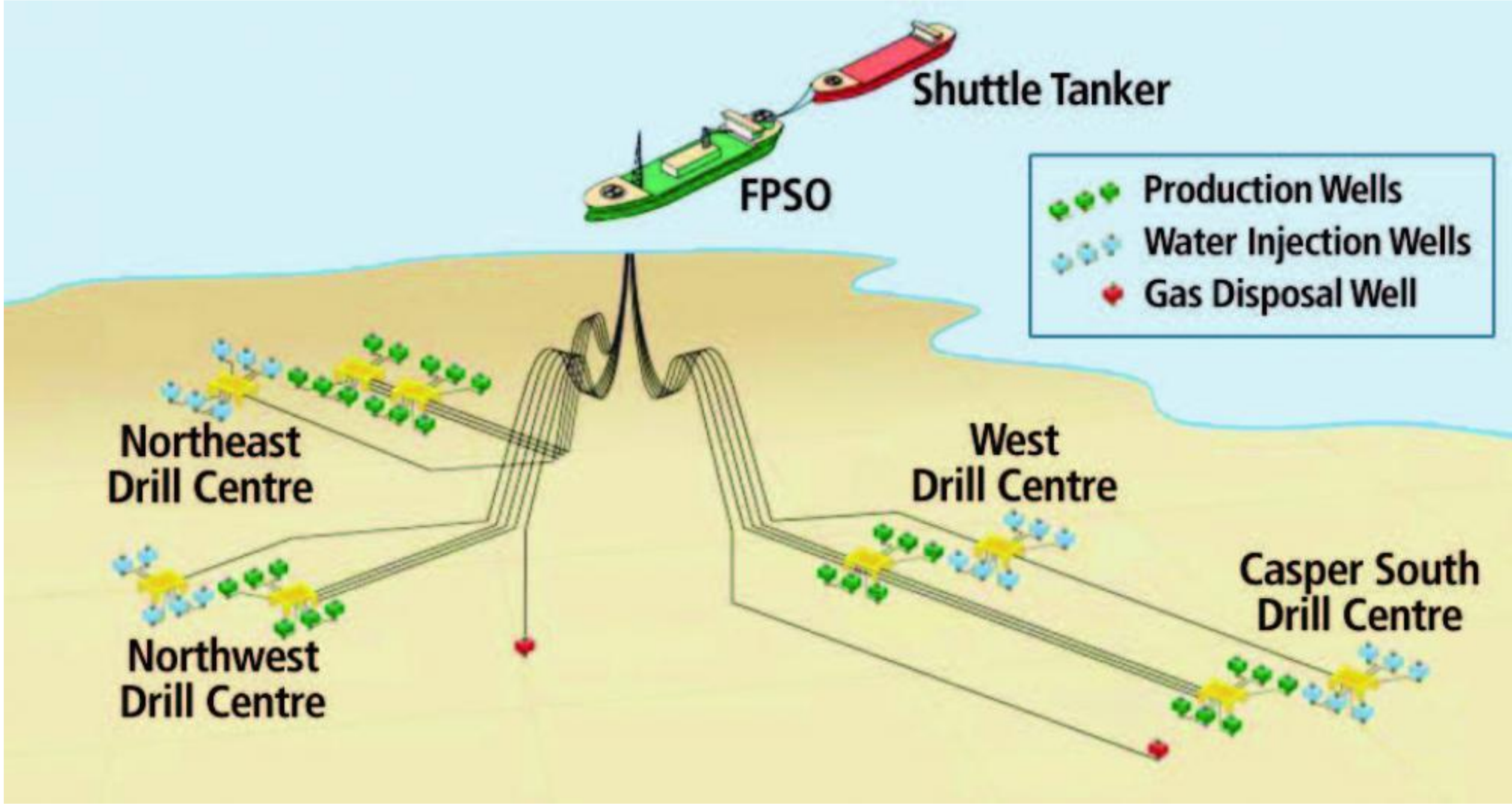
CAISAR

caisar-platform.com

PyRAT: Python Reachability Assessment Tool



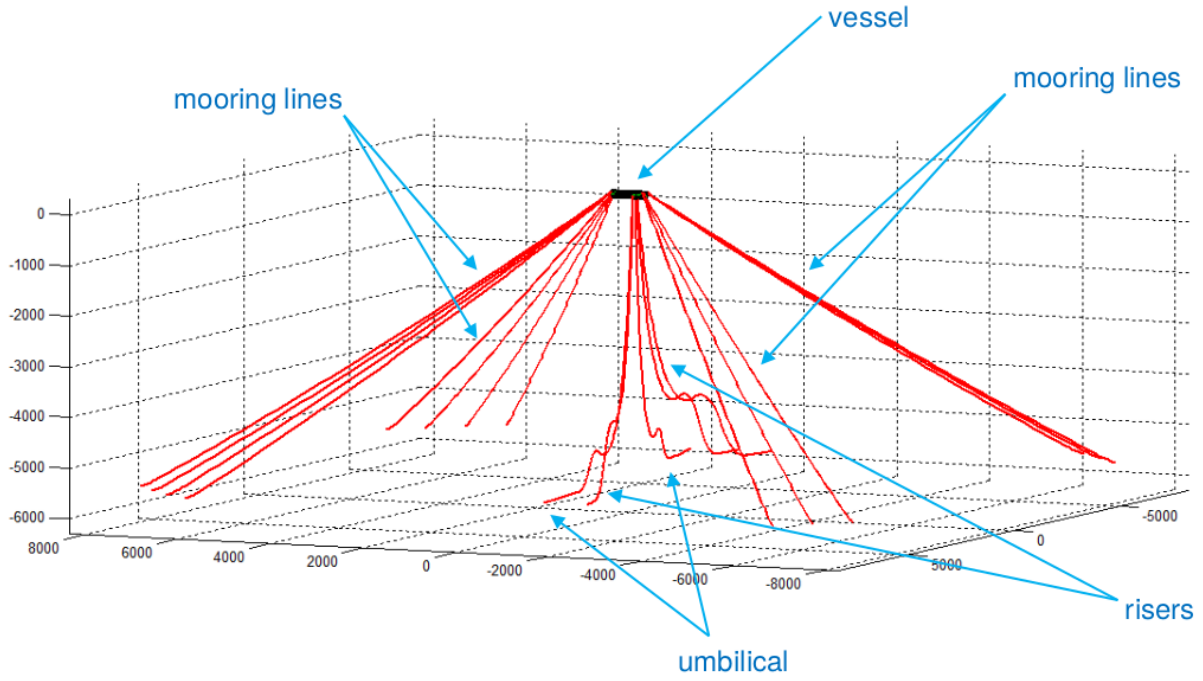
PyRAT: Python Reachability Assessment Tool
Use-case: Mooring lines failure detection



PyRAT: Python Reachability Assessment Tool
Use-case: Mooring lines failure detection

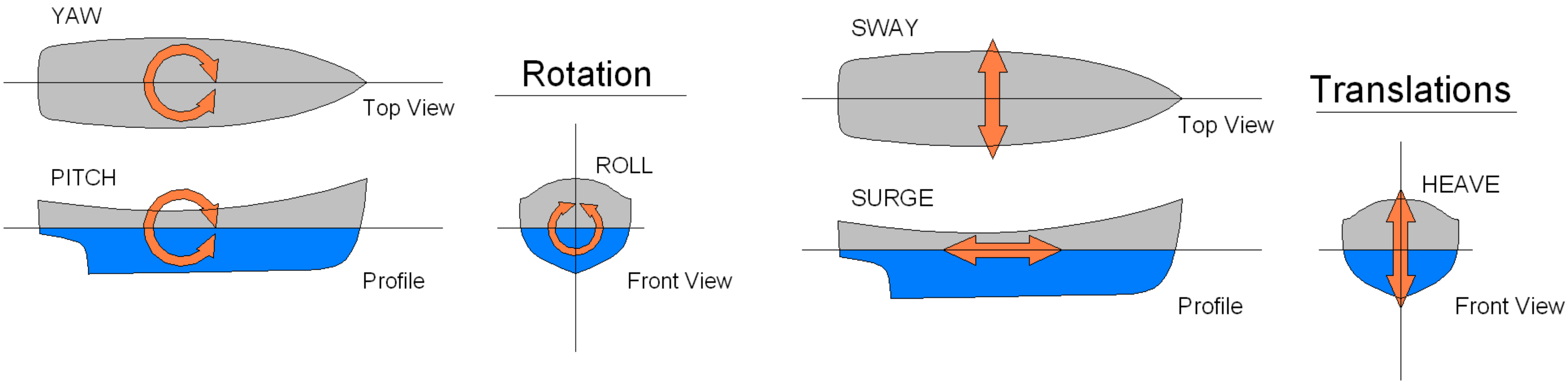
Mooring incidents (DeepStar® data from 1997-2012):

- 107 incidents from 73 facilities across the industry
- Potentially dire consequences
- Many FPSO have no means of monitoring lines
- Those who do face technical problems (robustness of equipment)



PyRAT: Python Reachability Assessment Tool
Use-case: Mooring lines failure detection

