Verification of FNN with Piecewise Linear Activation Functions using CEGAR

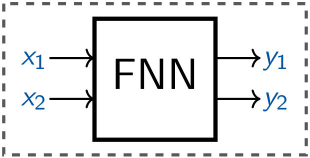
Final Presentation

Franz Link

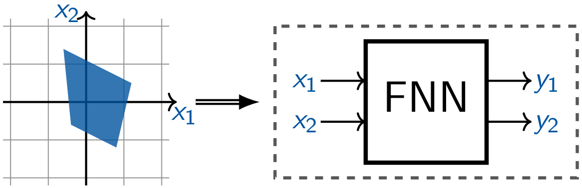
Supervisor: Prof. Dr. Dr.h.c. Erika Ábrahám
Second reviewer: apl. Prof. Dr. Thomas Noll
Advisor: László Antal

LuFG Theory of Hybrid Systems

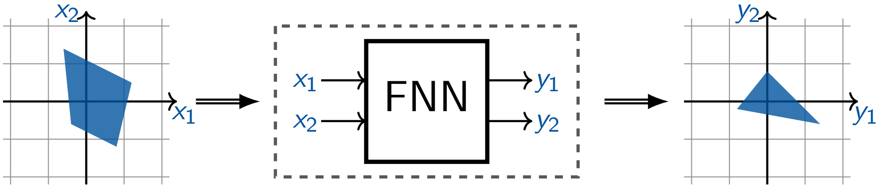
Motivation: Verification with CEGAR



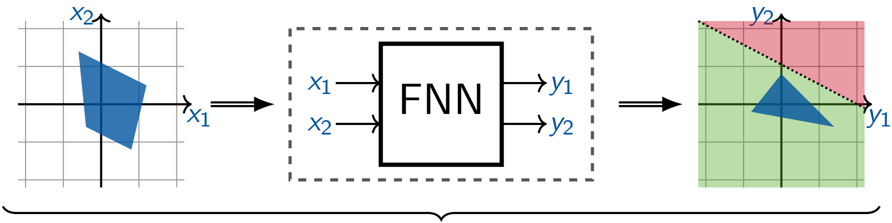
Motivation: Verification with CEGAR



Motivation: Verification with CEGAR

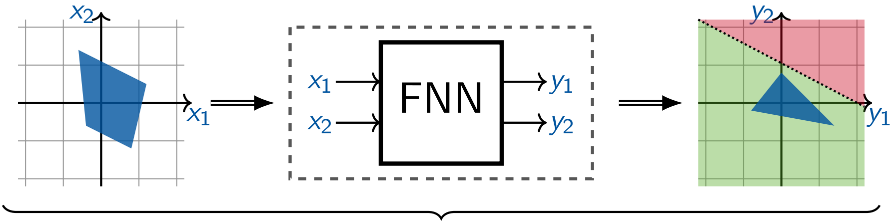


Motivation: Verification with CEGAR



Reachability Analysis based Verification

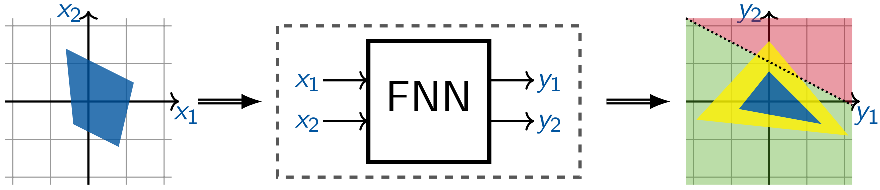
Motivation: Verification with CEGAR



Reachability Analysis based Verification

Exact
Complete
Sound
Slow

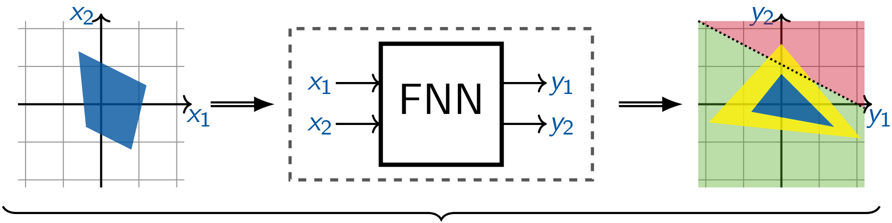
Motivation: Verification with CEGAR



Reachability Analysis based Verification

Exact	Over-Approximate
Complete	Incomplete
