Lecture 9: Verifying Robustness

Trustworthy Machine Learning
Spring 2024
Formal Methods for Verified Robustness

- Last lecture:
  - Formalizing program verification: Pre/post conditions
  - Verification as constraint solving
  - Robustness checking as program verification

- Today:
  - Specialized constraint solver ReluPlex for neural network verification
  - Verifying robustness by abstract interpretation (box domain)

Slides credit: Gagandeep Singh and Madhu Parthasarathy (UIUC)
Specifications over DNNs

Precondition

\[ \forall x_1, x_2, l_1 \leq x_1 \leq u_1, l_2 \leq x_2 \leq u_2 \]

DNN \( f \)

\[
\text{def } f(x_1, x_2): \\
x_3 = w_{13} \cdot x_1 + w_{23} \cdot x_2 + b_3 \\
x_4 = w_{14} \cdot x_1 + w_{24} \cdot x_2 + b_4 \\
x_5 = \max(0, x_3) \\
x_6 = \max(0, x_4) \\
x_7 = w_{57} \cdot x_5 + w_{67} \cdot x_6 + b_7 \\
x_8 = w_{56} \cdot x_5 + w_{68} \cdot x_6 + b_8 \\
\text{return } x_7, x_8
\]

Postcondition

\[ x_7 > x_8 \]

Either prove that the network output satisfies the postcondition for all inputs in the pre-condition or find a counterexample.
Robustness against adversarial perturbations

Network correctly classifies $I_0$ as “car”

$L_\infty$-ball around $I_0$ of radius $\epsilon$

$0.6 \leq x_0 \leq 0.65$

$0.55 \leq x_1 \leq 0.6$

......

$\phi$ with $\epsilon = 8/255$

$\psi$: network classifies image as “car”
Verification of Neural Networks

<table>
<thead>
<tr>
<th>Incomplete</th>
<th>Abstract interpretation: Box, Zonotope, DeepPoly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete</td>
<td>Mixed Integer Linear Programming (MILP)</td>
</tr>
<tr>
<td></td>
<td>SMT solvers (Reluplex)</td>
</tr>
</tbody>
</table>

Active area of research with annual competition: VNNComp
Current winner: alpha-beta crown
Reluplex: An SMT based approach
International Conference on Computer Aided Verification (CAV), 2017.
The Constraint Satisfaction Problem

Set of variables $V$
Atomic predicate:
• Linear inequality of the form $p = (\sum_{v_i \in V} a_i v_i \leq c_i)$
• ReLU equation of the form $p = (v_i = \text{ReLU}(v_j))$

Given a conjunction of atomic predicates $\varphi = p_1 \land \cdots \land p_t$ decide if $\varphi$ is satisfiable

• $\varphi_N$ : conjunction of atomic predicates gives relation between input, output, and hidden variables of $N$.
• Pre-condition is given by a conjunction of linear inequalities $I = \{x \mid x \models \varphi_I\}$ and post-condition is a disjunction of linear (strict) inequalities $F = \{z \mid z \models \varphi_F\}$
• Sufficient to check if $\varphi_N \land \varphi_I \land \neg \varphi_F$ is satisfiable!
The Simplex Algorithm

- Solves satisfiability of conjunction of linear inequalities

\[ \varphi = v_1 + v_2 \leq -5 \land v_1 - v_2 \geq 3 \]

- Step 1: Construct an initial configuration

\[ V = \{v_1, v_2, v_3, v_4\} \]

- Set of basic variable \( B \subseteq V \) Initially it is all new variables

- Tableau \( T \) contains one equation per basic variable; RHS only has non-basic variables

\[ u: V \to \mathbb{R} \cup \{\infty\} \]
\[ l: V \to \mathbb{R} \cup \{-\infty\} \]

Upper and lower bounds on variable

Valuation satisfies \( T \)

\[ u(v_3) = -5 \text{ and } l(v_4) = 3 \]
\[ \alpha(v_i) = 0 \text{ for all } v_i \in V \]