Patented dry monitoring detection systems, based on vessel positions and low-frequency periods (which can be obtained from Dual GPS)

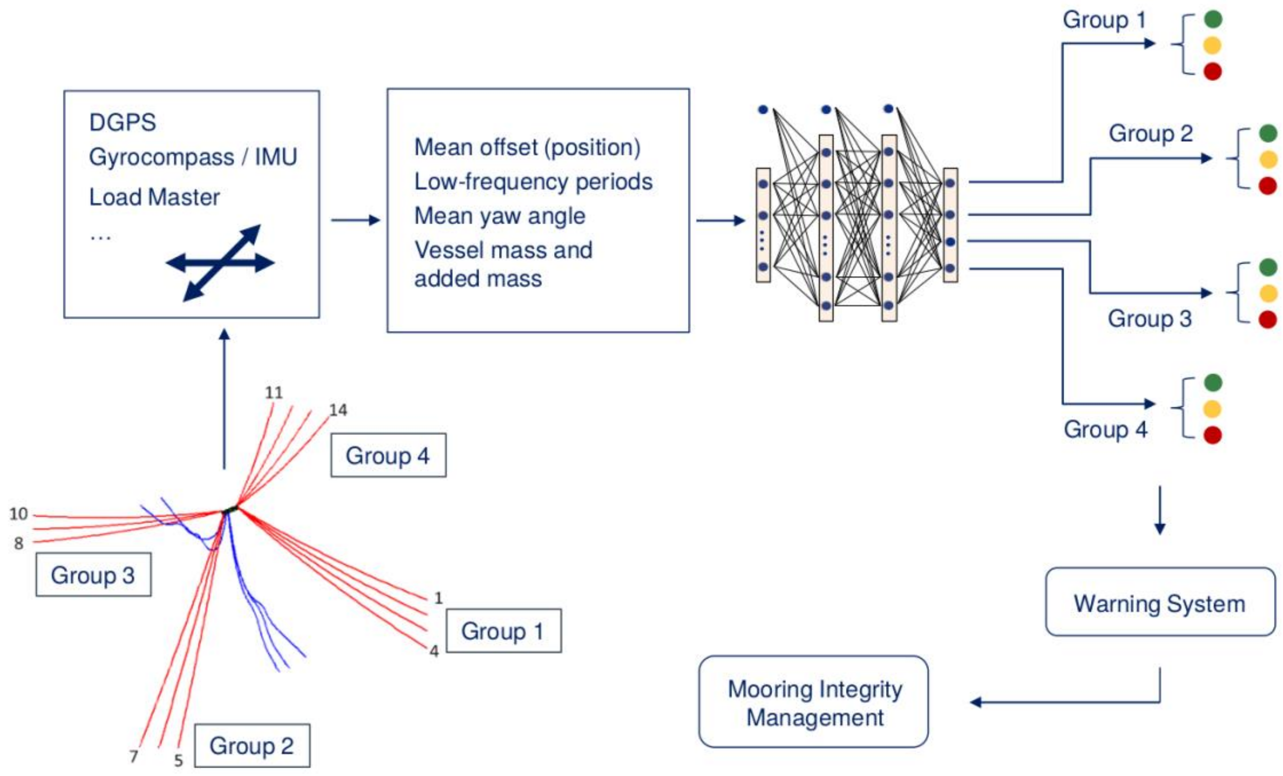


PyRAT: Python Reachability Assessment Tool
Use-case: Mooring lines failure detection

- Highly non-linear problem (machine learning to recognize and classify patterns)
- Ability to deal with some degrees of variations from various system components (such as mooring line stiffness) and with error or noise from monitoring system
- Cover a complete range of vessel drafts, expected vessel responses from environment conditions and directions and mooring line conditions

The model

- Input: Vessel movement, mass, offset, …
- Output: group-line failures

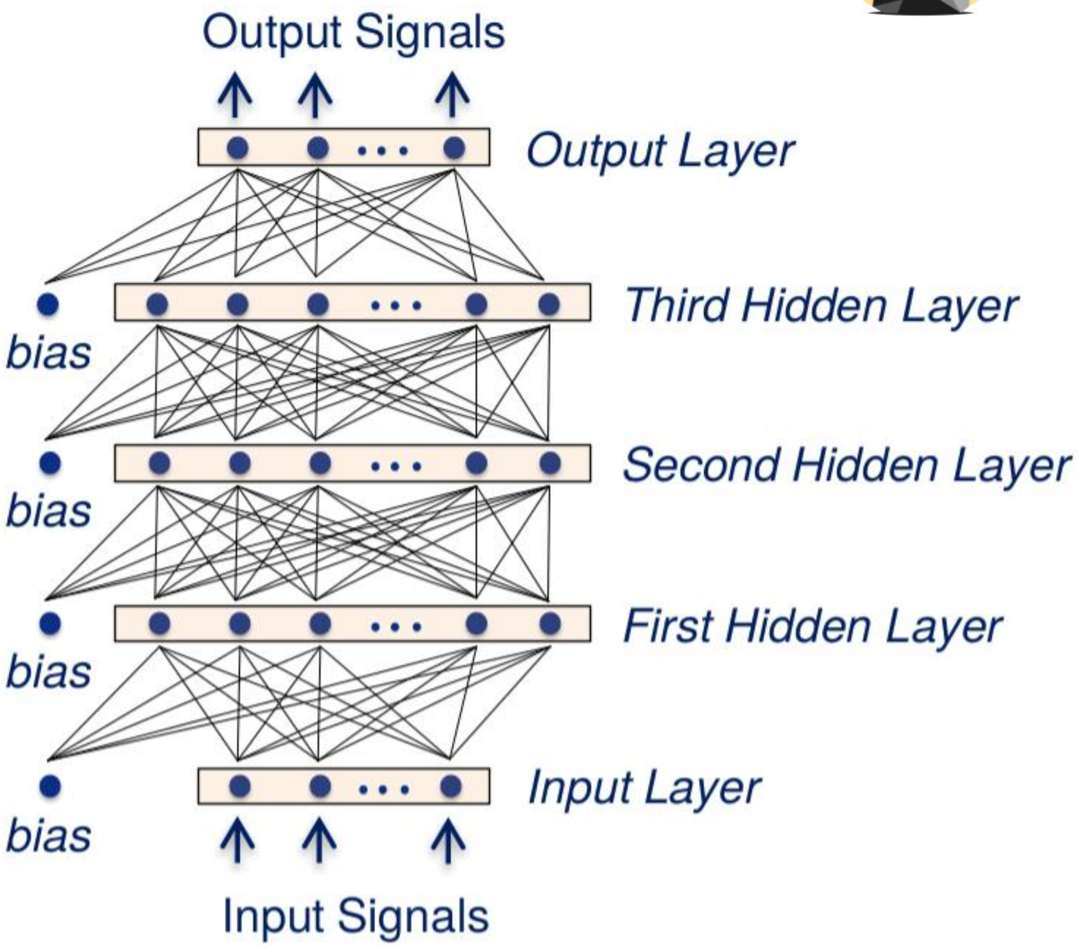


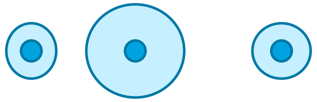
PyRAT: Python Reachability Assessment Tool
Use-case: Mooring lines failure detection

Ensuring robustness properties

- Stability of classification in presence of perturbation
- Perturbation per input (sensor sensitivity)
- Different perturbations for different inputs

(Also verified functional properties but NDA)





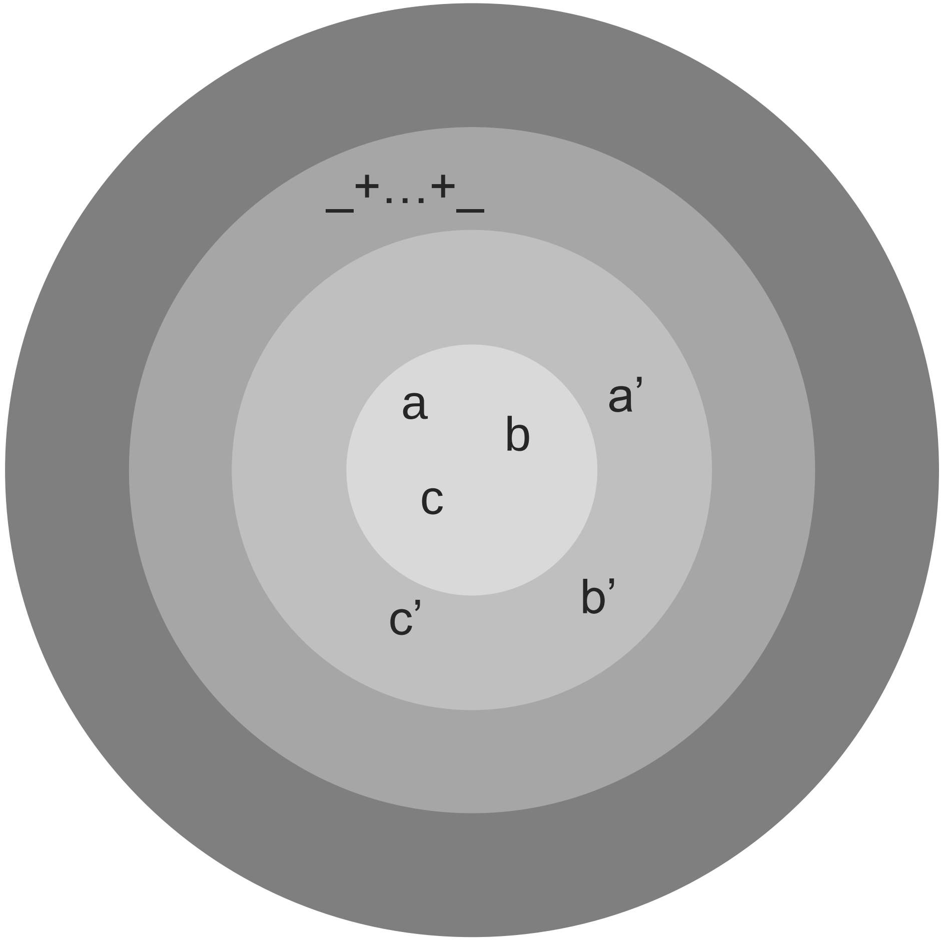
Rapid intro to FM

Examples for this talk:

- Property-based testing
- Abstract interpretation
- SMT Solving

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**



Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

Any logical formula can be converted into **Conjunctive Normal Form**

Satisfiability Modulo Theory

Equivalence rules can be used to translate any formula to CNF.

eliminate $\Rightarrow$	$A \Rightarrow B \equiv \neg A \vee B$
reduce the scope of $\neg$	$\neg(A \vee B) \equiv \neg A \wedge \neg B,$ $\neg(A \wedge B) \equiv \neg A \vee \neg B$
apply distributivity	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C),$ $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

Satisfiability Modulo Theory

However, there is a *linear time* translation to CNF that produces an *equisatisfiable* formula. Replace the distributivity rules by the following rules:

$$\frac{F[l_i \ op \ l_j]}{F[x], x \Leftrightarrow l_i \ op \ l_j}{*}$$

$$\frac{x \Leftrightarrow l_i \vee l_j}{\neg x \vee l_i \vee l_j, \neg l_i \vee x, \neg l_j \vee x}$$

$$\frac{x \Leftrightarrow l_i \wedge l_j}{\neg x \vee l_i, \neg x \vee l_j, \neg l_i \vee \neg l_j \vee x}$$

(*) x must be a fresh variable.

Satisfiability Modulo Theory

Translation of $(p \wedge (q \vee r)) \vee t$:

$$(p \wedge (q \vee r)) \vee t$$

$$(p \wedge x_1) \vee t, x_1 \Leftrightarrow q \vee r$$

$$x_2 \vee t, x_2 \Leftrightarrow p \wedge x_1, x_1 \Leftrightarrow q \vee r$$

$$x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, x_1 \Leftrightarrow q \vee r$$

$$x_2 \vee t, \neg x_2 \vee p, \neg x_2 \vee x_1, \neg p \vee \neg x_1 \vee x_2, \neg x_1 \vee q \vee r, \neg q \vee x_1, \neg r \vee x_1$$

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

a’+b
b’+c
c+e
c’+a’

Decide a

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

b

b’+c

c+e

c’

Decide a

Implies...

- b is true
- c is false

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a'). Clauses are disjunctions of literals (e.g. a+b'+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

b

c

c+e

c'

Decide a

Implies...

- b is true
  - c is true
- c is false

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

b

c

c+e

c’

Decide a

=> Contradiction.

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

a’+b
b’+c
c+e
c’+a’

~~Decide a~~

=> Contradiction.

Backtrack

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a'). Clauses are disjunctions of literals (e.g. a+b'+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

a'+b
b'+c
c+e
c'+a'

Now we know :
a **must be** *false*

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

b’+c

c+e

Now we know :

a **must be** *false*

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

b’+c
c+e

Decide b

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

c

c+e

Decide b

Implies

- c is true

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

c

c+e

Decide b

Implies

- c is true

Then e can have any value

Satisfiability Modulo Theory

A literal is an atom or its negation (a or a’). Clauses are disjunctions of literals (e.g. a+b’+c). A SAT problem is a conjunction of clauses. This is called **Conjunctive Normal Form.**

A great deal of research is dedicated to speed up SAT solving (a notoriously NP-complete problem), such as Conflict Driven Clause Learning, or symmetry breaking. But we will not see that here.

a’+b
b’+c
c+e
c’+a’

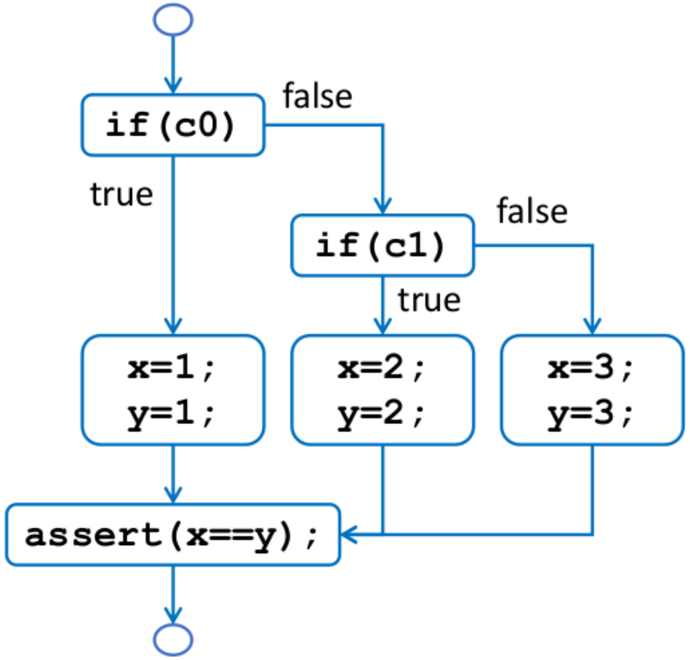
So now the assignments are

- False: a, True: b, c, e
- False: a, e, True: b, c

Satisfiability Modulo Theory

```
if(c0)
   x=1;y=1;
else
   if(c1)
      x=2;y=2;
   else
      x=3;y=3;
assert(x==y);
```

```
if(c0)
   x1=1;y1=1;
else
   if(c1)
      x2=2;y2=2;
   else
      x3=3;y3=3;
x4=ite(c0,x1,ite(c1,x2,x3));
y4=ite(c0,y1,ite(c1,y2,y3));
assert(x4==y4);
```



$$c_0 \longrightarrow x_4 = 1 \qquad c_0 \longrightarrow y_4 = 1$$

$$\neg c_0 \wedge c_1 \longrightarrow x_4 = 2 \qquad \neg c_0 \wedge c_1 \longrightarrow y_4 = 2$$

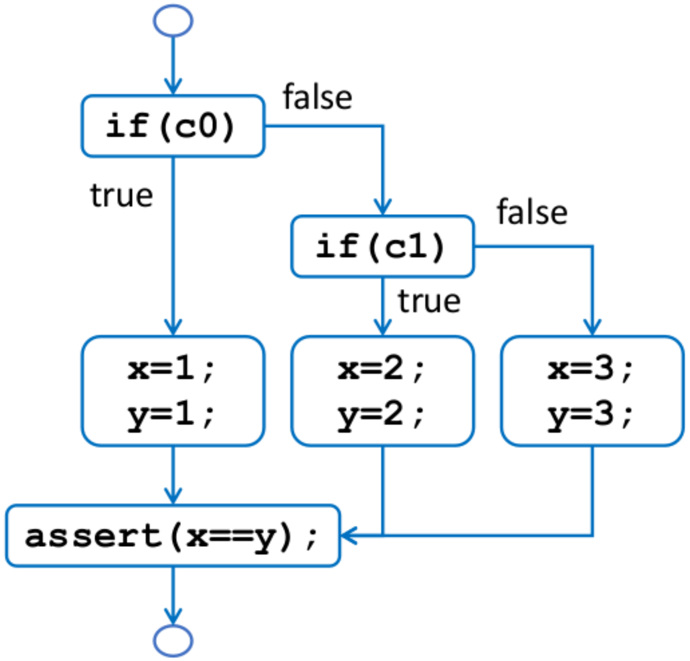
$$\neg c_0 \wedge \neg c_1 \longrightarrow x_4 = 3 \qquad \neg c_0 \wedge \neg c_1 \longrightarrow y_4 = 3$$

Control flow-guided smt solving for program verification (2018, Chen, Jianhui, and Fei He)

Satisfiability Modulo Theory

```
if(c0)
  x=1;y=1;
else
  if(c1)
    x=2;y=2;
  else
    x=3;y=3;
assert(x==y);
```

```
if(c0)
  x1=1;y1=1;
else
  if(c1)
    x2=2;y2=2;
  else
    x3=3;y3=3;
x4=ite(c0,x1,ite(c1,x2,x3));
y4=ite(c0,y1,ite(c1,y2,y3));
assert(x4==y4);
```



????