Sound	Sound
Slow	Fast

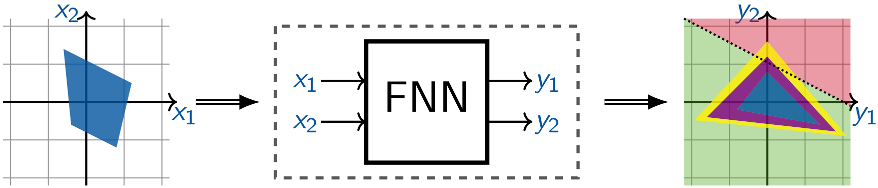
Motivation: Verification with CEGAR



Reachability Analysis based Verification

Exact	CEGAR	Over-Approximate
Complete	Complete	Incomplete
Sound	Sound	Sound
Slow	Potentially Faster	Fast

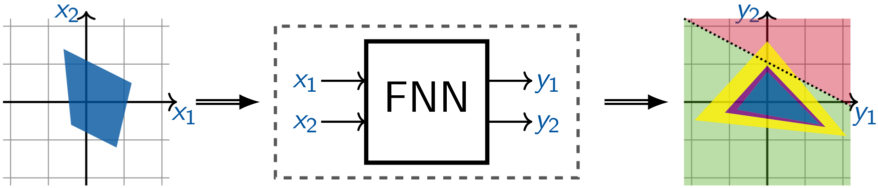
Motivation: Verification with CEGAR



Reachability Analysis based Verification

Exact	CEGAR	Over-Approximate
Complete	Complete	Incomplete
Sound	Sound	Sound
Slow	Potentially Faster	Fast

Motivation: Verification with CEGAR



Reachability Analysis based Verification

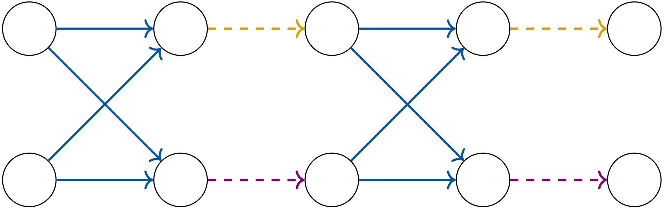
Exact	CEGAR	Over-Approximate
Complete	Complete	Incomplete
Sound	Sound	Sound
Slow	Potentially Faster	Fast

Outline

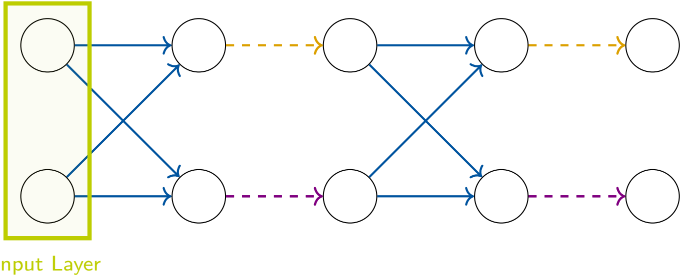
Outline

1. Preliminaries
   - Feedforward Neural Networks
   - Reachability Analysis
2. Verification with CEGAR
   - Safety Checking
   - Tracing a Counterexample
   - Refinement Heuristic
3. Performance
4. Conclusion