Add one new variable for each inequality

Goal of the algorithm: Update current values of non-basic variables to meet all lower/upper bounds
Simplex provides rules of the following kind for modifying a configuration $C = (V, B, T, u, l, \alpha)$

Apply rules until a SAT or UNSAT is derived

Some condition on $C$

\[ SAT \]

Some condition on $C$

\[ UNSAT \]

Some condition on $C$

Modified configuration $C'$

C<sub>0</sub> $\xrightarrow{R_1}$ C<sub>1</sub> $\xrightarrow{R_2}$ \cdots $\xrightarrow{R_h}$ C<sub>h</sub> $\xrightarrow{\text{SAT or UNSAT}}$
Slack Variables

\[ \text{slack}^+(v_i) = \{ v_j \notin B \mid (T_{i,j} > 0 \land \alpha(v_j) < u(v_j)) \lor (T_{i,j} < 0 \land \alpha(v_j) > l(v_j)) \} \]

\[ \text{slack}^-(v_i) = \{ v_j \notin B \mid (T_{i,j} < 0 \land \alpha(v_j) < u(v_j)) \lor (T_{i,j} > 0 \land \alpha(v_j) > l(v_j)) \} \]

\text{slack}^+(v_i) \) : Variables on RHS defining \( v_i \) whose values can be changed to \textbf{increase} the value of \( v_i \)

\( \text{e.g. variable } v_j \text{ has positive coefficient and its current value is less than its upper bound} \)

\text{slack}^-(v_i) \) : Variables on RHS defining \( v_i \) whose values can be changed to \textbf{decrease} the value of \( v_i \)
Slack Variables

\[
\text{slack}^+(v_i) = \{v_j \notin B \mid (T_{i,j} > 0 \land \alpha(v_j) < u(v_j)) \lor (T_{i,j} < 0 \land \alpha(v_j) > l(v_j))\}
\]

\[
\text{slack}^-(v_i) = \{v_j \notin B \mid (T_{i,j} < 0 \land \alpha(v_j) < u(v_j)) \lor (T_{i,j} > 0 \land \alpha(v_j) > l(v_j))\}
\]

\[
V = \{v_1, v_2, v_3, v_4\}
\]

- \(B = \{v_3, v_4\}\)
- \(T: v_3 = v_1 + v_2\) and \(v_4 = v_1 - v_2\)
- \(u(v_3) = -5\) and \(l(v_4) = 3\)
- \(\alpha(v_i) = 0\) for all \(v_i \in V\)

If we have \(u(v_1) = 0\) then

\[
\text{slack}^+(v_3) = \{v_1, v_2\}
\]

\[
\text{slack}^+(v_1) = \emptyset
\]
Simplex Rule: Successful Termination

\[ \forall v_i \in V, \quad l(v_i) \leq a(v_i) \leq u(v_i) \]

SAT \hspace{1cm} Success

Current valuation meets all lower/upper bounds: satisfying assignment found
Simplex Rule: Unsuccessful Termination

\[ v_i \in B, \quad (\alpha(v_i) < l(v_i) \land \text{slack}^+(v_i) = \emptyset) \lor (\alpha(v_i) > u(v_i) \land \text{slack}^-(v_i) = \emptyset) \quad \text{UNSAT} \]

Failure

There is a basic variable for which current value must be increased/decreased to meet lower/upper bound constraint but no such update of RHS vars is possible.
Simplex Rule: Pivot

If: A basic variable’s value needs to be increased/decreased to meet lower/upper bound and there is a possible variable on RHS whose value be changed for this purpose
Then make it non-basic by swapping their roles using pivot

\[
\begin{align*}
T &:= \text{pivot}(T, v_i, v_e), & B &:= B \cup \{v_e\} \backslash \{v_i\} \\
T &:= \text{pivot}(T, v_i, v_e), & B &:= B \cup \{v_i\} \backslash \{v_e\}
\end{align*}
\]
Pivot Example