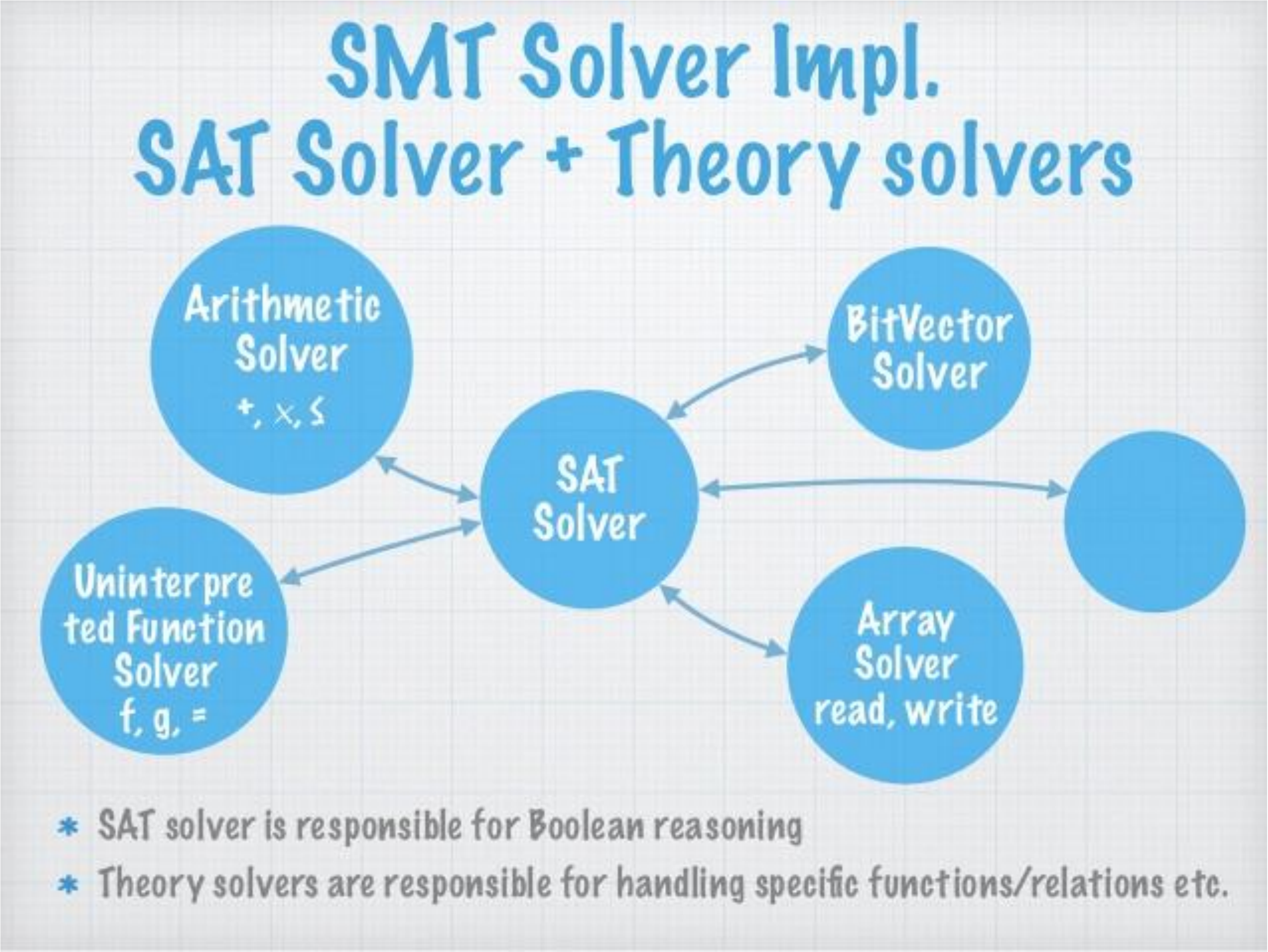
$$c_0 \longrightarrow x_4 = 1 \qquad c_0 \longrightarrow y_4 = 1$$

$$\neg c_0 \wedge c_1 \longrightarrow x_4 = 2 \qquad \neg c_0 \wedge c_1 \longrightarrow y_4 = 2$$

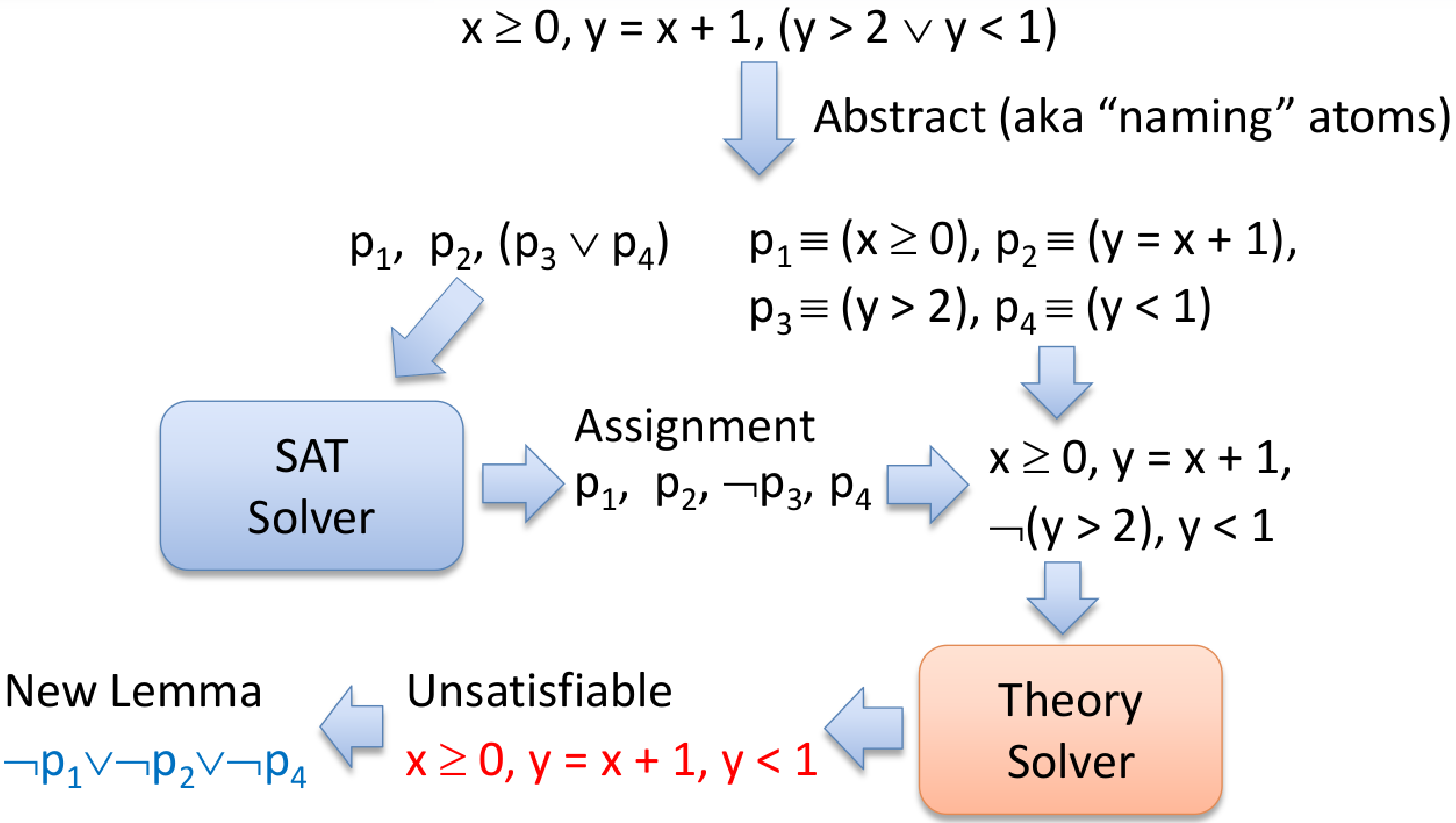
$$\neg c_0 \wedge \neg c_1 \longrightarrow x_4 = 3 \qquad \neg c_0 \wedge \neg c_1 \longrightarrow y_4 = 3$$

Control flow-guided smt solving for program verification (2018, Chen, Jianhui, and Fei He)