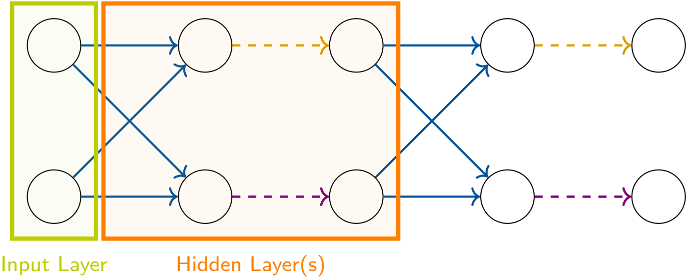
Feedforward Neural Networks (FNN) [Antal et al., 2023]



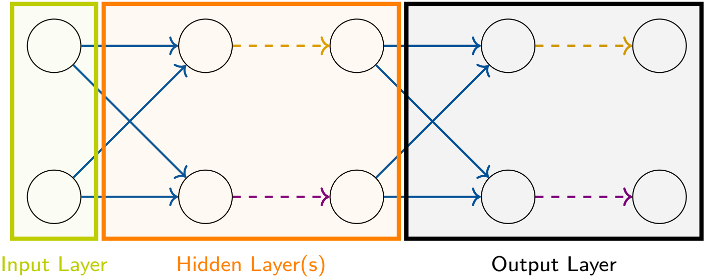
Feedforward Neural Networks (FNN) [Antal et al., 2023]



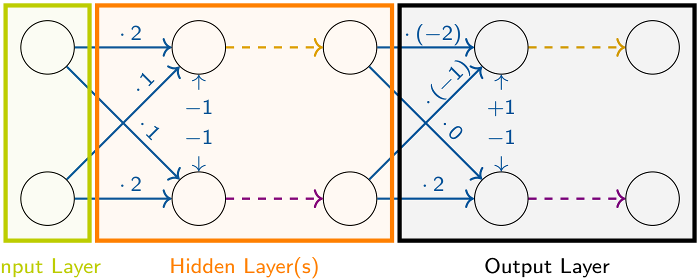
Feedforward Neural Networks (FNN) [Antal et al., 2023]



Feedforward Neural Networks (FNN) [Antal et al., 2023]



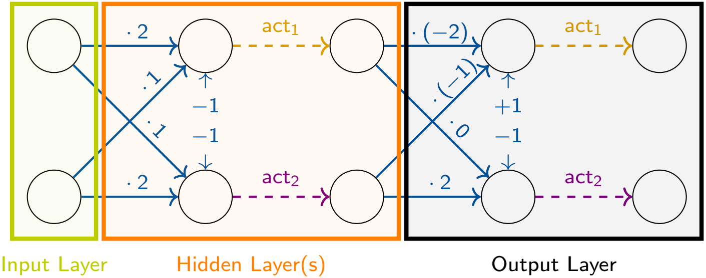
Feedforward Neural Networks (FNN) [Antal et al., 2023]



Layer:

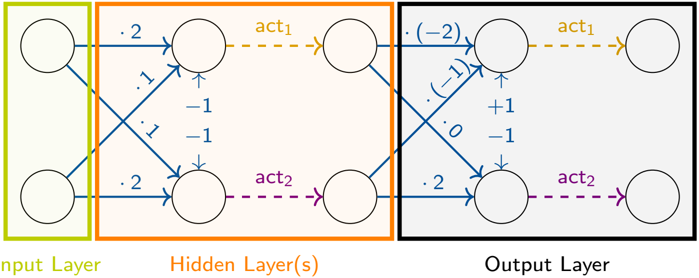
1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)

Feedforward Neural Networks (FNN) [Antal et al., 2023]



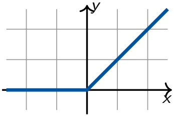
Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function



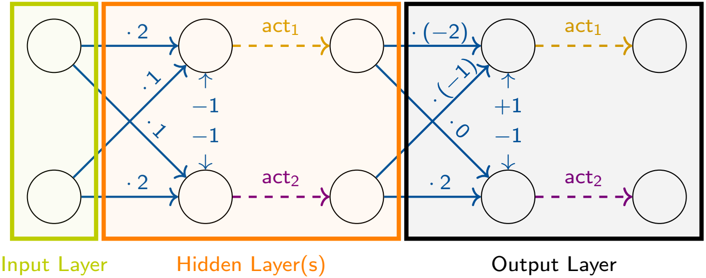
Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function