**Pivot:** Allows replacing basic variable with a non-basic variable

\[
T = \begin{align*}
    v_3 &= v_1 + v_2 \\
    v_4 &= v_1 - v_2
\end{align*}
\]

pivot \((T, 3, 1)\)

\[
\begin{align*}
    v_1 &= v_3 - v_2 \\
    v_4 &= (v_3 - v_2) - v_2
\end{align*}
\]

\[= T'\]
Simplex Rule: Update

For a non-basic variable, if its current value is less/greater than lower/upper bound then increase/decrease it to meet the bound

\[
v_j \notin B, \quad \alpha(v_j) < l(v_j) \lor \alpha(v_j) > u(v_j), \quad l(v_j) \leq \alpha(v_j) + \delta \leq u(v_j) \quad \Rightarrow \quad \alpha := \text{update}(\alpha, v_j, \delta)
\]
Update Example

**Update**: Allows updating value of a non-basic variable

\[
\begin{align*}
\alpha: & \\
\text{update}(v_3, -5): & \\
\alpha': & \\
\end{align*}
\]

\[
\begin{align*}
v_1 &= v_2 = v_3 = v_4 = 0 \\
v_1 &= v_3 - v_2 \\
v_4 &= v_3 - 2v_2 \\
T &= T' \\
v_1 &= v_3 = v_4 = -5 \\
v_2 &= 0
\end{align*}
\]
Simplex Algorithm in SMT Solver

- Starting with initial configuration, keep applying pivot/update rules until the termination condition holds

- Multiple rules may be applicable in a given configuration
  - Derivation tree captures all possible branches of rule applications

- Key engineering details of implementation
  - Which rule to choose in a given configuration
  - Choice of rules affects time to termination but does not require backtracking
  - How to apply rules efficiently (e.g. keeping track of slack variables)
Soundness and Completeness

**SOUNDNESS**: If there is a derivation to SAT (or UNSAT), $\varphi$ is satisfiable (or not).

**COMPLETENESS**: There is always a derivation to either SAT or UNSAT.
## Constraints in Neural network Verification

### Precondition

\[ \forall x_1, x_2, l_1 \leq x_1 \leq u_1, l_2 \leq x_2 \leq u_2 \]

