A Certified Proof Checker for Deep Neural Network Verification in Imandra

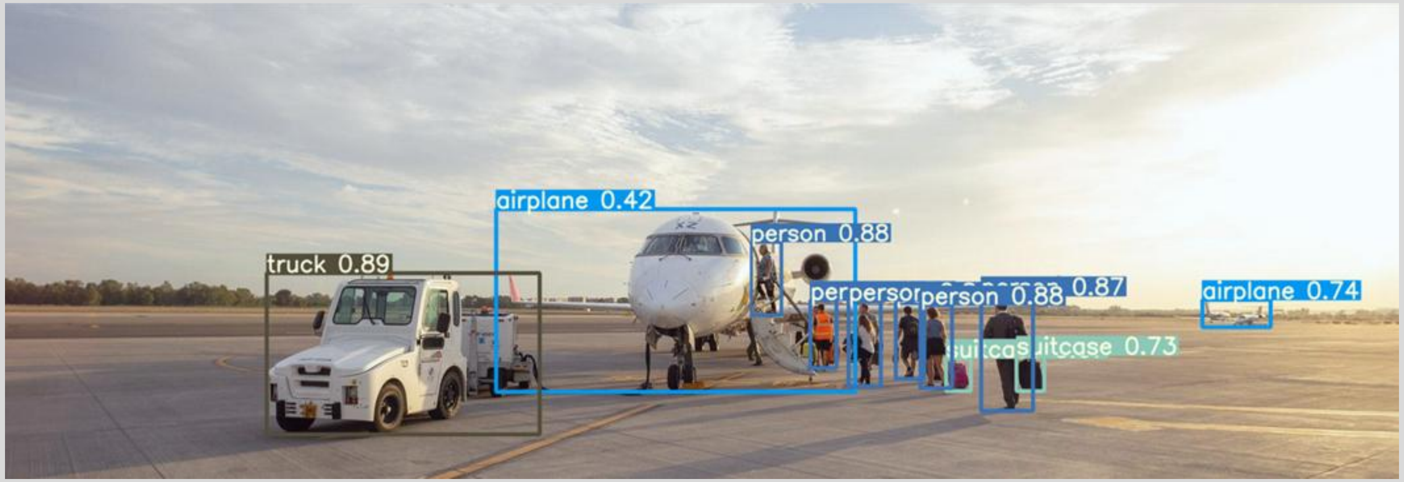
Remi Desmartin*, Omri Isac*, Grant Passmore, Ekaterina Komendantskaya, Kathrin Stark, Guy Katz

ITP, October 2025

Remi Desmartin, PhD student

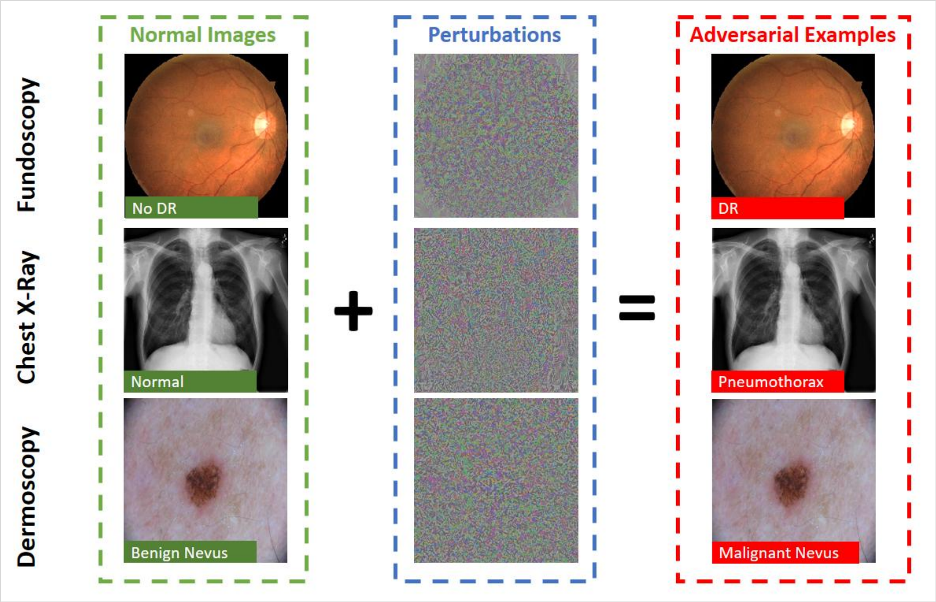
Heriot-Watt University

Deep Neural Networks (DNNs) accomplish groundbreaking results in many fields, including safety-critical.



Unlike traditional software, DNNs are opaque functions, trained over a large set of input and output examples.

This may lead to undesirable behavior of DNNs, which is **hard to predict** and **potentially dangerous.**

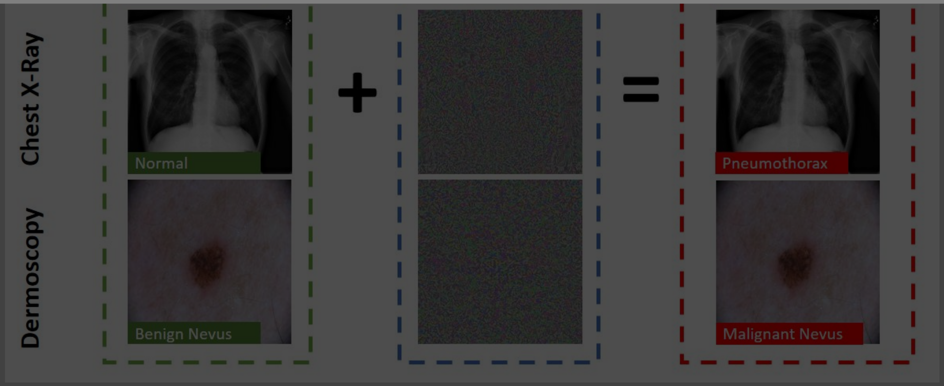


Understanding Adversarial Attacks on Deep Learning Based Medical Image Analysis Systems, Ma et al., 2019

This may lead to undesirable behavior of DNNs, which is hard to predict and potentially dangerous.

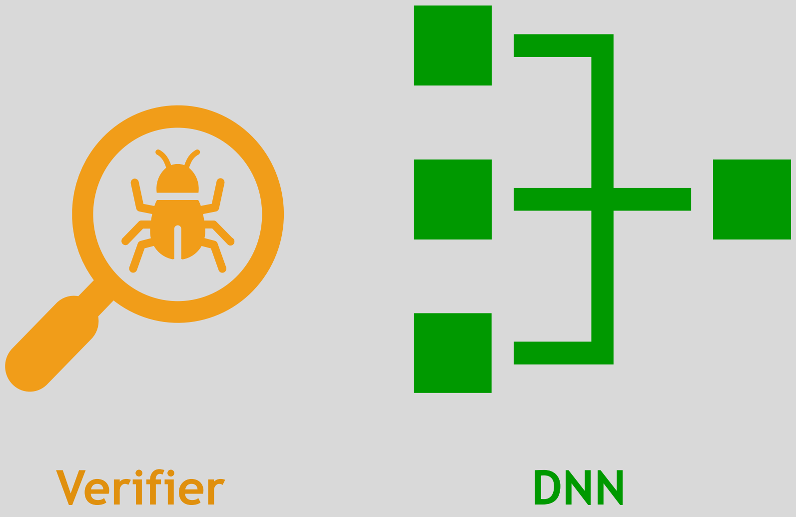
Can we rule out possibility of (some) errors?

Or find hidden errors, if exist

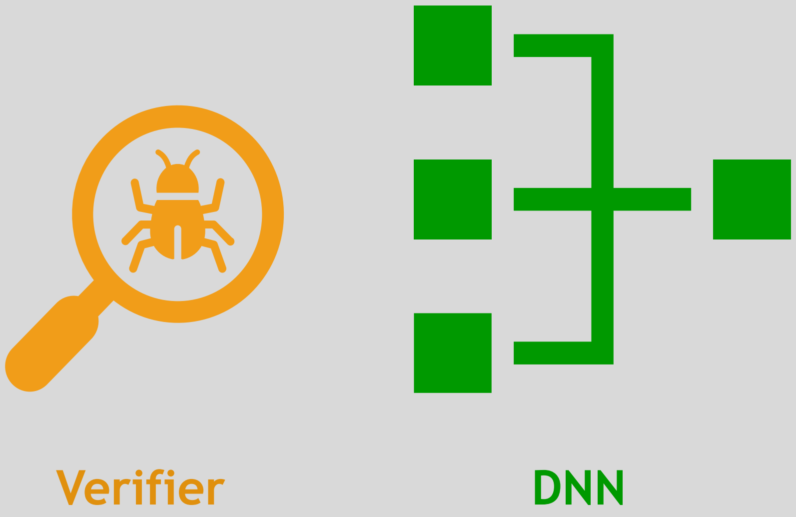


Understanding Adversarial Attacks on Deep Learning Based Medical Image Analysis Systems, Ma et al., 2019

The verification community developed various DNN verifiers, based on SMT, abstract interpretation, MILP and more.



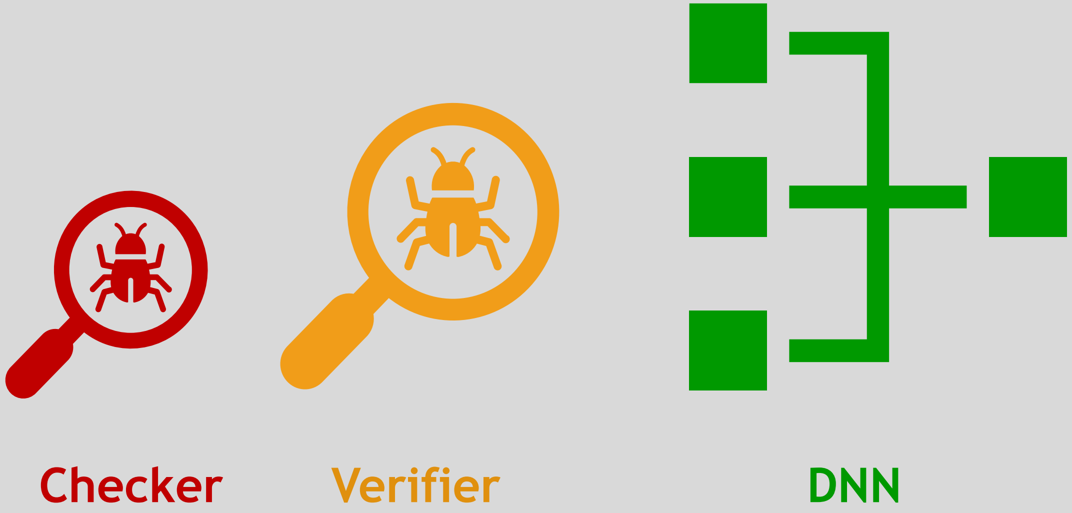
DNN verifiers are prone to bugs and may be numerically unstable, which can be exploited [Jia & Rinard, 2021].