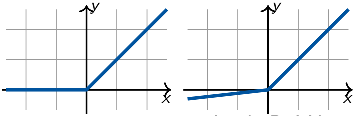
ReLU

Feedforward Neural Networks (FNN) [Antal et al., 2023]

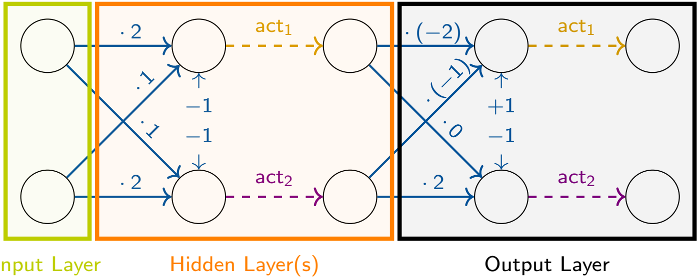


Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

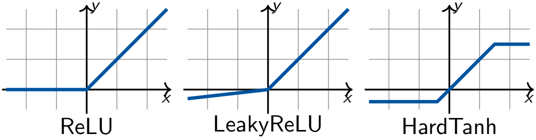


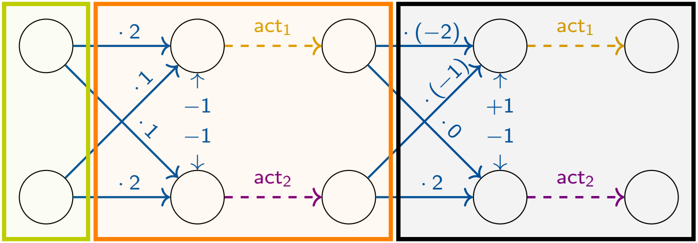
ReLU LeakyReLU



Layer:

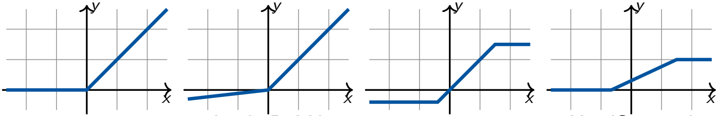
1. Affine mapping ($\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$)
2. Piecewise linear activation function





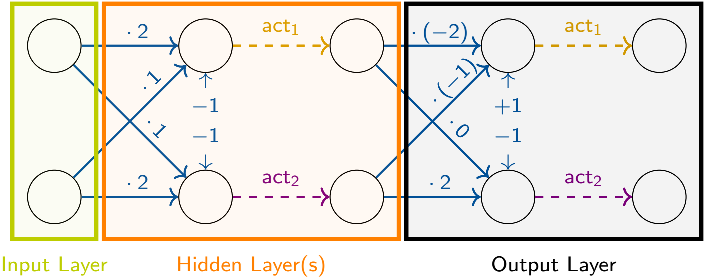
Layer:

1. Affine mapping ($\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$)
2. Piecewise linear activation function



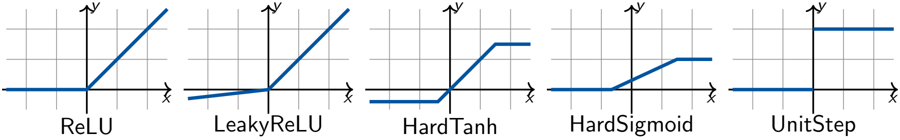
ReLU LeakyReLU HardTanh HardSigmoid

Feedforward Neural Networks (FNN) [Antal et al., 2023]