Satisfiability Modulo Theory

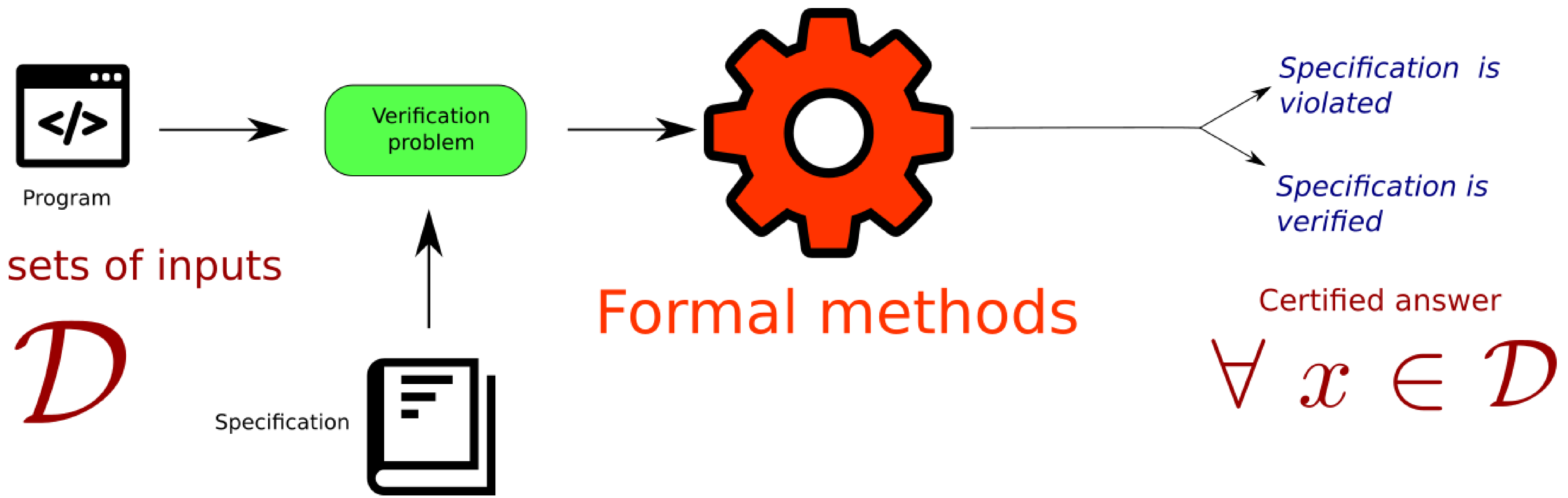


Satisfiability Modulo Theory

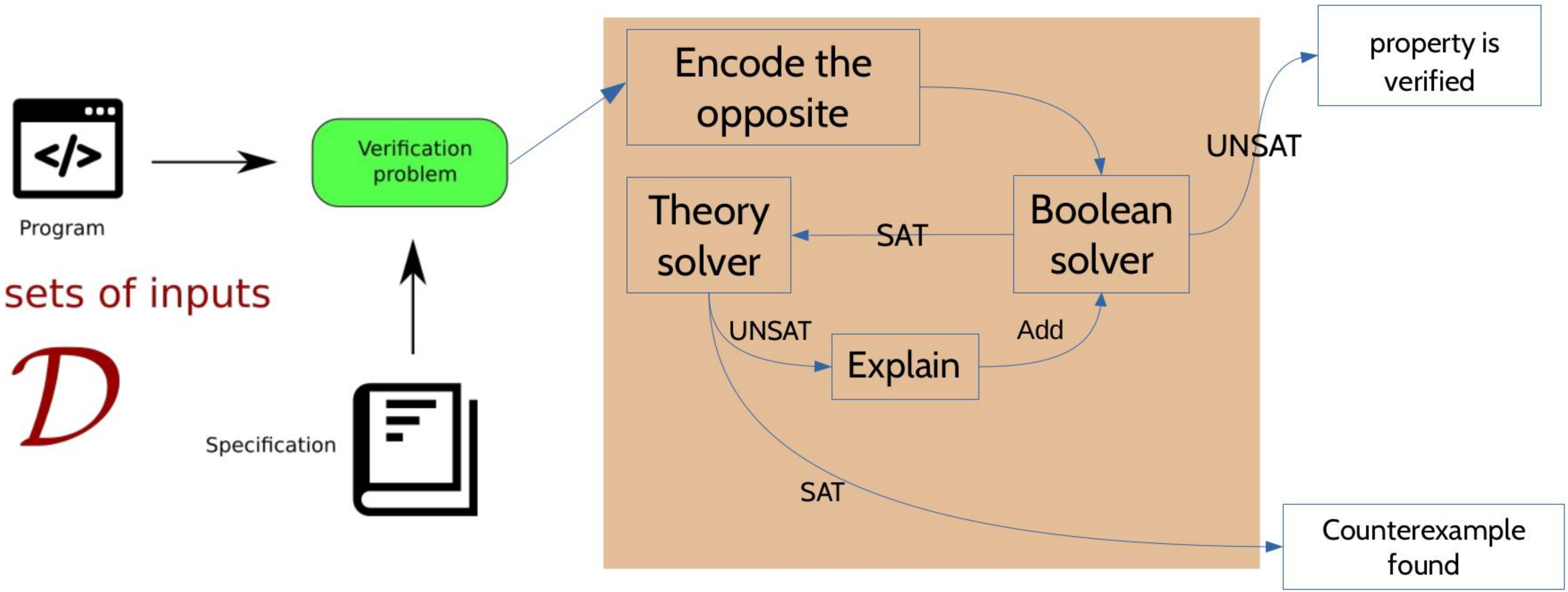


Images from Leonardo de Moura

Satisfiability Modulo Theory

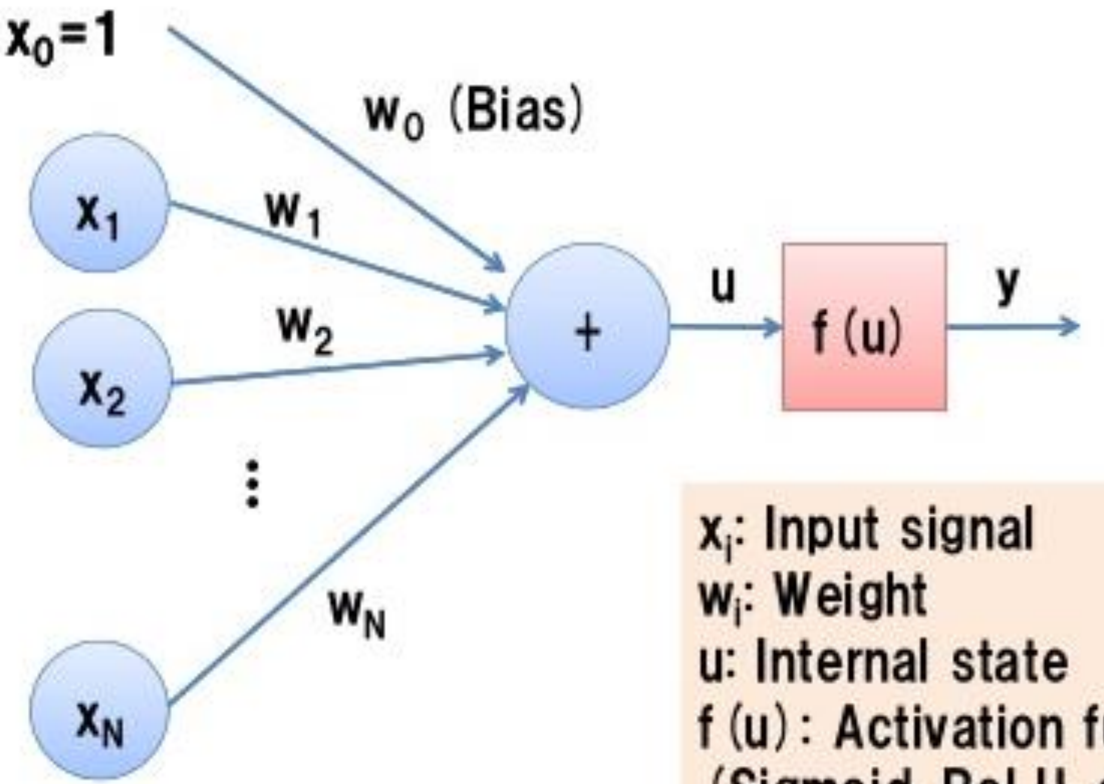


Satisfiability Modulo Theory



Satisfiability Modulo Theory

Artificial Neuron

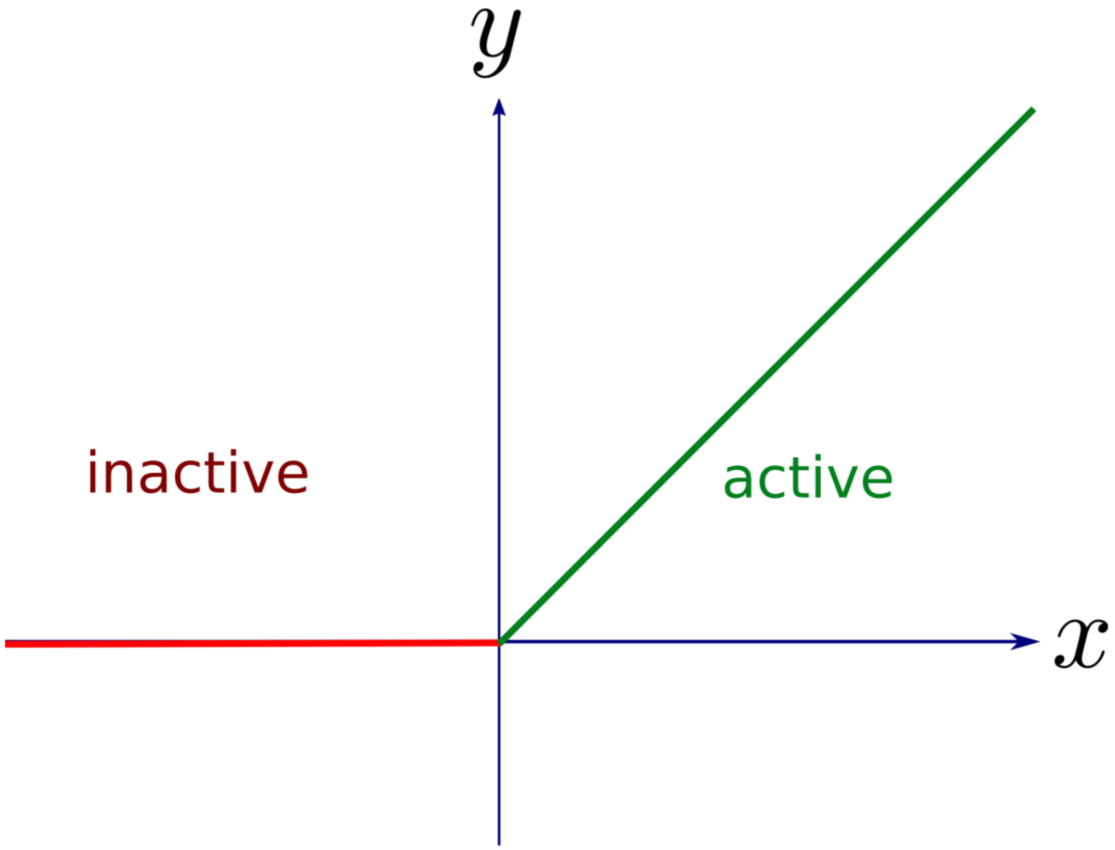


$$y = f(u)$$

$$u = \sum_{i=0}^{N} w_i x_i$$

x_i: Input signal
w_i: Weight
u: Internal state
f (u) : Activation function (Sigmoid, ReLU, etc.)
y: Output signal

Satisfiability Modulo Theory



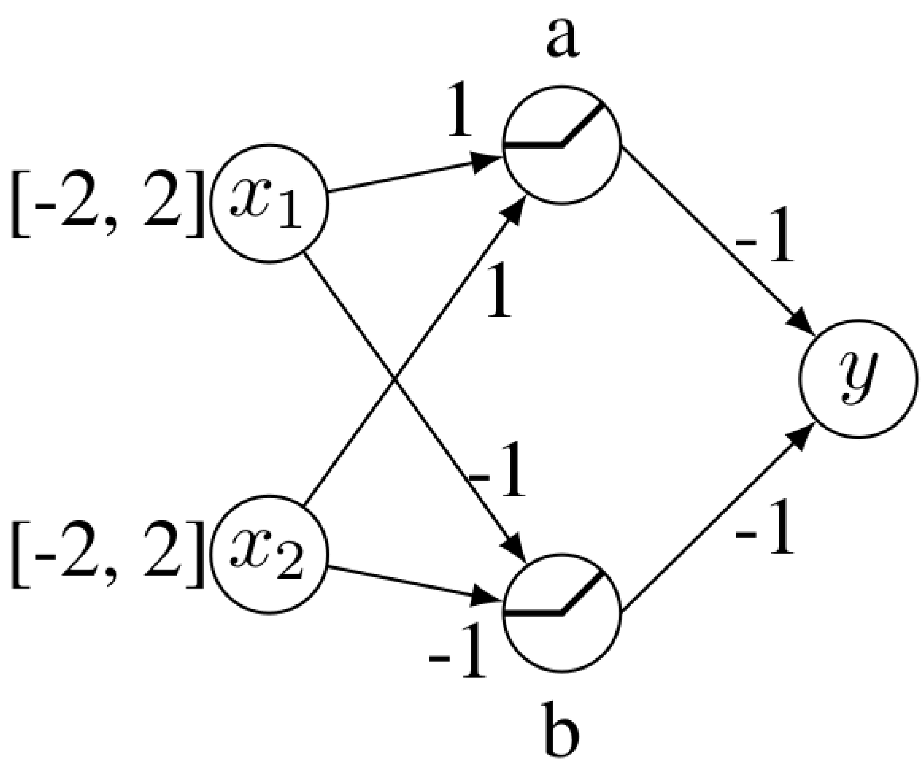
$$ReLU : x \rightarrow max(x, 0)$$

Satisfiability Modulo Theory

We know how to encode an ite already:
y = If (x < 0) then 0 else x