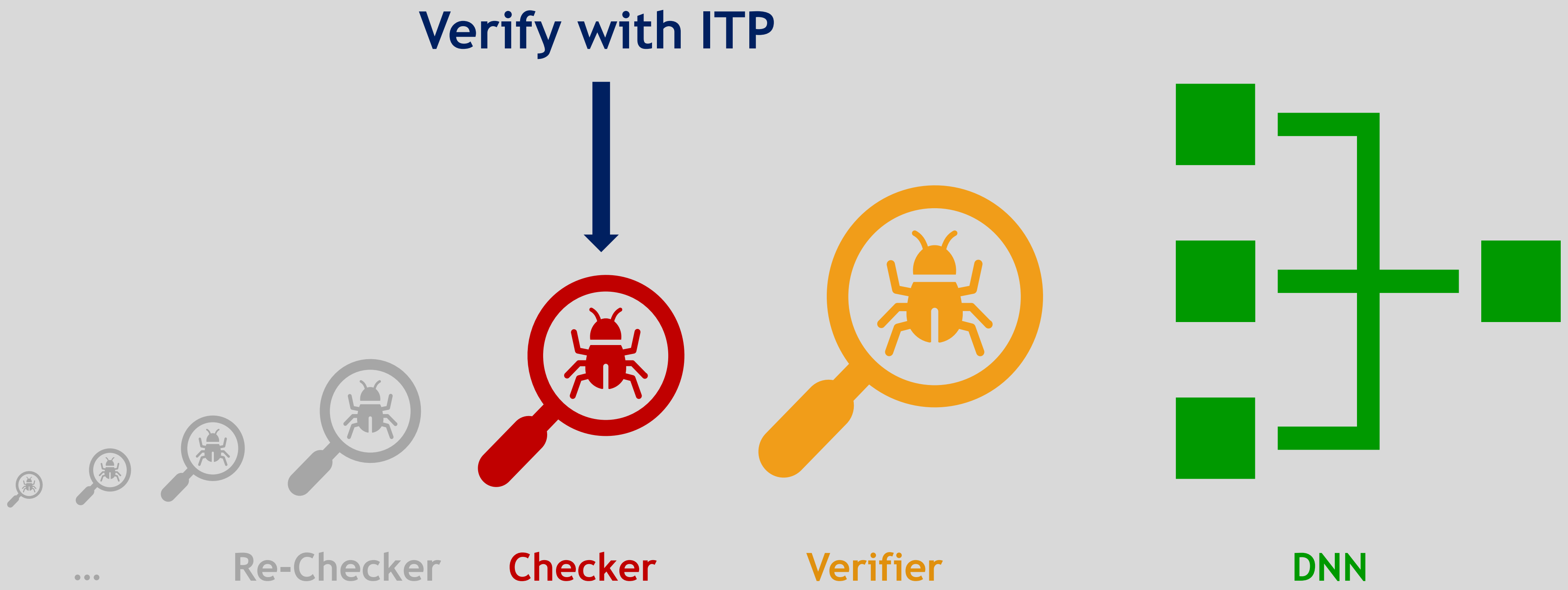
They are complex software and performance-oriented, thus their verification is very difficult.

Produce certificates of the verifiers' results:

- Checked using an independent, trusted and simple checker.
- Witness the correctness of the verifier or reveal errors.

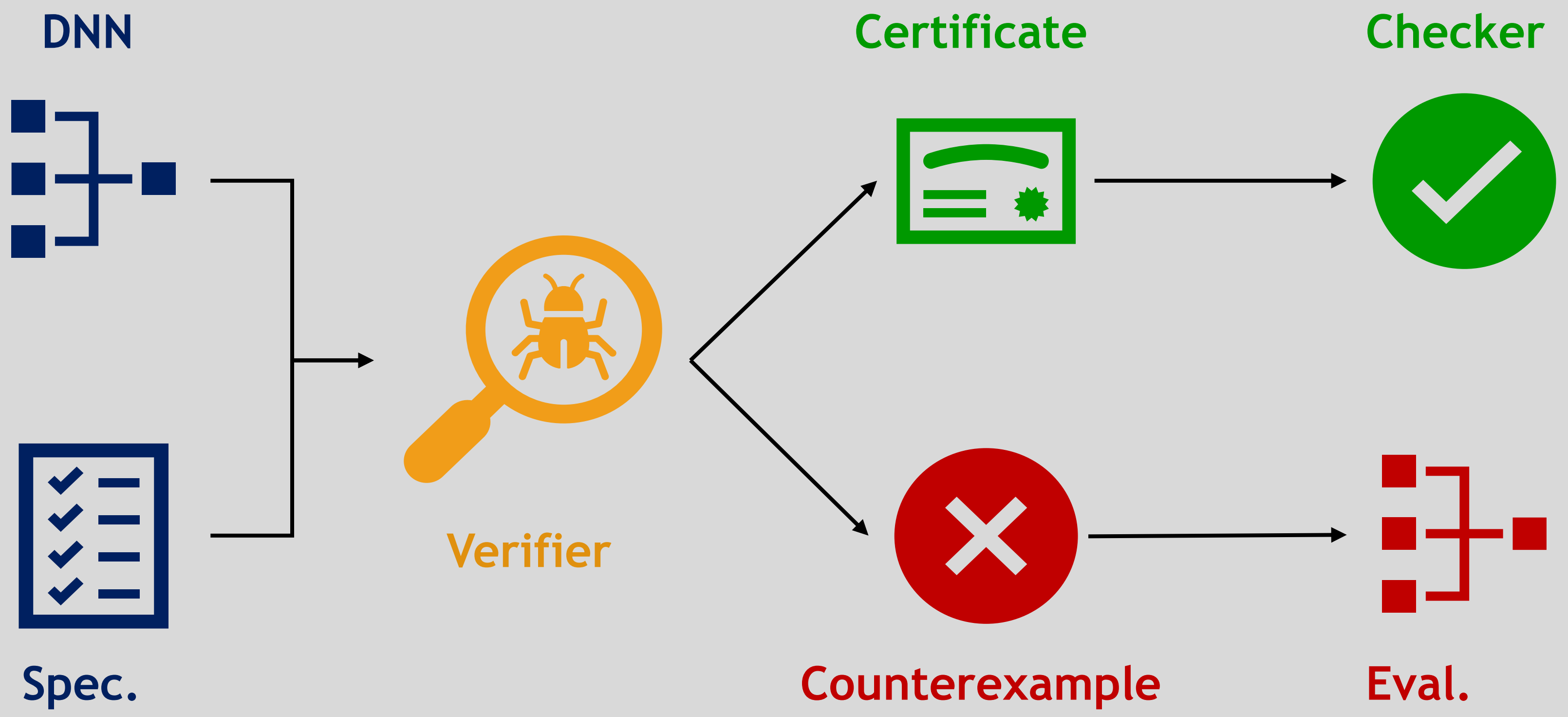


A common practice in SAT and SMT communities.

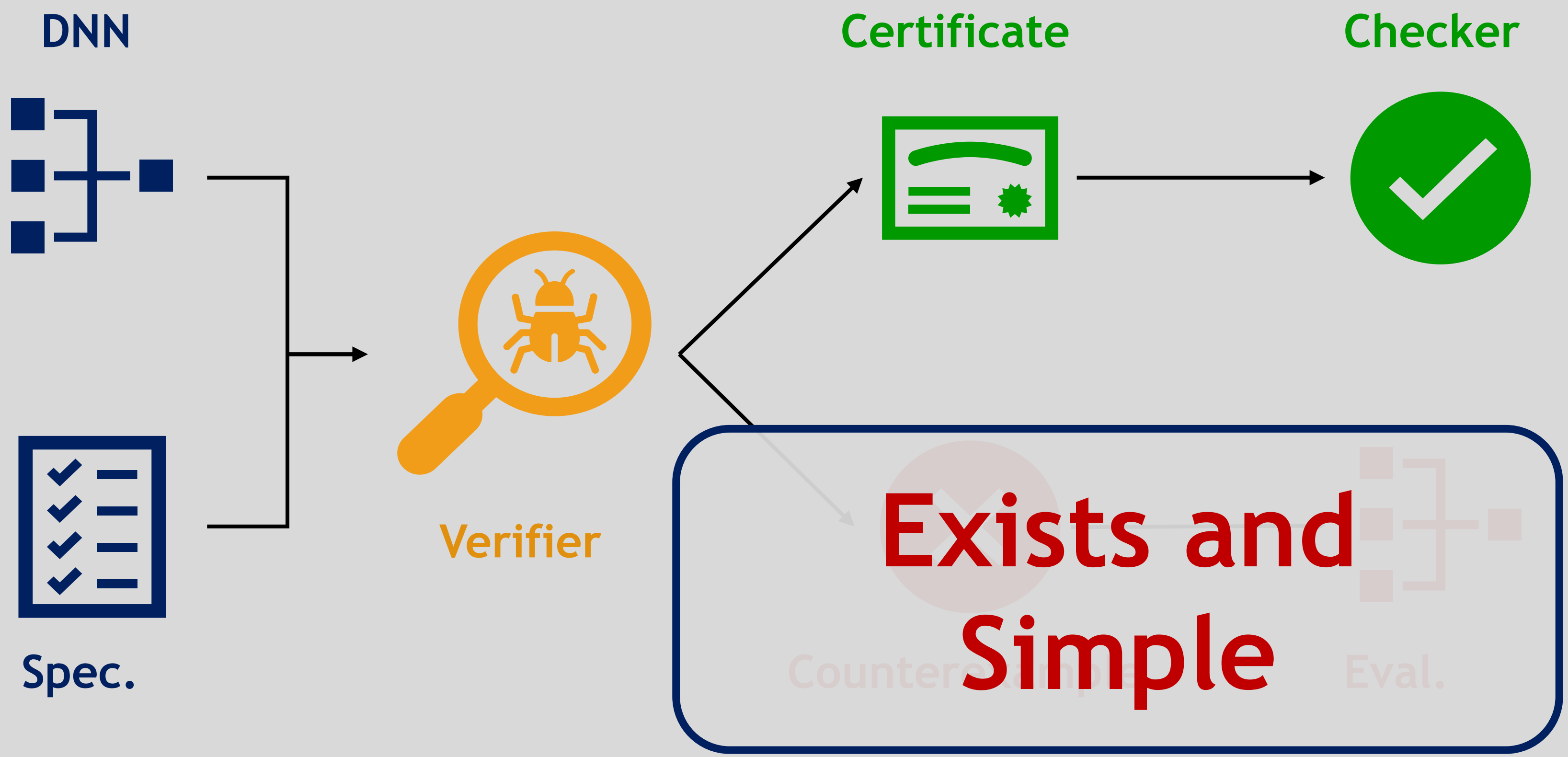


Trusting this scheme requires implementing a reliable checker, e.g., formally verified in an ITP.