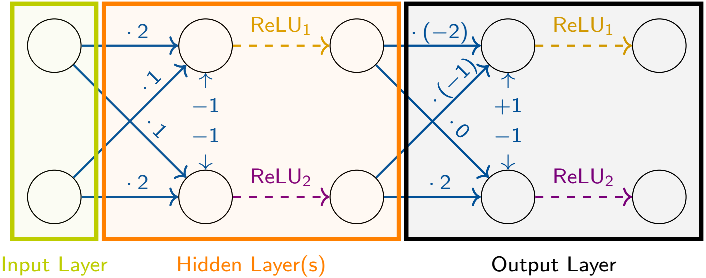


Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

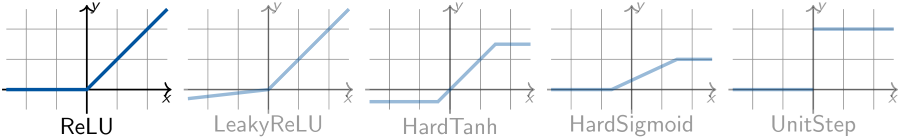


Feedforward Neural Networks (FNN) [Antal et al., 2023]

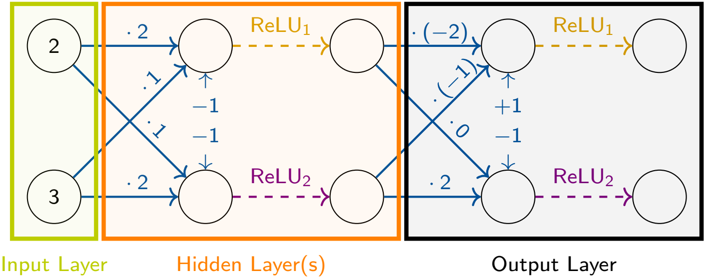


Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

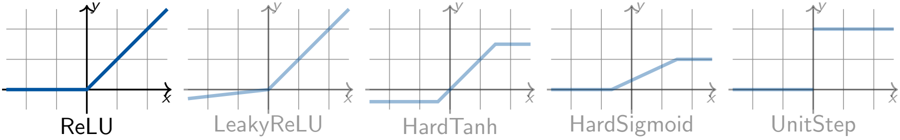


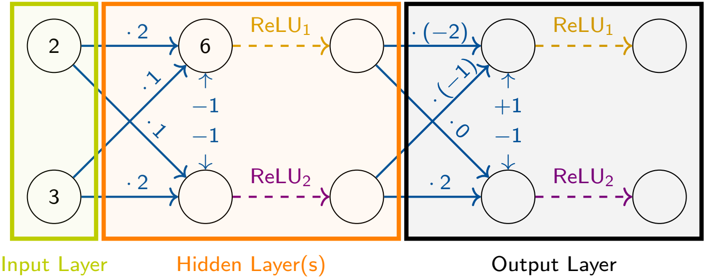
Feedforward Neural Networks (FNN) [Antal et al., 2023]



Layer:

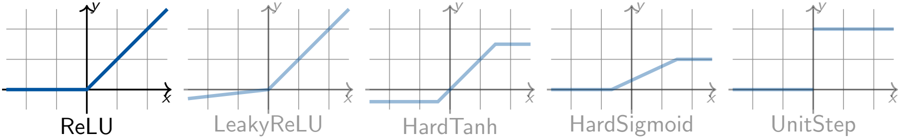
1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function



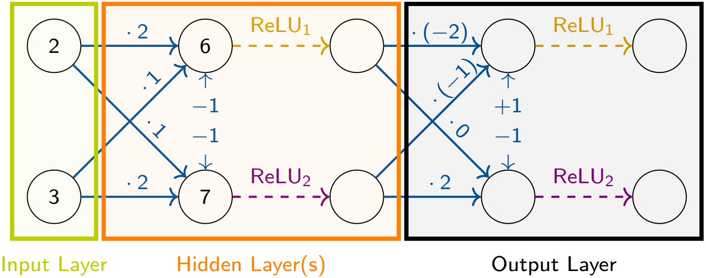


Layer:

1. Affine mapping $(\boldsymbol{Wx} + \boldsymbol{b})$
2. Piecewise linear activation function

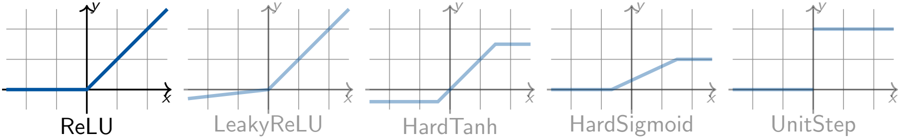


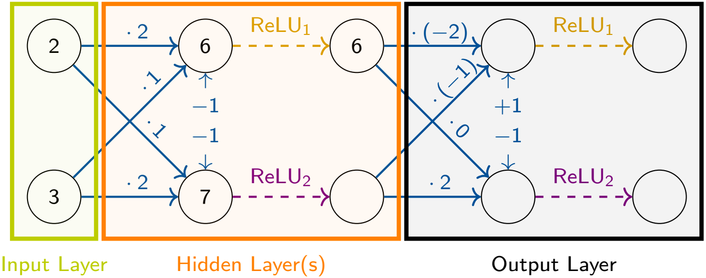
Feedforward Neural Networks (FNN) [Antal et al., 2023]



Layer:

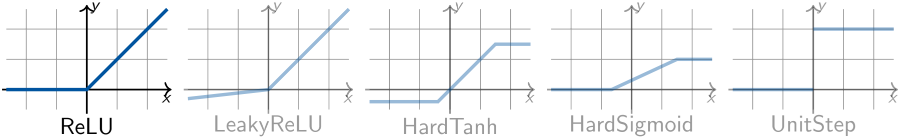
1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function



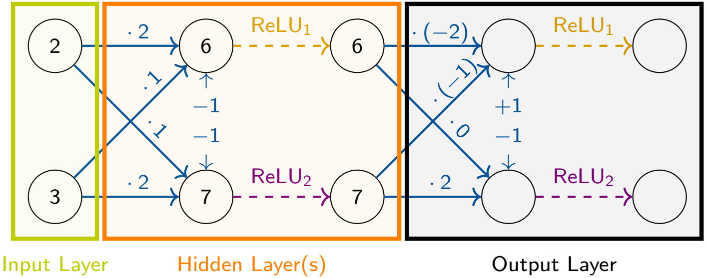


Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

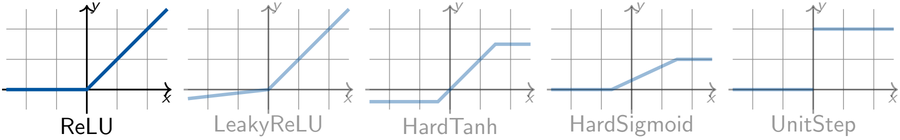


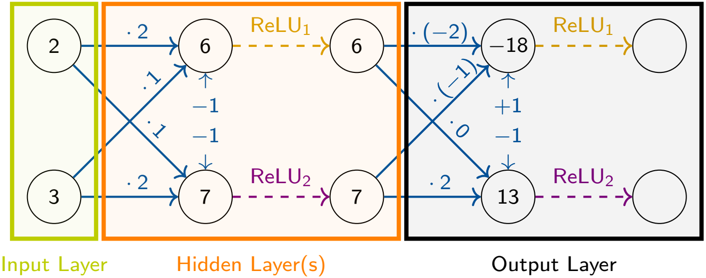
Feedforward Neural Networks (FNN) [Antal et al., 2023]



Layer:

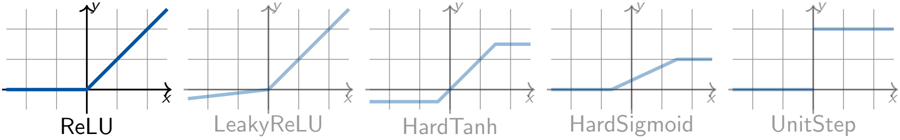
1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function