Becomes
(x < 0) => (y = 0)
(x >= 0) => (y = x)
Which becomes
Not (x<0) or y = 0
Not (x>=0) or y = x

We now that the relu is just a max, which is just an ite.
So let's speak simpler

Satisfiability Modulo Theory



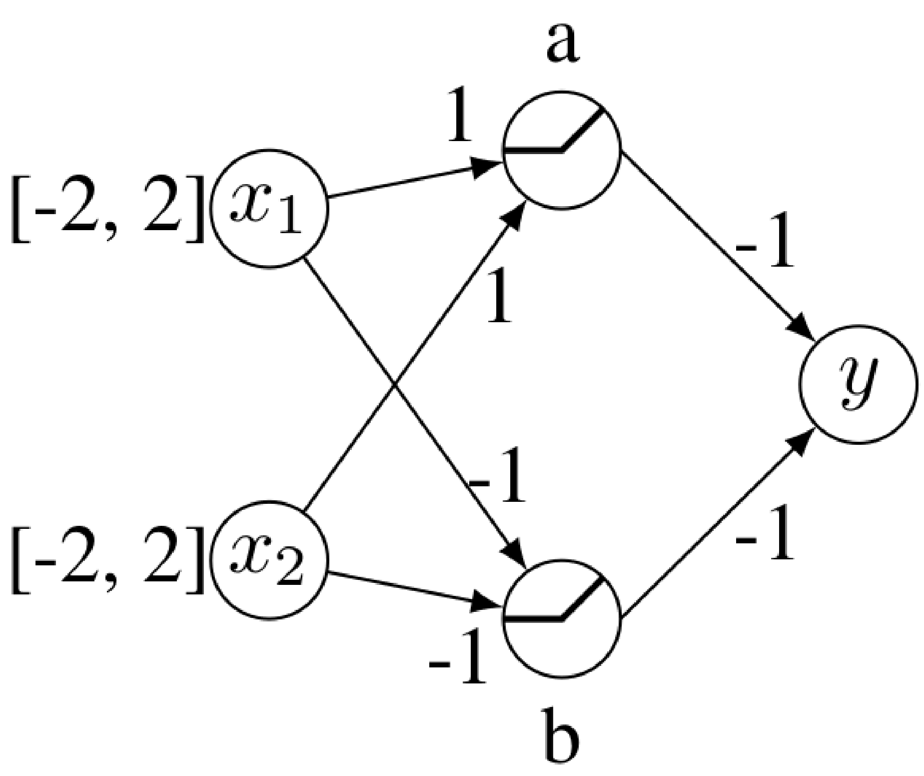
Prove that $y > -5$

Satisfiability Modulo Theory

$$-2 \leq x_1 \leq 2$$
$$a_{\text{in}} = x_1 + x_2$$
$$a_{\text{out}} = \max(a_{\text{in}}, 0)$$

$$-2 \leq x_2 \leq 2$$
$$b_{\text{in}} = -x_1 - x_2$$
$$b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y \leq -5$$
$$y = -a_{\text{out}} - b_{\text{out}}$$



Prove that $y > -5$

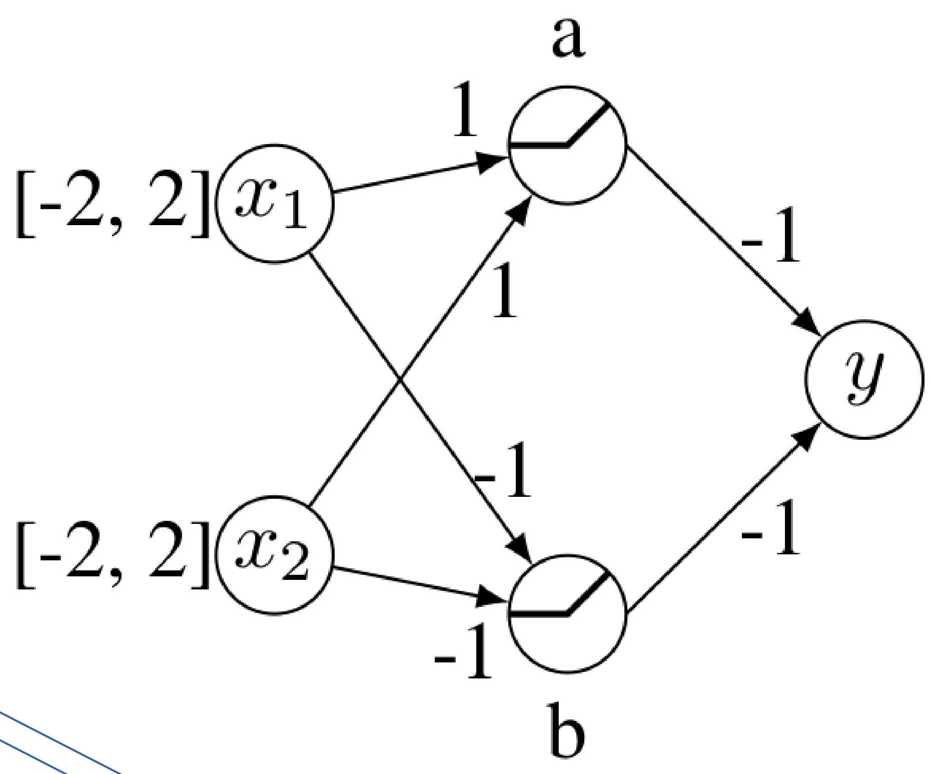
Piecewise Linear Neural Network verification: A comparative study (2017, Bunel, Turkaslan, Torr, Kohli)