### DNN \( f \)

```python
def f(x_1, x_2):
    x_3 = w_{13} \cdot x_1 + w_{23} \cdot x_2 + b_3
    x_4 = w_{14} \cdot x_1 + w_{24} \cdot x_2 + b_4
    x_5 = \text{ReLU}(x_3) = \max(0, x_3)
    x_6 = \text{ReLU}(x_4) = \max(0, x_4)
    x_7 = w_{57} \cdot x_5 + w_{67} \cdot x_6 + b_7
    x_8 = w_{56} \cdot x_5 + w_{68} \cdot x_6 + b_8
    \text{return } x_7, x_8```

### Postcondition

\[ x_7 > x_8 \]
Simplex to Reluplex

• Solves satisfiability of conjunction of linear inequalities and ReLU equations.

\[ \varphi = v_1 + v_2 \leq -5 \land v_1 - v_2 \geq 3 \land v_1 = ReLU(v_2) \]

• Step 1: Construct an initial configuration \( C = (V, B, T, u, l, \alpha, R) \)

\[ V = \{v_1, v_2, v_3, v_4\} \]

• \( B = \{v_3, v_4\} \)
• \( T : v_3 = v_1 + v_2 \) and \( v_4 = v_1 - v_2 \)
• \( u(v_3) = -5 \) and \( l(v_4) = 3 \)
• \( \alpha(v_i) = 0 \) for all \( v_i \in V \)
• \( R = \{(v_1 = ReLU(v_2))\} \)

Set of ReLU constraints \( R \)
Modified Successful Termination Test

\[ \forall v_i \in V. \; l(v_i) \leq \alpha(v_i) \leq u(v_i), \quad \forall (v_f = \text{ReLU}(v_b)) \in R. \; \alpha(v_f) = \text{ReLU}(\alpha(v_b)) \]

Current valuation meets all lower/upper bounds and all ReLU constraints
Additional Pivot Rule

\[
v_i \in B, \quad \exists v_j. (v_j = \text{ReLU}(v_i)) \in R \lor (v_i = \text{ReLU}(v_j)) \in R, \quad v_e \notin B, T_{i,e} \neq 0
\]

\[
T := \text{pivot}(T, v_i, v_e), \quad B := B \cup \{v_e\} \setminus \{v_i\}
\]

ReluPivot

If a basic variable \( v_i \) is involved in a ReLU constraint,
Then swap it’s a role with a non-basic variable \( v_e \) in its RHS with non-zero coefficient using pivot
Additional Update Rules

\[
\frac{v_j \notin B, \quad (v_j = \text{ReLU}(v_i)) \in R, \quad \alpha(v_j) \neq \text{ReLU}(\alpha(v_i))}{\alpha := \text{update}(\alpha, v_j, \text{ReLU}(\alpha(v_i)) - \alpha(v_j))} \quad \text{Update}_f
\]

\[
\frac{v_i \notin B, \quad (v_j = \text{ReLU}(v_i)) \in R, \quad \alpha(v_j) \neq \text{ReLU}(\alpha(v_i)), \quad \alpha(v_j) \geq 0}{\alpha := \text{update}(\alpha, v_i, \alpha(v_j) - \alpha(v_i))} \quad \text{Update}_b
\]

If a non-basic variable \( v_i \) is involved in a ReLU constraint that’s violated by its current value, then update its value to satisfy the ReLU constraint.
New Split Rule

\[
\begin{align*}
(v_j = \text{ReLU}(v_i)) \in R, & \quad l(v_i) < 0, \quad u(v_i) > 0 \\
u(v_i) := 0, & \quad l(v_i) := 0 \\
\end{align*}
\]

For the constraint \( v_j = \text{ReLU}(v_i) = \max(0, v_i) \), when \( v_i \) can be both positive and negative we have two cases:

Case 1: \( v_i \) is positive (achieved by setting its lower bound to 0)
Case 2: \( v_i \) is negative (achieved by setting its upper bound to 0)

The two cases create two branches in the derivation tree
A priori we don’t know which one will lead to success (so may require backtracking in proof search)

Case split is due to non-linearity of ReLU and crux of computational difficulty
ReluPLEX Algorithm in SMT Solver

- Starting with initial configuration, keep applying rules until SAT leaf found or all branches caused by split lead to UNSAT

- Multiple rules may be applicable in a given configuration
  - Derivation tree captures all possible branches of rule applications
  - Key difference with Simplex: Backtracking (exploring different branches) may be required!
  - Key benefit of Reluplex: Case split is demand driven and happens only when necessary

- Key engineering details of implementation
  - Which rule to choose in a given configuration
  - Choice of rules affects backtracking and time to termination
  - How to apply rules efficiently and how to backtrack efficiently
Soundness and Completeness