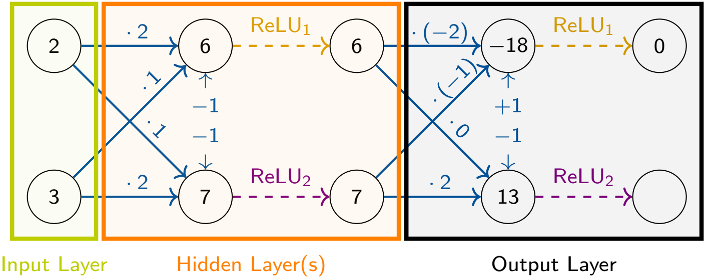


Layer:

1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

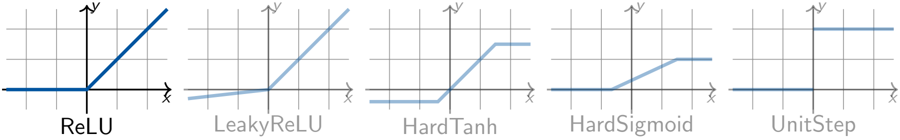


Feedforward Neural Networks (FNN) [Antal et al., 2023]

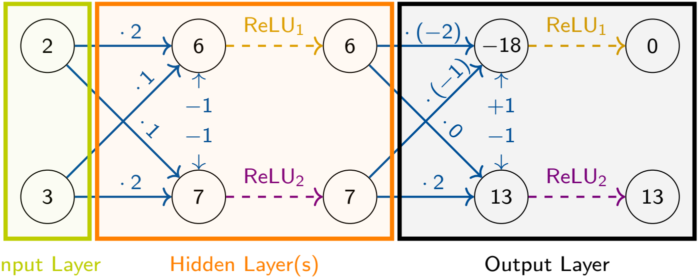


Layer:

1. Affine mapping ($\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$)
2. Piecewise linear activation function

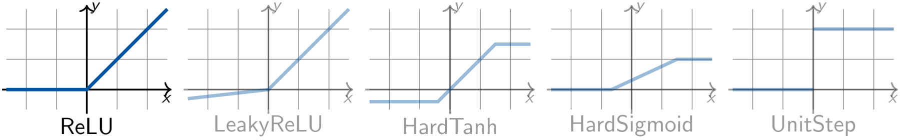


Feedforward Neural Networks (FNN) [Antal et al., 2023]

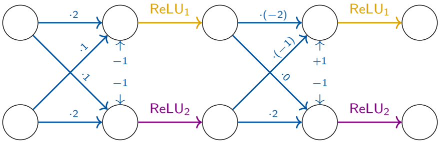


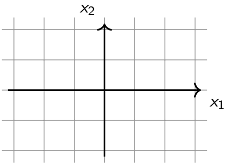
Layer:

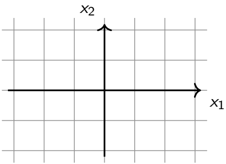
1. Affine mapping ($\boldsymbol{Wx} + \boldsymbol{b}$)
2. Piecewise linear activation function

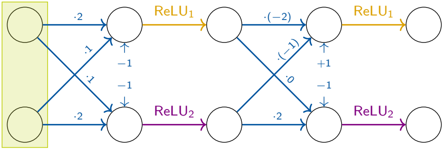


Reachability Analysis [Tran et al., 2019]







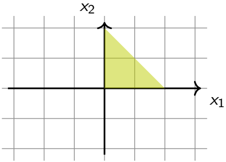


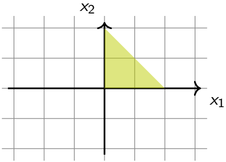
Exact

-

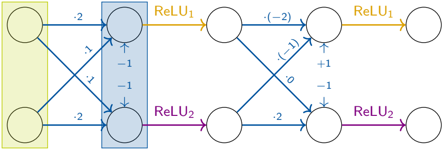
Over-Approximate

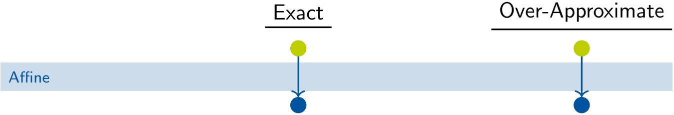
-

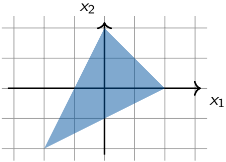


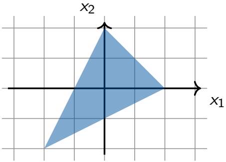


Reachability Analysis [Tran et al., 2019]