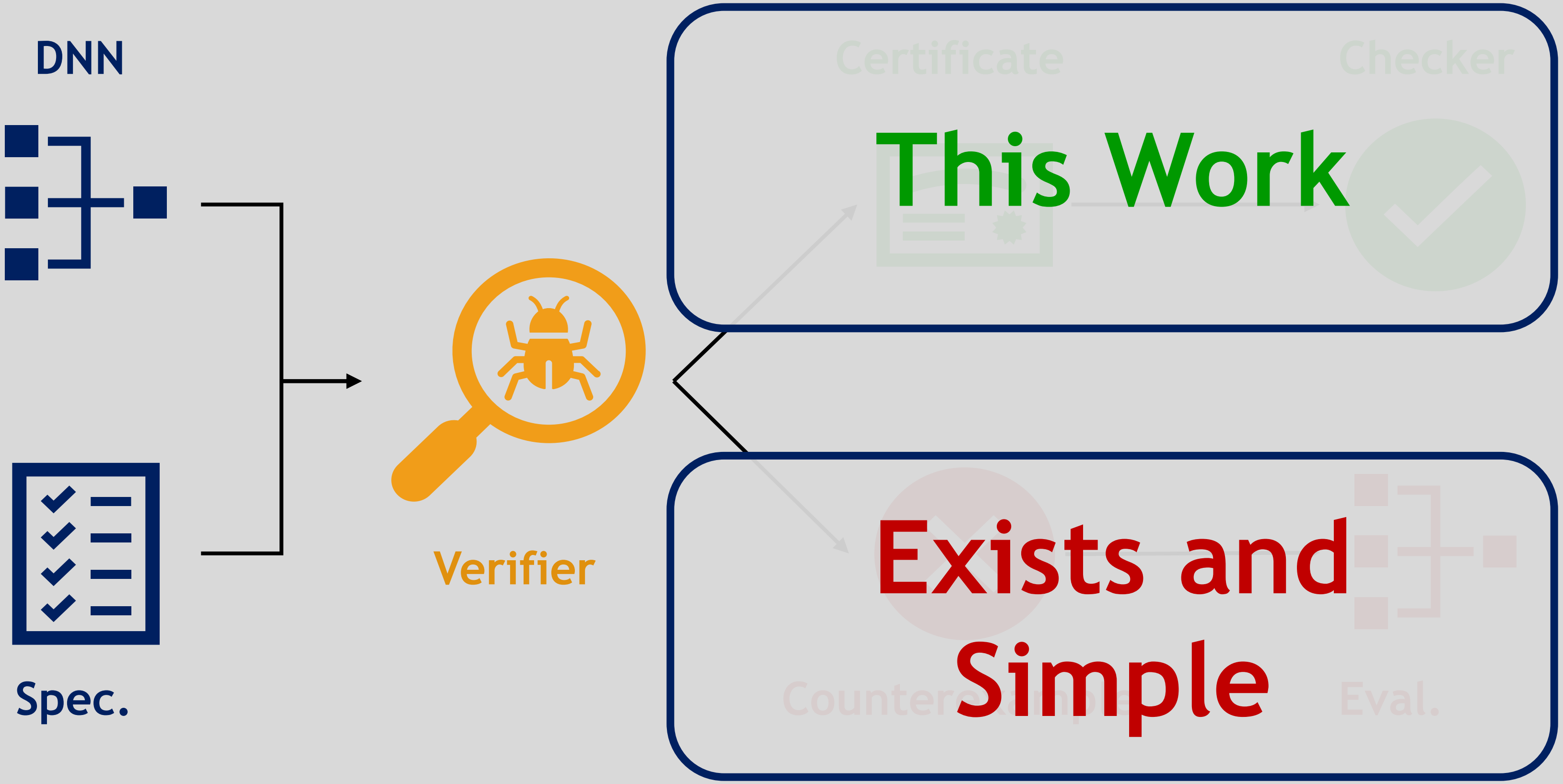
The Model of Trust



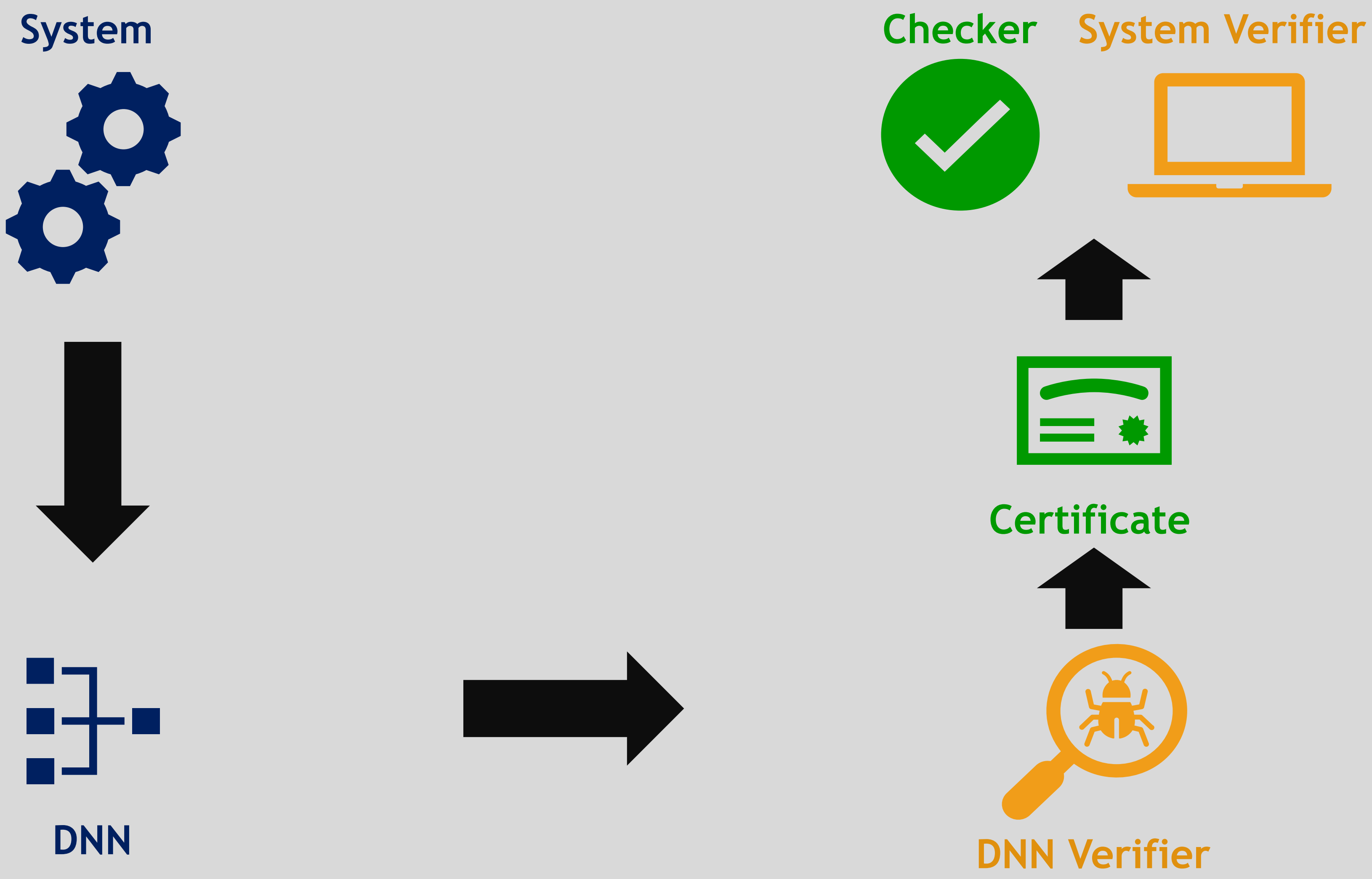
The Model of Trust



The Model of Trust



Integration in System Verification



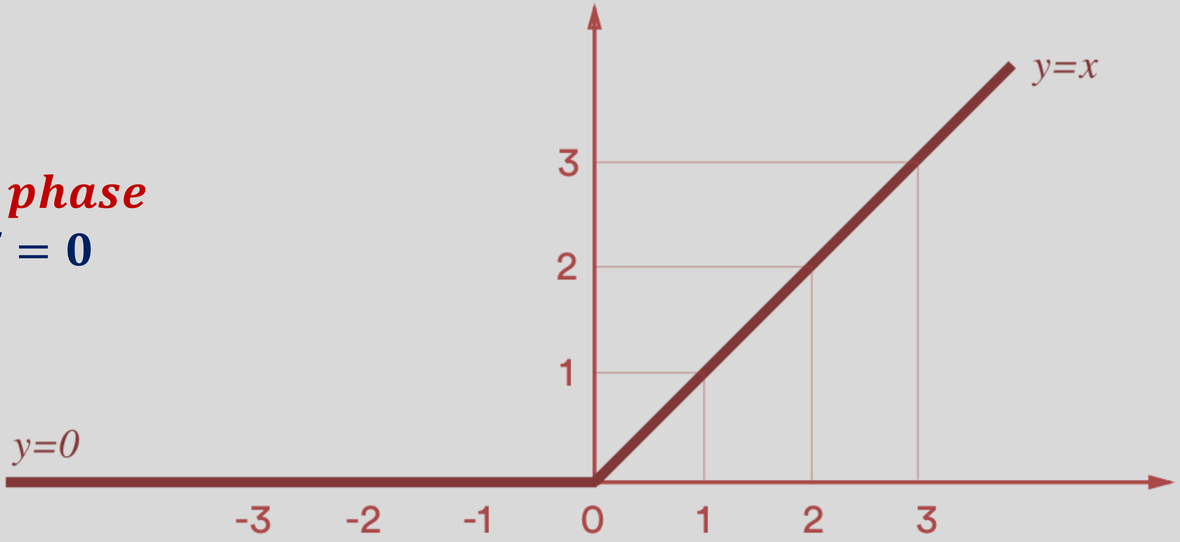
Background

Deep Neural Networks

- Layered function:
  - Affine: $b_i = \sum_j w_{i.j} \cdot f_j + c_i$
  - Nonlinear activations: $f_i = a_i(b_i)$

- The ReLU activation:

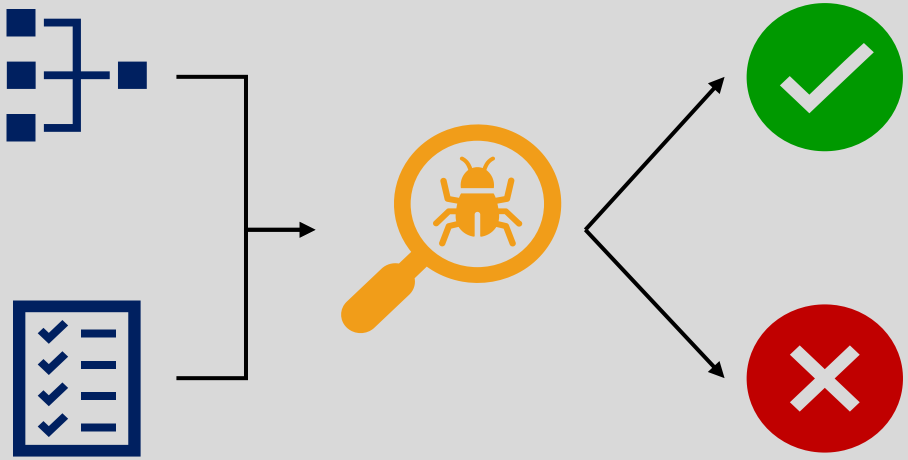
Inactive phase
$b \leq 0, f = 0$



Active phase
$b \geq 0, f = b$

DNN Verification

- Given a DNN $\mathcal{N}$ an input property $\varphi(x)$ and an output property $\psi(y)$, decide whether there exists an input vector x that violates $\varphi(x) \to \psi(\mathcal{N}(x))$.
  - If exists, the problem is satisfiable (`SAT`);
  - Otherwise unsatisfiable (`UNSAT`).