Satisfiability Modulo Theory

$$-2 \leq x_1 \leq 2$$
$$a_{\text{in}} = x_1 + x_2$$
$$a_{\text{out}} = \max(a_{\text{in}}, 0)$$

$$-2 \leq x_2 \leq 2$$
$$b_{\text{in}} = -x_1 - x_2$$
$$b_{\text{out}} = \max(b_{\text{in}}, 0)$$

$$y \leq -5$$
$$y = -a_{\text{out}} - b_{\text{out}}$$



Prove that $y > -5$

Piecewise Linear Neural Network verification: A comparative study (2017, Bunel, Turkaslan, Torr, Kohli)

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
```

```
(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))
```

```
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
```

```
(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))

(assert (= ain (+ x1 x2)))
(assert (= bin (- x1 (- x2))))
```

```
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))

(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))

(assert (= ain (+ x1 x2)))
(assert (= bin (- x1 (- x2))))

(assert (= aout (reluR ain)))
(assert (= bout (reluR bin)))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
```

```
(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))

(assert (= ain (+ x1 x2)))
(assert (= bin (- x1 (- x2))))
```

```
(assert (= aout (reluR ain)))
(assert (= bout (reluR bin)))

(assert (= y (- aout (- bout))))
```

```
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)
(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))

(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))

(assert (= ain (+ x1 x2)))
(assert (= bin (- x1 (- x2))))

(assert (= aout (reluR ain)))
(assert (= bout (reluR bin)))

(assert (= y (- aout (- bout))))

(assert (<= y -5))
```

Satisfiability Modulo Theory – SMT-Lib Standard

$$-2 \leq x_1 \leq 2 \qquad -2 \leq x_2 \leq 2 \qquad y \leq -5$$

$$a_{\text{in}} = x_1 + x_2 \qquad b_{\text{in}} = -x_1 - x_2 \qquad y = -a_{\text{out}} - b_{\text{out}}$$

$$a_{\text{out}} = \max(a_{\text{in}}, 0) \qquad b_{\text{out}} = \max(b_{\text{in}}, 0)$$

```
(declare-fun x1 () Int)
(declare-fun x2 () Int)
(declare-fun ain () Int)
(declare-fun aout () Int)
(declare-fun bin () Int)
(declare-fun bout () Int)
(declare-fun y () Int)

(assert (<= (- 2) x1))
(assert (<= (- 2) x2))
(assert (<= x1 2))
(assert (<= x2 2))

(assert (= ain (+ x1 x2)))
(assert (= bin (- x1 (- x2))))

(assert (= aout (reluR ain)))
(assert (= bout (reluR bin)))

(assert (= y (- aout (- bout))))

(assert (<= y -5))
(check-sat)

(define-fun relu ((res Int )) Int (ite (> res 0 ) res 0 ))
```

Tedious isn’t it...

CAISAR

Characterizing Artificial Intelligence Safety and Reliability

A federative platform for analysis of artificial intelligence system components

cea

What CAISAR is

Principle: Maximize coverage of AI models and properties

- Common expressive specification language
- Easy extensibility through clear interfaces
- Heuristic-aided V&V analysis
- Common aggregation of analysis outputs

Target: SVM, Neural Networks, XGBoost models, ensemble models,…

Application: depending on the used plug-ins. Currently includes

- SAVer for SVM
- Colibri for XGboost
- PyRAT, AB-Crown, Nnenum, Marabou for NN

Background: The federative platform strategy for V&V has been successful for critical SW (see, for example, Frama-C and Why3)

What CAISAR is
Characterizing AI Safety And Robustness

Aimed at all AI systems

While the current frenzy of AI trustworthiness is mostly focused neural networks, our industrial partners can also use other types of AI (e.g. SVM, XGBoost) in their products. This is why **CAISAR targets a wider range of AI systems.**

Standard oriented

By relying on AI standards (ONNX, NNnet) and formal methods standards (SMT, CP), CAISAR maximizes the potential for **inclusiveness**. Any tool that supports these standards is a potential addition to the CAISAR platform.

Modular and extensible

Written in the functional language OCaml, adding a verification, analysis or testing software to CAISAR's toolsuit is made easier through a unified interface, and an instantiation guided by data-types.

Interoperability

An internal representation for property language and AI representation helps reinforce the **synergy between the different tools** in CAISAR, where one analyser can rely on, and complement, the partial output of another.

Maintainable

Functional programming provides easy mathematical reading, lowering the entry barrier for understanding the inner workings of CAISAR. Strong typing also minimizes errors with informative messages.

Open-source

Our vision of CAISAR is collaborative in essence. To encourage cooperation and build a community of trust-minded entities, an easy access to the platform, supplied with documentations and tutorials, is our priority.

What CAISAR is
Characterizing AI Safety And Robustness

Property ϕ_1.

- Description: If the intruder is distant and is significantly slower than the ownship, the score of a COC advisory will always be below a certain fixed threshold.
- Tested on: all 45 networks.
- Input constraints: $\rho \geq 55947.691$, $v_{\text{own}} \geq 1145$, $v_{\text{int}} \leq 60$.
- Desired output property: the score for COC is at most 1500.

```
let function normalize_t (i: t) (mean: t) (range: t) : t =
  (i .- mean) ./ range
let function denormalize_t (i: t) (mean: t) (range: t) : t =
  (i .* range) .+ mean
let function normalize_input (i: input) : input =
  Vector.mapi i normalize_by_index
let function denormalize_output_t (o: t) : t =
  denormalize_t o
    (7.51888402010059753166615337249822914600372314453125:t)
      (373.9499200000000200816430151462554931640625:t)
let runP1 (i: input) : t
  requires { has_length i 5 }
  (* constraints the inputs to respect the specification *)
  requires { valid_input i }
  requires { intruder_distant_and_slow i }
  ensures { result .≤ (1500.0:t) }  =
    let j = normalize_input i in
    let o = (nn @@ j)[clear_of_conflict] in
    (denormalize_output_t o)
```

What CAISAR is
Characterizing AI Safety And Robustness

```
goal pruned:
  CSV.forall_ dataset (fun _ e →
    forall perturbed_e.
      has_length perturbed_e (length e) →
      FeatureVector.valid feature_bounds perturbed_e →
      let perturbation = perturbed_e - e in
      ClassRobustVector.bounded_by_epsilon perturbation eps →
      let out_1 = nn_1@@perturbed_e in
      let out_2 = nn_2@@perturbed_e in
      .- delta .≤ out_1[0] .- out_2[0] .≤ delta
  )
```

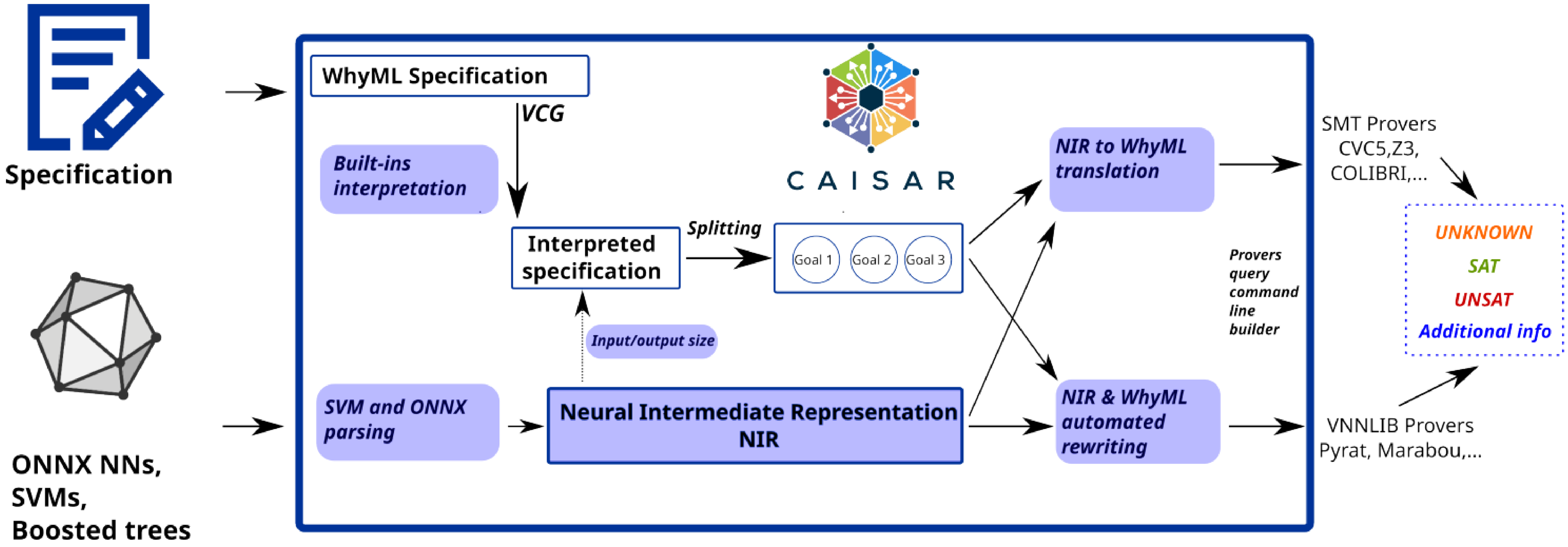
Fig. 13: A WhyML specification with several NNs at once

What CAISAR is
Characterizing AI Safety And Robustness