**SOUNDNESS**: If there is a derivation tree with at least one SAT leaf, $\varphi$ is satisfiable. If there is a derivation tree with all UNSAT leaves, $\varphi$ is not satisfiable.

**COMPLETENESS**: There is always a derivation tree in which every leaf is either SAT or UNSAT.
Comparison with existing SMT solvers

- Can encode $v_1 = ReLU(v_2)$ as $(v_2 \geq 0 \land v_1 = v_2) \lor (v_2 \leq 0 \land v_1 = 0)$.
- Existing SMT solvers perform many case splits.
- Reluplex can avoid/reduce splitting by using new pivot and update rules first.

<table>
<thead>
<tr>
<th></th>
<th>$\varphi_1$</th>
<th>$\varphi_2$</th>
<th>$\varphi_3$</th>
<th>$\varphi_4$</th>
<th>$\varphi_5$</th>
<th>$\varphi_6$</th>
<th>$\varphi_7$</th>
<th>$\varphi_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Z3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Yices</td>
<td>1</td>
<td>37</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MathSat</td>
<td>2040</td>
<td>9780</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gurobi</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reluplex</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>93</td>
<td>4</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

Time to termination in seconds with 4 hour timeout

Properties are from case study of neural-network-based controller for collision avoidance protocol ACAS (see ReluPlex paper; Katz et al; CAV 2017)
Experiments

Table: Local adversarial robustness tests. All times are in seconds.

<table>
<thead>
<tr>
<th>Point</th>
<th>δ = 0.1 Result</th>
<th>δ = 0.075 Result</th>
<th>δ = 0.05 Result</th>
<th>δ = 0.025 Result</th>
<th>δ = 0.01 Result</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>Point 1</td>
<td>SAT</td>
<td>135</td>
<td>SAT</td>
<td>239</td>
<td>SAT</td>
<td>24</td>
</tr>
<tr>
<td>Point 2</td>
<td>UNSAT</td>
<td>5880</td>
<td>UNSAT</td>
<td>1167</td>
<td>UNSAT</td>
<td>285</td>
</tr>
<tr>
<td>Point 3</td>
<td>UNSAT</td>
<td>863</td>
<td>UNSAT</td>
<td>436</td>
<td>UNSAT</td>
<td>99</td>
</tr>
<tr>
<td>Point 4</td>
<td>SAT</td>
<td>2</td>
<td>SAT</td>
<td>977</td>
<td>SAT</td>
<td>1168</td>
</tr>
<tr>
<td>Point 5</td>
<td>UNSAT</td>
<td>14560</td>
<td>UNSAT</td>
<td>4344</td>
<td>UNSAT</td>
<td>1331</td>
</tr>
</tbody>
</table>

Neural Network: Fully connected, 8 layers, 300 neurons
Abstract Interpretation using Boxes

Incomplete methods

We will investigate a specific type of incomplete method, based on bound propagation through the neural network. Starting with the initial pre-condition $\phi$, we will “pass” $\phi$ through the network, computing a convex over-approximation of the effect of each layer on $\phi$. Next, let's look at the “recipe” for certification with bound propagation.
Step1: compute convex $g(\phi)$ by propagating $\phi$

$\phi$:

$x_0 = [0.588, 0.65]$
$x_1 = [0.545, 0.608]$
$x_2 = [0.533, 0.596]$
...
$x_{3071} = [0.4, 0.463]$

All possible outputs

$g(\phi)$

$a_0 = 0$
$a_1 = 2.60 + 0.015\eta_0 + 0.023\eta_1 + 5.181\eta_2 + ...$
$a_2 = 4.63 - 0.005\eta_0 - 0.006\eta_1 + 0.023\eta_2 + ...$
...
$a_9 = 0.12 - 0.125\eta_0 + 0.102\eta_1 + 3.012\eta_2 + ...$
$\forall i. \eta_i \in [0,1]$
Step2: verify $\psi$

$g(\phi)$ is an over-approximation, hence if we fail to prove $\psi$, it could be the property is actually violated or there was an over-approximation introduced during propagation which prohibits provability.

$\psi$: every point in $\phi$ classifies as car

$\psi$: $\forall l \neq car. o_{car} > o_l$
Key challenge: how to produce convex shapes?

To instantiate incomplete methods which use bound propagation, we need two parts:

1. What is the convex approximation? E.g., Box, Zonotope, Polyhedra

2. How are these convex approximations produced? That is, what is the effect of the layer on a given approximation? This effect is often called an abstract transformer as it transforms abstract shapes.
Popular convex shapes

Box: \( l_i \leq x_i \leq u_i \)