DNN Verification

- Assuming QF linear properties, and ReLU activations → linear + piecewise linear constraints:
- Find assignment for $\boldsymbol{V} \in \mathbb{R}^n$
  - Linear:

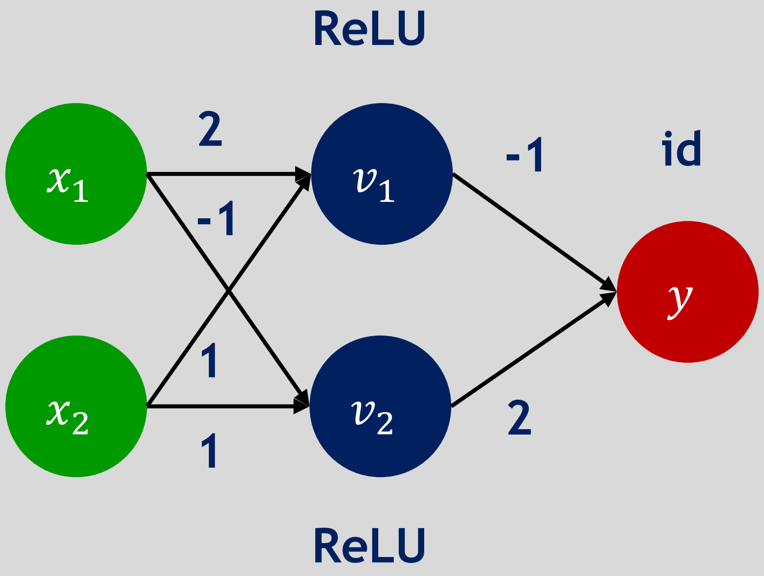
$$\boldsymbol{A} \cdot \boldsymbol{V} = \bar{\boldsymbol{0}} \;\; [\boldsymbol{A} \in \mathbb{R}^{m \times n}]$$

$$\boldsymbol{l} \le \boldsymbol{V} \le \boldsymbol{u} \;\; [\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n]$$

  - Piecewise Linear:

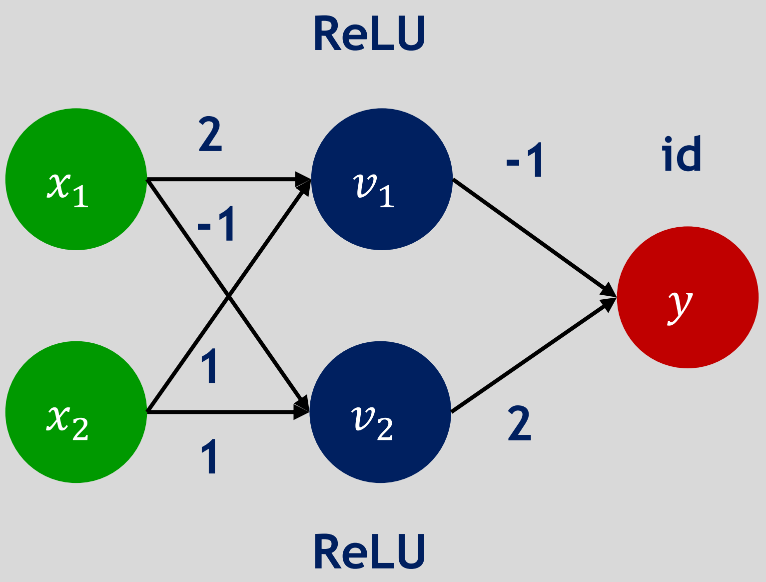
$$\boldsymbol{f_i} = \boldsymbol{ReLU}(\boldsymbol{b_i}) \; [\boldsymbol{f_i}, \boldsymbol{b_i} \in \boldsymbol{V}]$$

Verification Query Example



$$\varphi(x) := x \in [0, 1]^2$$
$$\psi(y) := y \in [4, 5]$$

Verification Query Example



$$\varphi(x) := x \in [0,1]^2$$
$$\psi(y) := y \in [4,5]$$

Variables:

$$V = [x_1 \quad x_2 \quad b_1 \quad b_2 \quad f_1 \quad f_2 \quad aux_1 \quad aux_2 \quad y]^\intercal$$

Tableau:

$$A = \begin{bmatrix} 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

Bounds:

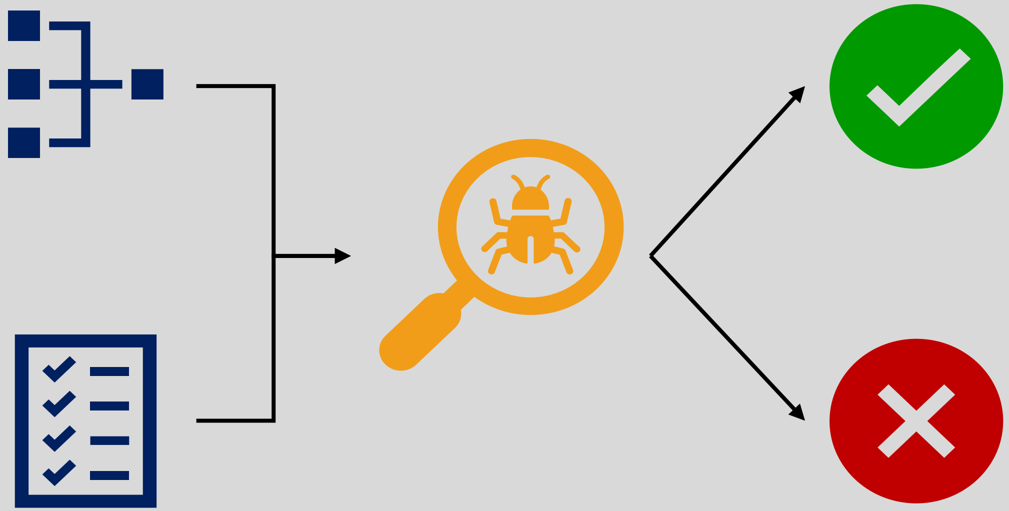
$$u = [1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2 \quad 5]^\intercal$$
$$l = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4]^\intercal$$

ReLUs:

$$f_1 = ReLU(b_1)$$
$$f_2 = ReLU(b_2)$$

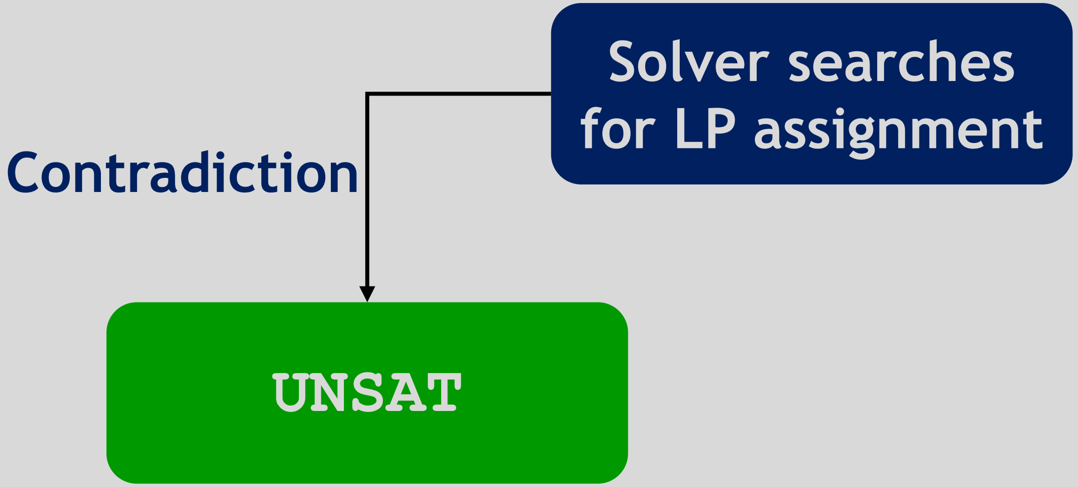
DNN Verification Algorithms

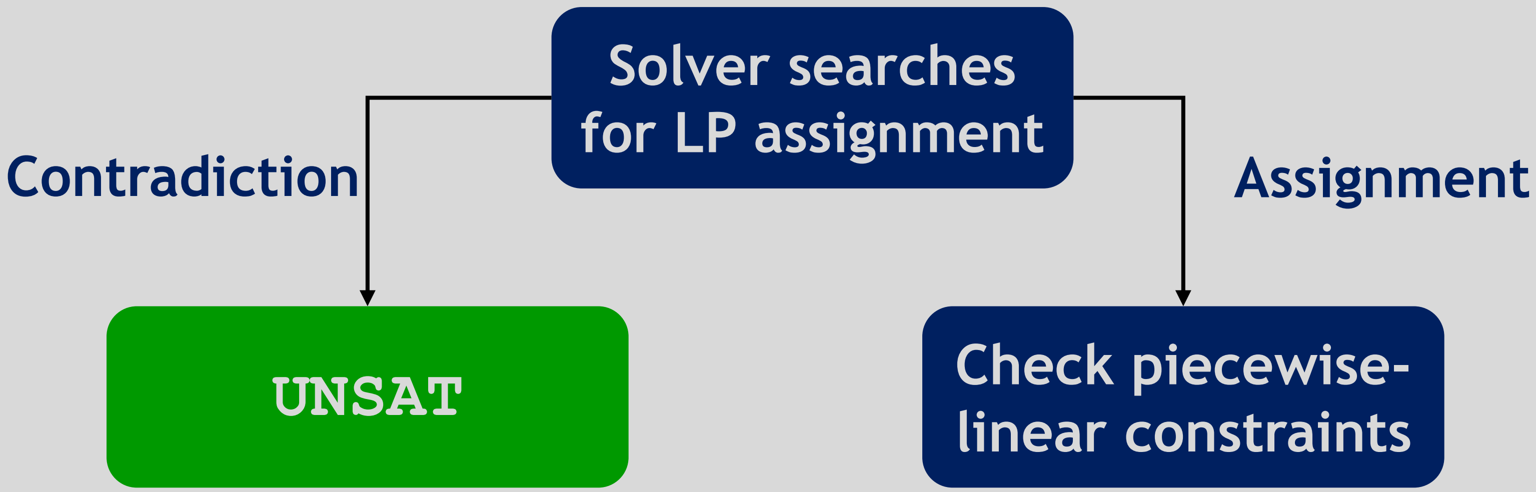
- Many verification algorithms **search for a satisfying assignment** with LP solvers + splitting approach.