```
goal splitted:
  CSV.forall_ dataset (fun l e →
    forall perturbed_e.
      has_length perturbed_e (length e) →
      FeatureVector.valid feature_bounds perturbed_e →
      let perturbation = perturbed_e - e in
      ClassRobustVector.bounded_by_epsilon perturbation eps →
      let out1 = pre_nn@@perturbed_e in
      let out2 = post_nn@@out1 in
      forall j. Label.valid label_bounds j → j ≠ l →
      out2[l] .≥ out2[j]
)
```

Fig. 14: A WhyML specification for the composition of NNs

What CAISAR is (going to be)
Characterizing AI Safety And Robustness



Our lab

Verification of safety and robustness formal specifications through Abstract Interpretation	Metamorphic testing applied to AI (Available for teaching)	Open-source, modular, extensible platform to Characterize AI Safety And Robustness	Open-source Symbolic AI tools, Safe-by-design Constraint solvers	Case-based reasoning, explainability, out-of-distribution detection
PyRAT	AIMOS	CAISAR	Colibri & co	PARTICUL
Verification	**Test**	**Platform**	**Symbolic**	**XAI & uncertainty**

Symbolic AI: Colibri's

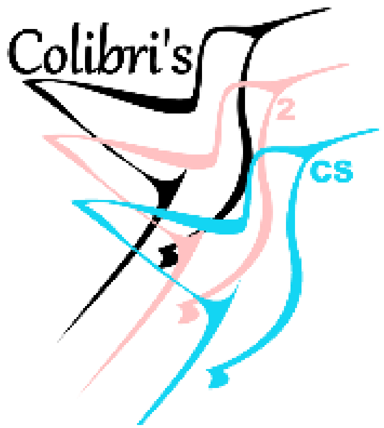
Principle: Safe-by-design Symbolic AI through a constraint solving library

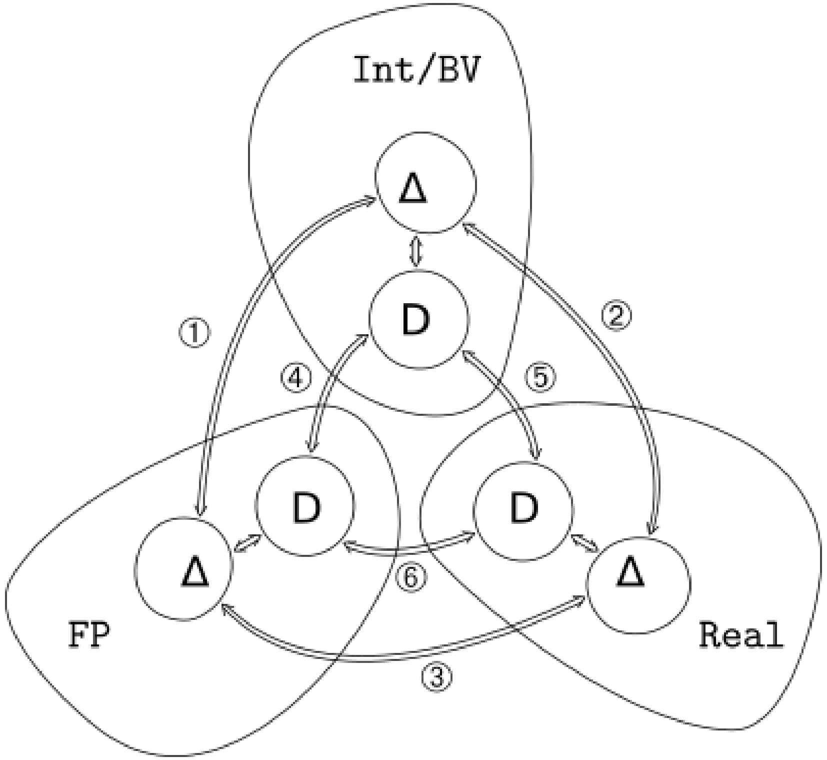
- Separately prove, in Why3, the necessary bricks for constraint solving: Floating-point numbers, integers, bit-vectors, strings, etc.
- Allow for selection of these bricks to tailor the construction of a solver to the needs of the user
- Automatically extract a C implementation of the solver

Target: XGBoost models, embedded software

Application: Energy sector (e.g., IRSN), space (e.g., NASA). Can also be used as a verification tool (winner of SMT-Competition since 2017), which makes it an essential brick of other tools such as Frama-C and GATeL.

Background: Constraint solving is used in several critical software domains





Colibri.frama-c.com

“ Cool, but where do I start ?

Potential user

Who are you? What do you want?

- You're a student, a researcher, or a professional who want to evaluate solutions…
  - You want automatic test generation with AIMOS or verification with PyRAT: free licence available
  - You want a more general way of testing and verification: CAISAR is open-source
  - You want to play around with XAI methods : CaBRNet is open-source

- You're a teacher and want to use tools in lab sessions for your students
  - Same as above but there are also course material available that we can help you adapt

- You're a professional who wants to use tools in a production setting
  - Open-source platforms and their documentations are available, support licenses are possible
  - License is possible for closed-source tools PyRAT and AIMOS

- When in doubt: contact us ! Zakaria.chihani@cea.fr

Not the complete picture…

<u>“Traditional” human-written software</u>

General purpose provers and platforms (Why3, Alt-Ergo, Z3, Colibri, TLA+, K-framework,…)

B-Method

Krakatoa, OpenJML (java programs)

SPARK (Ada programs)

Papyrus

SCADE

GATeL (Lustre programs)

Frama-C, Astrée

Polyspace (C/C++ programs)

CompCert (Compilation of C)

CoVaC (Compiler validation)

aiT WCET analyzers

seL4 (proven microkernel)

Unisim

BAP

BINSEC

Miasm

Software → High level (model… source code…) …… … low level → Machine

CNN-Cert

Planet

ReluVal

Not NN

SAVer

Neurify

Eran

PyRAT

Sherlock

MIPVerify

Differentiable Abstract Interpretation for Provably Robust Neural Networks
Matthew Mirman[1] Timon Gehr[1] Martin Vechev[1]

Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation
Antoine Miné
Sorbonne Universités, UPMC Univ. Paris 06, CNRS, LIP6

AI[2]: Safety and Robustness Certification of Neural Networks with Abstract Interpretation

Symbolic Execution for Deep Neural Networks
Divya Gopinath[1], Kaiyuan Wang[2], Mengshi Zhang[2], Corina S. Păsăreanu[1], Sarfraz Khurshid[2]

Algorithms for Verifying Deep Neural Networks
Changliu Liu
Carnegie Mellon University, Pittsburgh, PA 15213
Tomer Arnon
Christopher Lazarus
Clark Barrett
Mykel J. Kochenderfer
Stanford University, Stanford, CA 94305

Symbolic Execution for Importance Analysis and Adversarial Generation in Neural Networks
Divya Gopinath*, Mengshi Zhang†, Kaiyuan Wang†, İsmet Burak Kadron‡, Corina S. Păsăreanu*, Sarfraz Khurshid

New Passive and Active Attacks on Deep Neural Networks in Medical Applications
Invited Talk
Cheng Gongye, Hongjia Li, Xiang Zhang, Majid Sabbagh, Geng Yuan, Xue Lin, Thomas Wahl, and Yunsi Fei

EXPLOITING VERIFIED NEURAL NETWORKS VIA FLOATING POINT NUMERICAL ERROR
TECHNICAL REPORT
Kai Jia, MIT CSAIL, jiakai@mit.edu
Martin Rinard, MIT CSAIL, rinard@csail.mit.edu

Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search
Adnan Siraj Rakin†, Zhezhi He† and Deliang Fan

GPUVerify: A Verifier for GPU Kernels*
Adam Betts[1] Nathan Chong[1] Alastair F. Donaldson[1] Shaz Qadeer[2] Paul Thomson[1]

<u>Machine generated models</u>