Zonotope: \( \hat{x}_i = \alpha_0 + \sum \alpha_i \epsilon_i, \epsilon_i \in [-1,1] \)

Octagon: \( l_i \leq x_i \leq u_i, \pm x_i \pm x_j \leq c_{ij} \)

Polyhedra: \( \sum a_i x_i \leq c \)
A trade-off between approximation and speed exists:

- transformers for Box are fast, but imprecise,
- while Polyhedra is more precise but has exponential complexity.
Box Abstract Transformer: Addition

\[ [a, b] +^\# [c, d] = [a + c, b + d] \]

Addition transformer

- Here, \( a, b \in R^m \) where \( \forall i.a_i \leq b_i \)
- \(^\#\) denotes the abstract effect of the operation on the box

Example in 1-dim (interval addition) \[ [1, 3] +^\# [5, 8] = [6, 11] \]
Box Abstract Transformers for ReLU Networks

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>([a, b] +^# [c, d] = [a + c, b + d])</td>
<td>Addition transformer</td>
</tr>
<tr>
<td>(-^# [a, b] = [-b, -a])</td>
<td>Negation transformer</td>
</tr>
<tr>
<td>(ReLU^# [a, b] = [ReLU(a), ReLU(b)])</td>
<td>ReLU transformer</td>
</tr>
<tr>
<td>(\lambda^# [a, b] = [\lambda \cdot a, \lambda \cdot b])</td>
<td>Multiplication by a constant (\lambda &gt; 0)</td>
</tr>
<tr>
<td>([a, b] &gt;^# [c, d] \text{ holds iff } a &gt; d)</td>
<td>Comparison transformer</td>
</tr>
</tbody>
</table>

- Here, \(a, b \in \mathbb{R}^m\) where \(\forall i. a_i \leq b_i\)
- \(ReLU(x) = \max(0, x)\)
- \(^#\) denotes the abstract effect of the operation on the box
Computation with Box transformer

We have 2 pixels \((x_1, x_2)\) as input ranging over \([0, 0.3]\) and \([0.1, 0.4]\).

Bounds using Box:
- \(0.1 \leq x_3 \leq 0.7\)
- \(-0.4 \leq x_4 \leq 0.2\)

Exact bounds would be:
- \(x_3 = x_1 + x_2\)
- \(x_4 = x_1 - x_2\)
- \(0 \leq x_1 \leq 0.3\)
- \(0.1 \leq x_2 \leq 0.4\)

\[\begin{align*}
\text{def } f_1(x_1, x_2): \\
x_3 &= x_1 + x_2 \\
x_4 &= x_1 - x_2 \\
\text{return } x_3, x_4
\end{align*}\]
Optimal Box transformer is not exact!

Box approximation contains extra points

Exact
Key Point

Even though the Box abstract transformer for the affine computation is optimal for the Box relaxation, it may not be complete (exact)! Nonetheless, even if not exact, it may be enough to verify the property of interest.
Box succeeds in verifying robustness

We have 2 pixels \((x_1, x_2)\) as input ranging over \([0, 0.3]\) and \([0.1, 0.4]\), we want to prove that \(o_0 > o_1\) holds for all inputs.

\[
\begin{align*}
\text{def } f(x_1, x_2): \\
x_3 &= x_1 + x_2 \\
x_4 &= x_1 - x_2 \\
x_5 &= \max(0, x_3) \\
x_6 &= \max(0, x_4) \\
o_0 &= x_5 + x_6 + 0.5 \\
o_1 &= 2 \cdot x_5 - x_6 - 1 \\
\text{return } o_0, o_1
\end{align*}
\]

Using Box, we succeed in proving the network classifies any input in the range as 0. This is because \([0.6, 1.4] > [-1, 0.4]\), provably so.
Box fails in verifying robustness

Let us slightly increase the range of the input pixels to [0, 0.6] and [0.1, 0.7]

Using Box, we failed to prove the network classifies any input in the range as 0, even though property actually holds. This is because [0.6, 2.3] is not > [-1.3, 1.6], provably so.
Robustness Certification

- Instead of Box domain, one can use Zonotopes to get reasonably scalable and more precise approximations

- Constraint solving and abstract domains can be combined in different ways

- Robustness verification can be integrated into training
  - Parameters are updated not only to minimize loss but also to ensure that a verification procedure (e.g. based on Box abstract domain) gives robustness guarantee
Adversarial Robustness Recap

- Adversarial Examples
- Adversarial training
- Certified robustness via randomized smoothing
- Sample of current research on robustness for LLMs
- Formal methods for verified robustness
  - Specialized constraint solver ReluPlex for neural network verification
  - Verifying robustness by abstract interpretation (box domain)