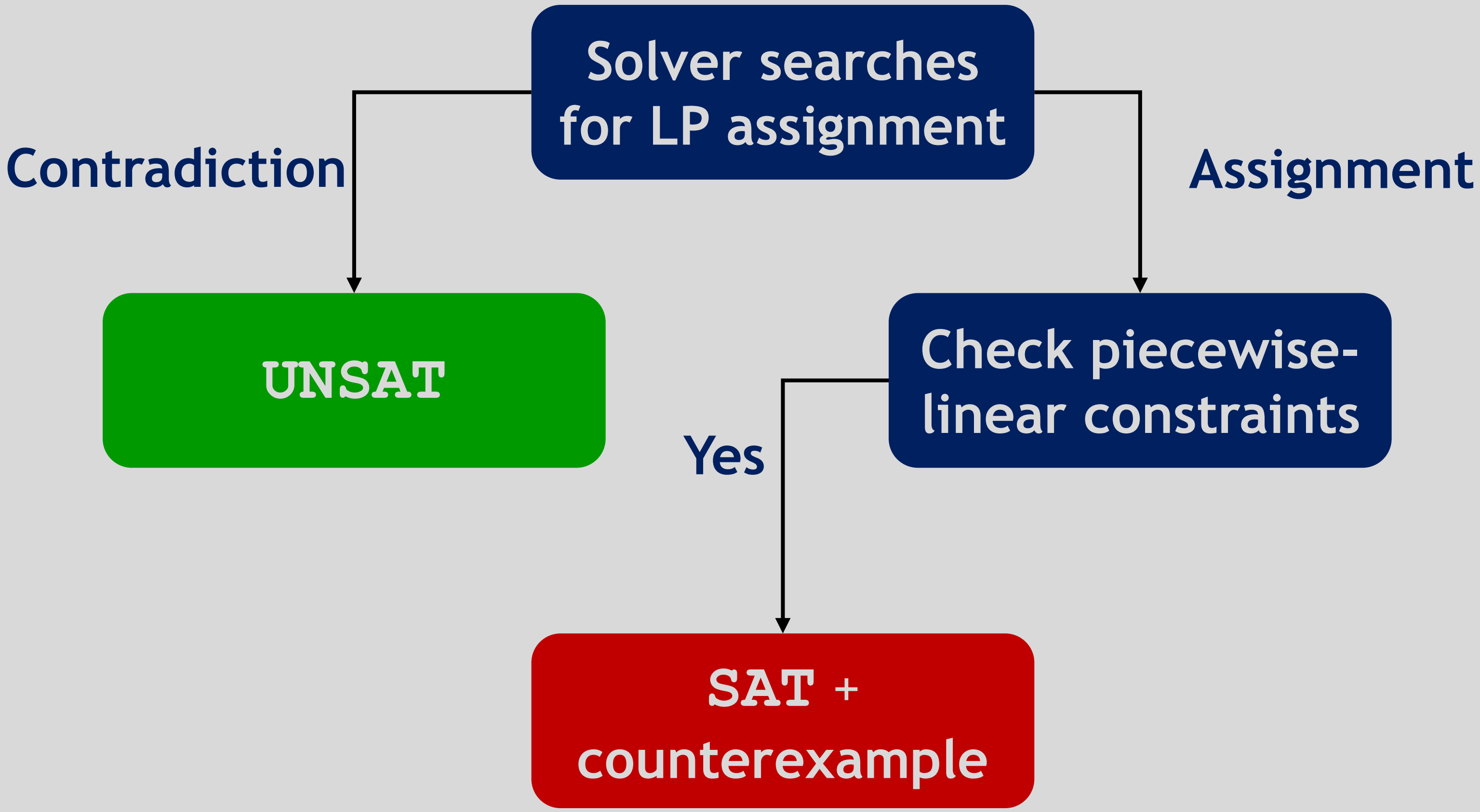


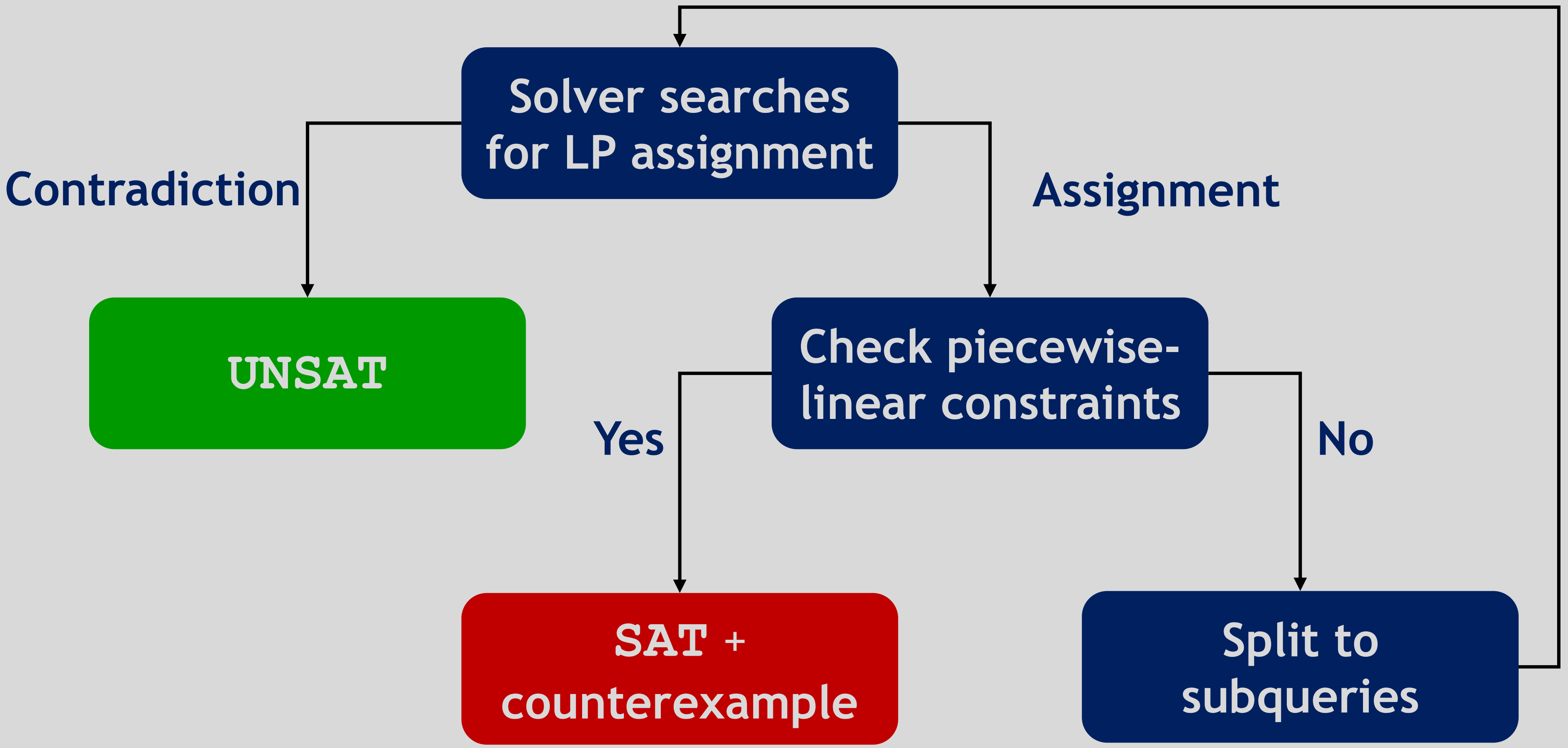
$$\boldsymbol{A \cdot V = \overline{0}}$$
$$\boldsymbol{l \leq V \leq u}$$
$$\boldsymbol{f_i = ReLU(b_i)}$$

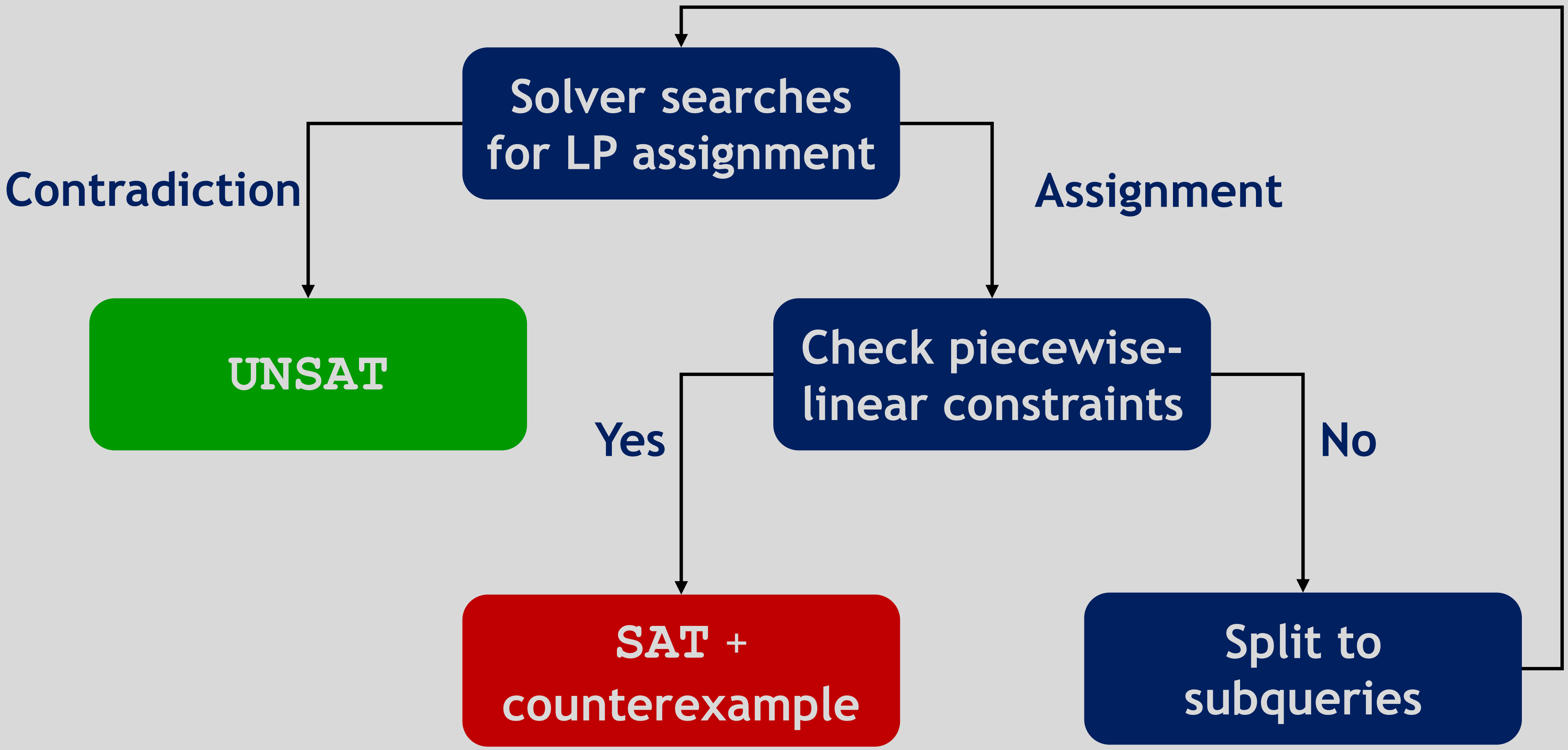
Solver searches for LP assignment


Solver searches for LP assignment
Contradiction
UNSAT


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check piecewise-linear constraints


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check piecewise-linear constraints
Yes
SAT + counterexample


Solver searches for LP assignment
Contradiction
Assignment
UNSAT
Check piecewise-linear constraints
Yes
No
SAT + counterexample
Split to subqueries



Splitting creates a tree structure to the search.

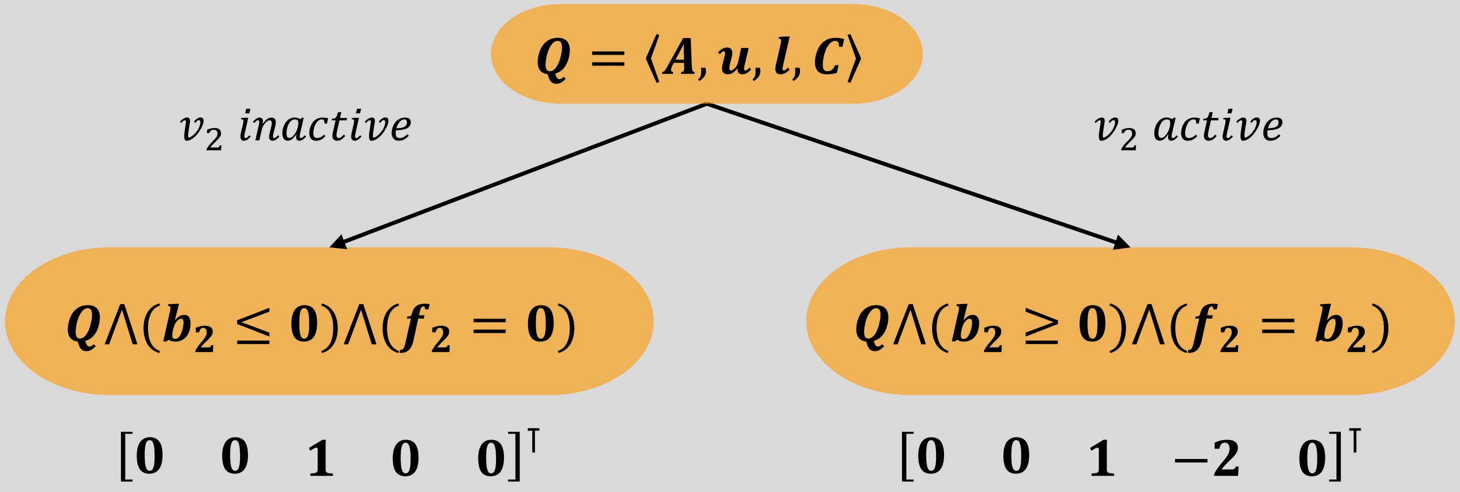
The query is UNSAT if and only if all leaves are UNSAT.

Certificate Production

- Certifying `SAT` is straightforward; Certifying `UNSAT` is more complicated, due to NP-Hardness [Katz et. al, ’17; Sälzer and Lange, ‘21].
- Marabou DNN verifier produces and checks `UNSAT` proofs (C++).

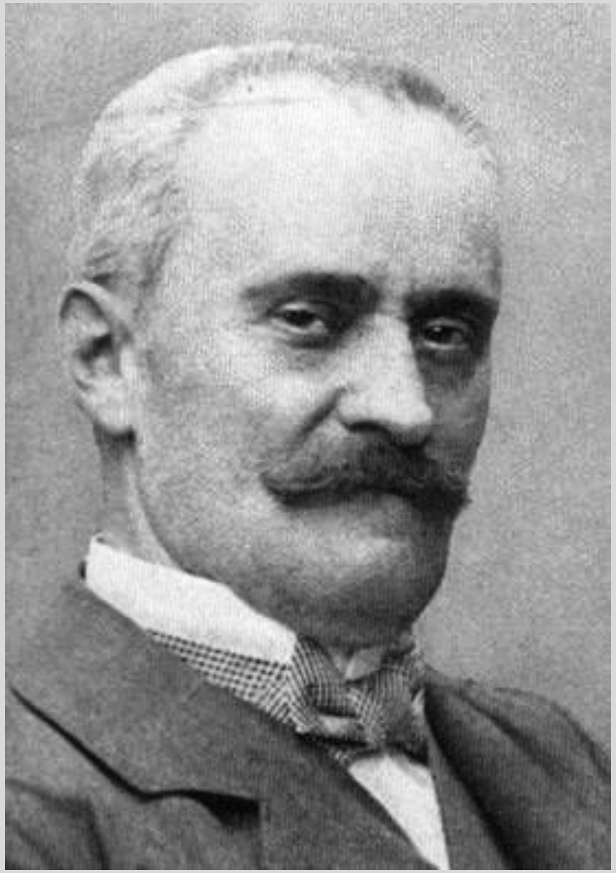
Neural Network Verification with Proof Production, Isac et. al, FMCAD 2022

The Certificate Tree



In the certificate-tree, nodes represent case-splits; leaves contain proofs of contradictions, which are represented by a vector.

The Farkas Lemma (Variant)

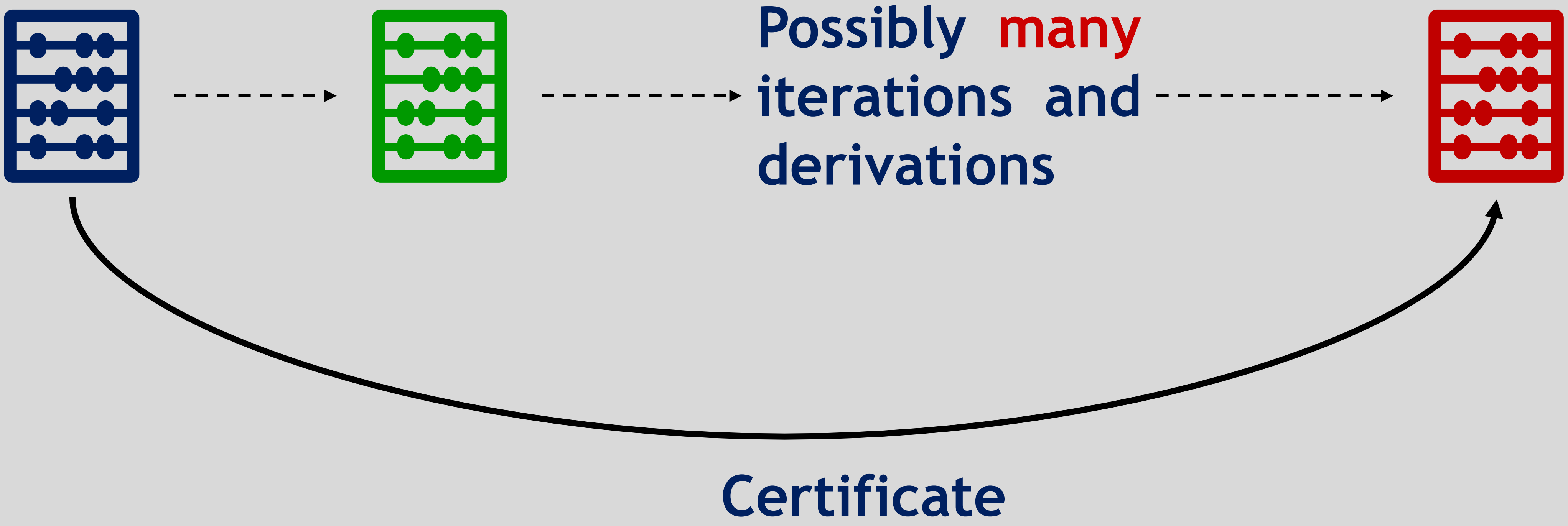


Gyula Farkas

For any set of linear equations $AV = 0$ where $l \leq V \leq u$ exactly one holds:

- There exists a solution $l \leq V \leq u$ s.t. $AV = 0$.
- There exists a refutation y s.t. $\forall l \leq V \leq u$: $y^{\top} AV < 0$.

Refutation can be constructed during execution of DNN verifier.


Possibly many iterations and derivations
Certificate

The Certificate Tree

$$\boldsymbol{Q = \langle A, u, l, C \rangle}$$

v_2 *inactive*

$$\boldsymbol{Q \wedge (b_2 \leq 0) \wedge (f_2 = 0)}$$

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{bmatrix}^\intercal$$

v_2 *active*

$$\boldsymbol{Q \wedge (b_2 \geq 0) \wedge (f_2 = b_2)}$$

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{-2} & \mathbf{0} \end{bmatrix}^\intercal$$

Certificate checking consists of traversing the tree, checking splits are covering and checking leaves using the Farkas lemma.

Certified Checking

Imandra

- An OCaml-based FPL and an ITP.
- Supports arbitrary precision real arithmetic.

```
theorem farkas_unsat (s : system)(x : var_vect) (c : certificate) =
     well_formed s x && check_cert s c
     ==>
     eval_system s x = false
[@@by [% use cert_is_neg s c x]
@> [% use solution_is_not_neg s c x]
@> auto] [@@fc]
```

Data Representation

$$Q = \langle A, u, l, C \rangle$$

$$Q \wedge (b_2 \le 0) \wedge (f_2 = 0) \qquad\qquad Q \wedge (b_2 \ge 0) \wedge (f_2 = b_2)$$

$$[0 \quad 0 \quad 1 \quad 0 \quad 0]^\intercal \qquad\qquad [0 \quad 0 \quad 1 \quad -2 \quad 0]^\intercal$$

Variables:

$$V = [x_1 \quad x_2 \quad b_1 \quad b_2 \quad f_1 \quad f_2 \quad aux_1 \quad aux_2 \quad y]$$

Tableau:

$$A = \begin{bmatrix} 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

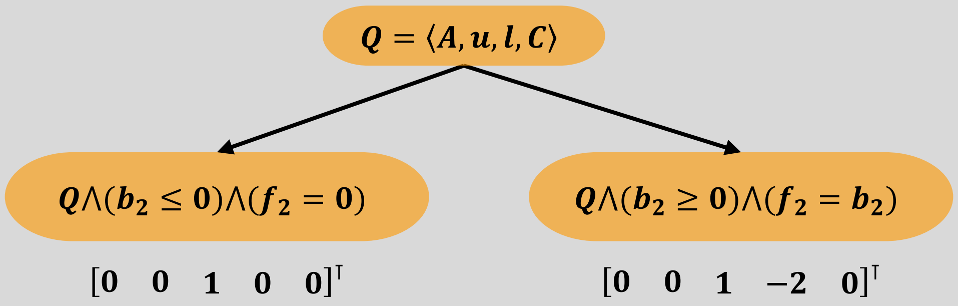
Bounds:

$$u = [1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2 \quad 5]$$
$$l = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4]$$

ReLUs:

$$f_1 = ReLU(b_1)$$
$$f_2 = ReLU(b_2)$$

Data Representation




```
type proofTree =
    | Leaf of real list
    | Node of Split * proofTree * proofTree
type Split = Relu of int * int * int
```

Variables:

$$V = [x_1 \quad x_2 \quad b_1 \quad b_2 \quad f_1 \quad f_2 \quad aux_1 \quad aux_2 \quad y]$$

Tableau:

$$A = \begin{bmatrix} 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

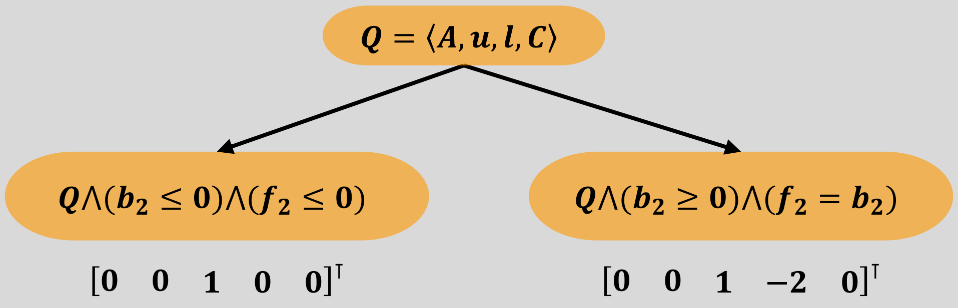
Bounds:

$$u = [1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2 \quad 5]$$
$$l = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4]$$

ReLUs:

$$f_1 = ReLU(b_1)$$
$$f_2 = ReLU(b_2)$$

Data Representation




```
type proofTree =
    | Leaf of real list
    | Node of Split * proofTree * proofTree
type Split = Relu of int * int * int
```

Variables:

$$V = [x_1 \quad x_2 \quad b_1 \quad b_2 \quad f_1 \quad f_2 \quad aux_1 \quad aux_2 \quad y]$$

Tableau:

$$A = \begin{bmatrix} 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

Bounds:

$$u = [1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2 \quad 5]$$
$$l = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4]$$

ReLUs:

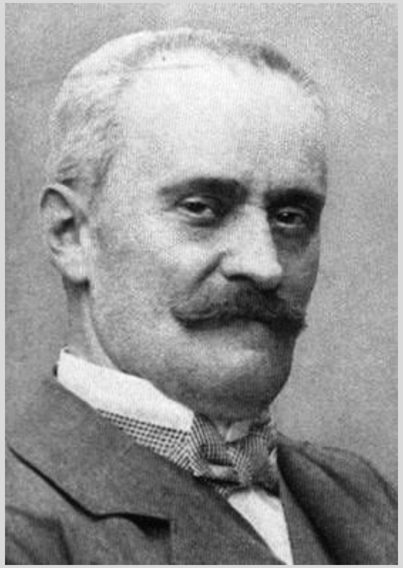
$$f_1 = ReLU(b_1)$$
$$f_2 = ReLU(b_2)$$

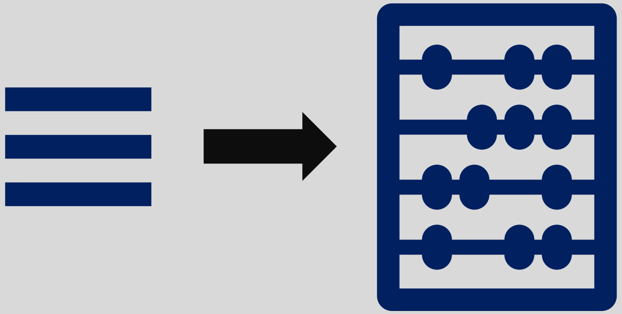
```
type poly = Real.t list
type expr =
   | Eq of poly
   | Geq of poly
type system = expr list
```

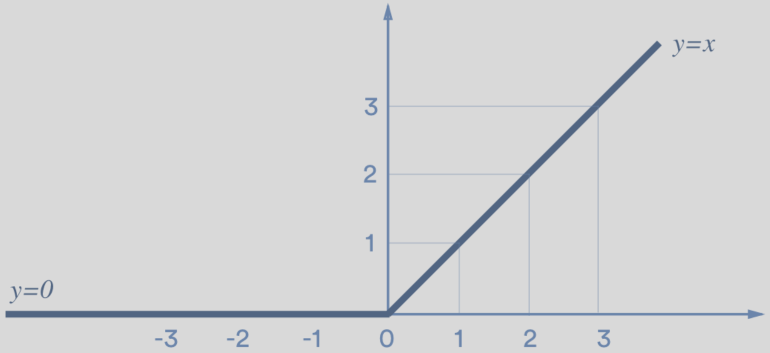
The Checking Function

```
let rec check_tree tableau upper_bounds lower_bounds constraints
proof_node =
match proof_node with
      | Proof_tree.Leaf contradiction ->
            check_contradiction … (* applies the Farkas lemma *)
      | Proof_tree.Node ( split, left, right ) ->
            let valid_split = … in
            let valid_children = … in (* includes recursive calls *)
            valid_split && valid_children
```

Proof Main Components





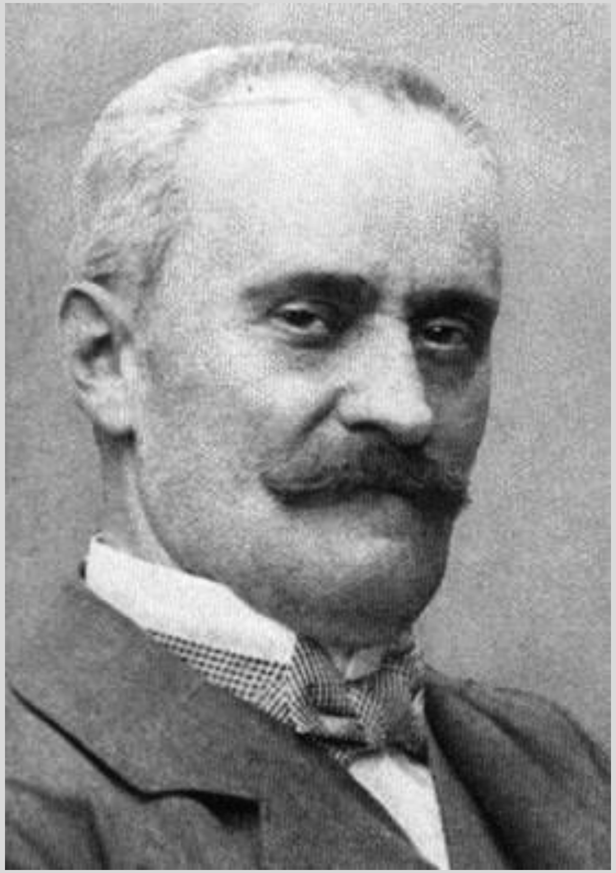


Farkas lemma (UNSAT direction)

Type reduction soundness

ReLU splits are covering

The Farkas Lemma (Generalized)

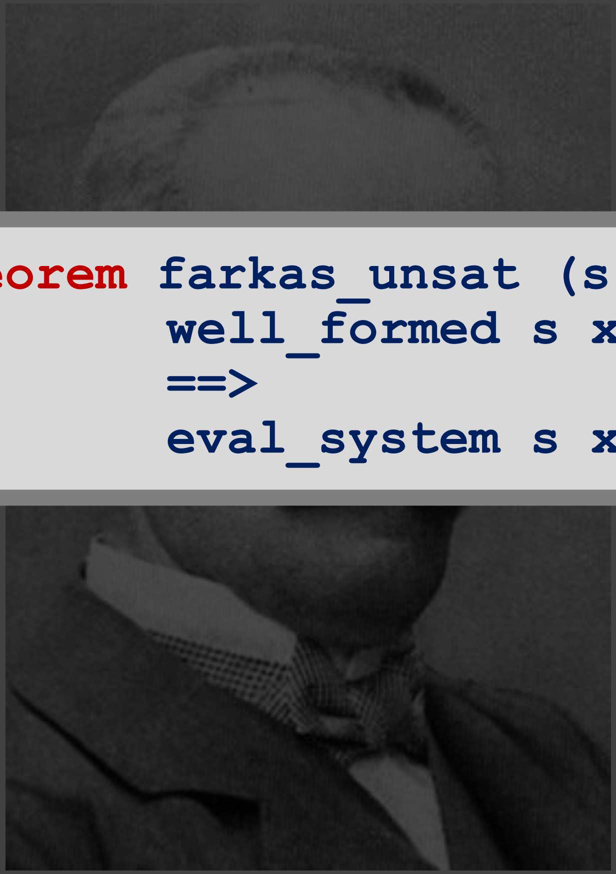


Gyula Farkas

Given a polynomial system $\wedge_{i=1}^{N}(p_i\,(x) = 0) \wedge \wedge_{i=1}^{K}(q_i\,(x) \geq\ 0)$ exactly one holds:

- There exists a solution.
- There exists a refutation: $\mathbb{I} \in \mathbb{R}^N, \mathbb{C} \in \mathbb{R}^K_{\geq 0}$ with $\mathbb{I} \cdot p_i + \mathbb{C} \cdot q_j < 0$.

The Farkas Lemma (Generalized)



Gyula Farkas

Given a polynomial system $\wedge_{i=1}^{N}(p_i\,(x) = 0) \wedge \wedge_{i=1}^{K}(q_i\,(x) \geq\ 0)$ exactly one holds:

```
theorem farkas_unsat (s : system)(x : var_vect) (c : certificate) =
        well_formed s x && check_cert s c
        ==>
        eval_system s x = false
```

Data Representation

Matrix and vectors

Variables:

$$V = [x_1 \quad x_2 \quad b_1 \quad b_2 \quad f_1 \quad f_2 \quad aux_1 \quad aux_2 \quad y]$$

Tableau:

$$A = \begin{bmatrix} 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}$$

Bounds:

$$u = [1 \quad 1 \quad 3 \quad 1 \quad 3 \quad 1 \quad 3 \quad 2 \quad 5]$$
$$l = [0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4]$$

`mk_system` →

System of polynomial equations and inequalities

Tableau:

$$2x_1 + x_2 - b_1 = 0$$

…

Bounds:

$$1 - x_1 \geq 0$$
$$x_1 \geq 0$$

…

If the system is `UNSAT`, so is the LP using the matrix and vectors.

Data Representation

Matrix and vectors

System of polynomial Equations and

```
lemma soundness_check_cert_composition tableau upper_bounds lower_bounds x =
    let sys = mk_system (mk_eq_constraints tableau) upper_bounds lower_bounds in
    well_formed_tableau_bounds tableau upper_bounds lower_bounds &&
    List.length x = List.length (List.hd tableau) &&
    not (eval_system sys x)
    ==>
    not (is_in_kernel tableau x) || not (bounded x upper_bounds lower_bounds)
```

…

If the system is `UNSAT`, so is are the input matrix and vectors.

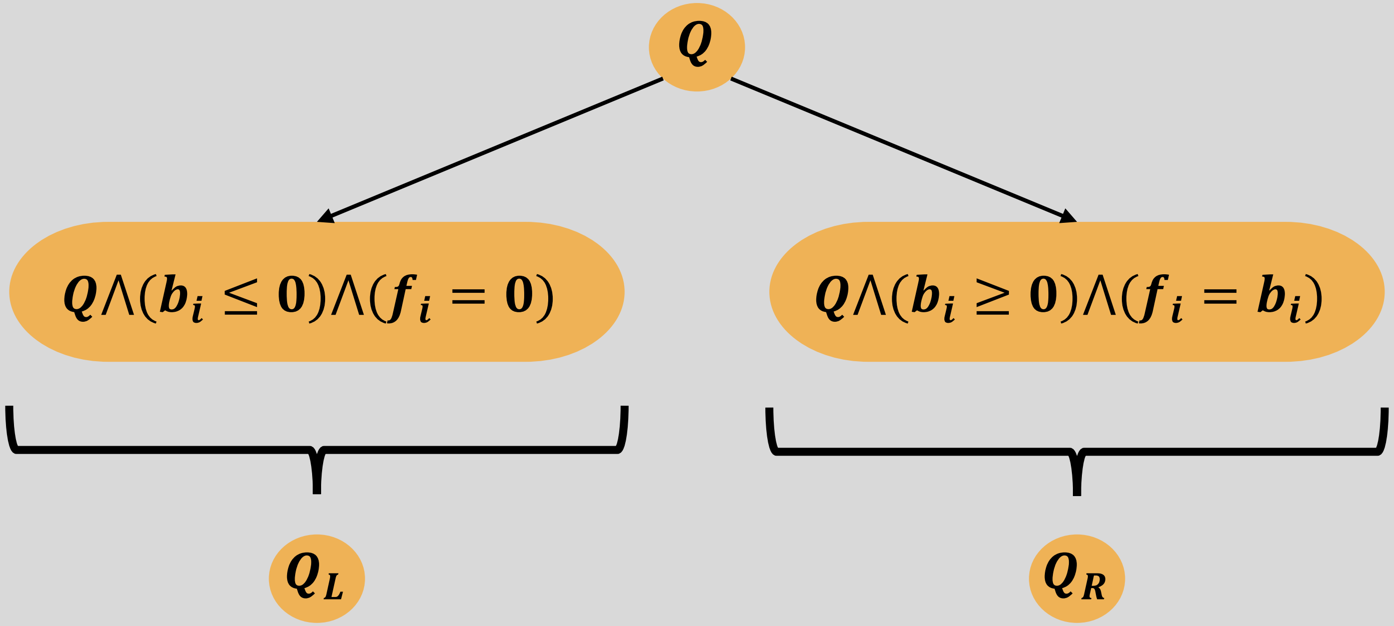
Soundness of Leaf Checking

If leaf checking returns true, then there is no solution to underlying linear query

Soundness of Leaf Checking

```
lemma soundness_leaf tableau upper_bounds lower_bounds relu_constraints x
proof_tree =
    match proof_tree with
    | Node _ -> true
    | Leaf contradiction ->
        well_formed_tableau_bounds tableau upper_bounds lower_bounds &&
        List.length x = List.length (List.hd tableau) &&
        check_tree (mk_eq_constraints tableau) upper_bounds lower_bounds
             relu_constraints proof_tree
        ==>
        unsat tableau upper_bounds lower_bounds relu_constraints x
```

Soundness of ReLU Splits



If the split corresponds to a ReLU constraint in Q, and both Q_L and Q_R are `UNSAT`, so is Q.

Soundness of ReLU Splits

```
theorem soundness_relu_split_matching tableau us ls constraints xs split =
    let (lb_left, ub_left), (lb_right, ub_right) = update_bounds_from_split ls
      us split in
    List.length xs = List.length ls
    && List.length xs = List.length us
    ==>
    match split with
    | ReluSplit (b,f,aux) ->
        List.mem (Relu (b,f,aux)) constraints
        && sat tableau us ls constraints xs
        ==>
        (sat tableau ub_left lb_left constraints xs
        || sat tableau ub_right lb_right constraints xs)
    | _ -> true
```

If the split corresponds to a ReLU constraint in Q, and both Q_L and Q_R are UNSAT, so is Q.

Main Result

```
let rec check_tree tableau upper_bounds lower_bounds constraints
proof_node =
match proof_node with
      | Proof_tree.Leaf contradiction ->
            check_contradiction …
      | Proof_tree.Node ( split, left, right ) ->
            let valid_split = … in
            let valid_children = … in
            valid_split && valid_children
```

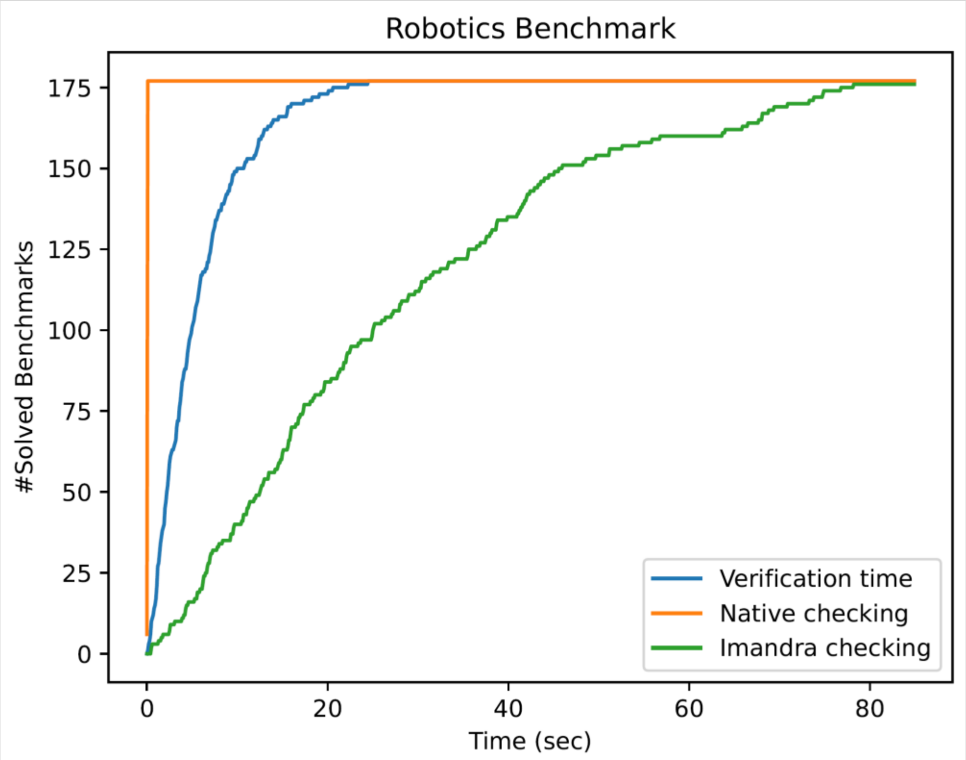
Main Result

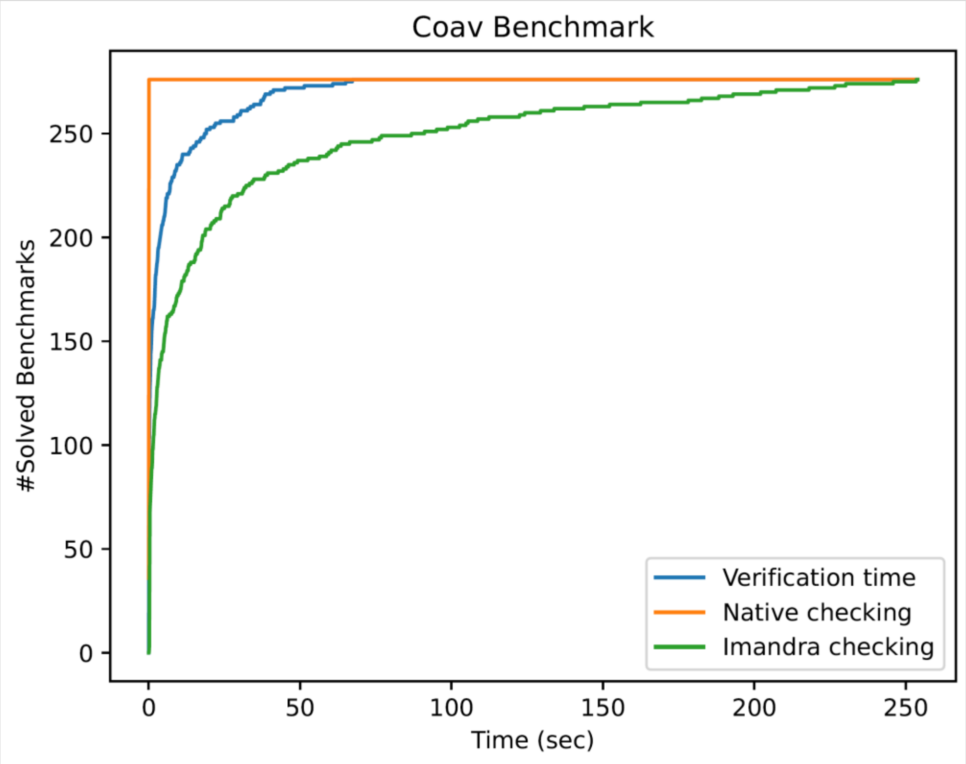
```
let rec check_tree tableau upper_bounds lower_bounds constraints
proof_node =
```

```
theorem check_tree_soundness (tableau: real list list) (upper_bounds: real list)
(lower_bounds: real list)
        (constraints: Constraint.t list) (tree: Proof_tree.t) (x: real list) =
    valid_proof tableau upper_bounds lower_bounds constraints tree
    && well_formed_vector tableau x
    ==>
    unsat tableau upper_bounds lower_bounds constraints x
```

```
            let valid_children = … in
            valid_split && valid_children
```

Performance Evaluation





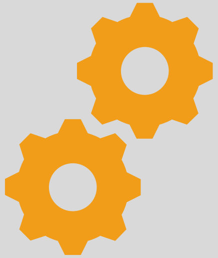
Evaluation shows a $\times \sim 4.7$ checking-to-verification delay, suggesting a clear tradeoff between reliability and scalability.

L.o.C. Evaluation

Result	Module name	L.O.C	Aux. Lem.	Library Dependencies (accumulating)
Poly Farkas lemma (**Theorem 5**)	farkas.iml	194	21	Imandra Standard Libraries: Real, List, Polynomials
Sound application of DNN polynomial Farkas lemma (**Theorem 7**)	well_formed_reduction.iml bound_reduction.iml tableau_reduction.iml	41 359 356	13	farkas.iml, certificate.iml, arithmetic.iml, util.iml, tightening.iml, constraint.iml, proof_tree.iml, checker.iml, bound_reduct_g.iml, mk_bound_poly.iml
Soundness of leaf checking (**Lemma 10**)	leaf_soundness.iml	145	40	sat.iml, split.iml bound_reduction.iml, well_formed_reduction.iml, tableau_reduction.iml
Single variable splits are covering (**Lemma 12**)	single_var_split_soundness.iml	81	10	
ReLU splits are covering (**Lemma 14**)	relu_split_soundness.iml relu_case_1_bounded.iml relu_case_2_bounded.iml	113 337 338	18	relu.iml
Soundness of node checking	node_soundness.iml	78	19	relu_split_soundness.iml, single_var_split_soundness.iml
Soundness (**Theorem 15**)	checker_soundness.iml	71	146	leaf_soundness.iml, node_soundness.iml
Total:		**2113**	**267**	

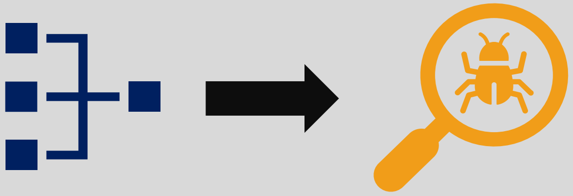
Table 2 Summary of the entire formalisation. The Table reads as follows: a result A is proven in module `A_mod`, which is N lines long, calls M auxiliary lemmas, and depends on libraries as listed.

Future Work



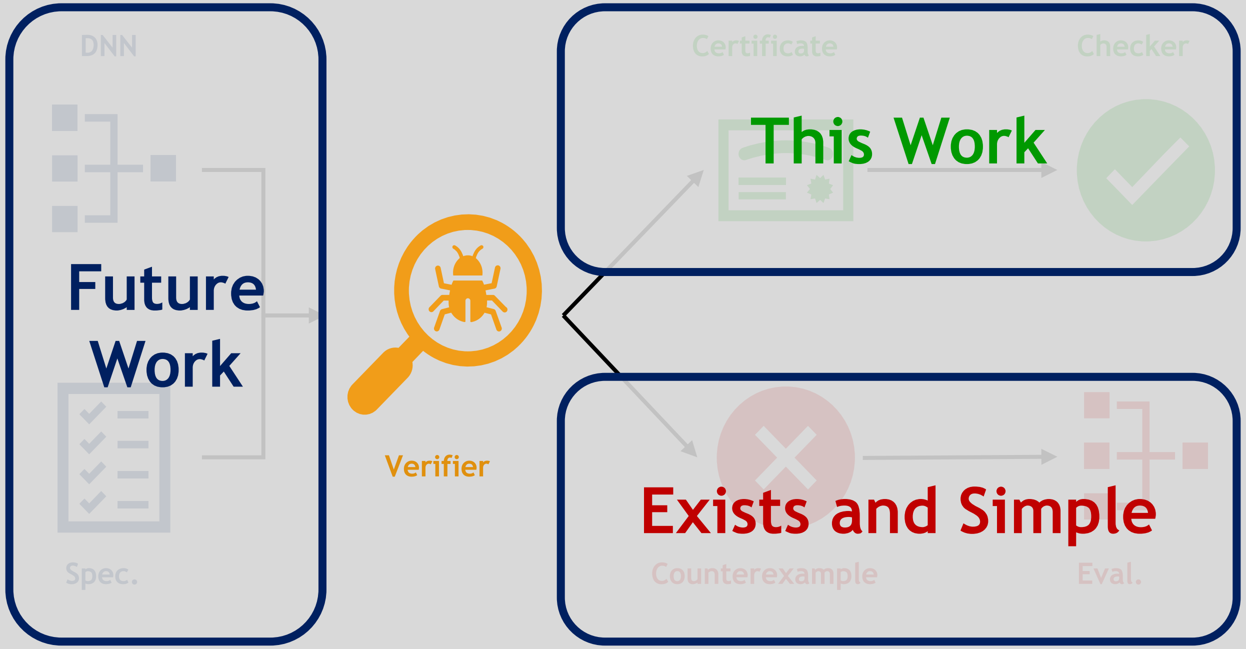
Integrate in a system verifier

Support certificate optimizations



Bridge gap between DNN and query

Conclusion


DNN
Future Work
Spec.
Verifier
Certificate
Checker
This Work
Exists and Simple
Counterexample
Eval.

Feel free to contact:
omri.isac@mail.huji.ac.il

paper + code available