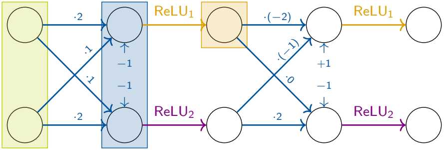


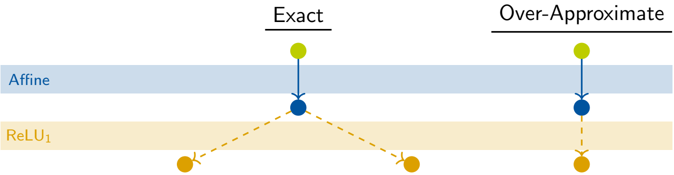


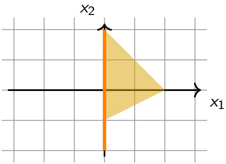


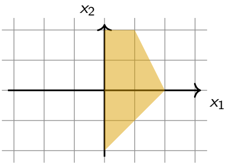


Reachability Analysis [Tran et al., 2019]

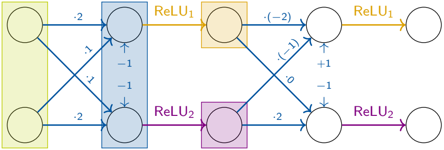


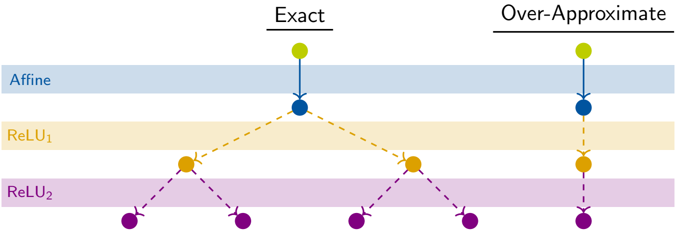


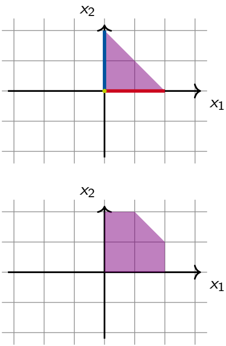




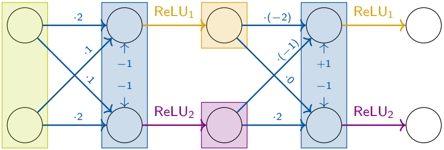
Reachability Analysis [Tran et al., 2019]

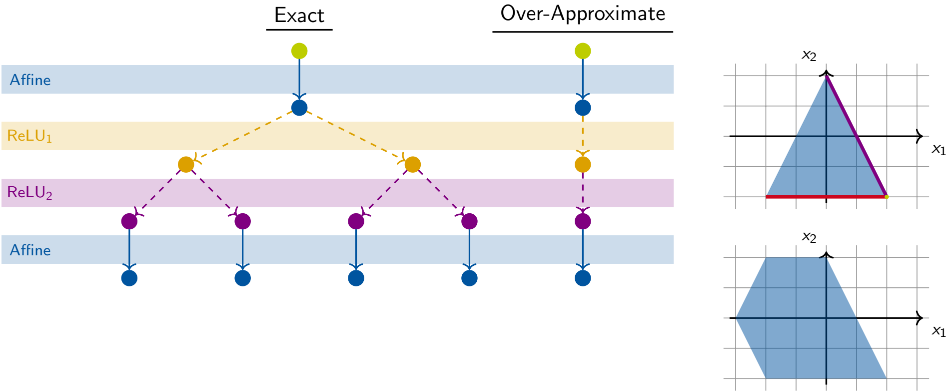




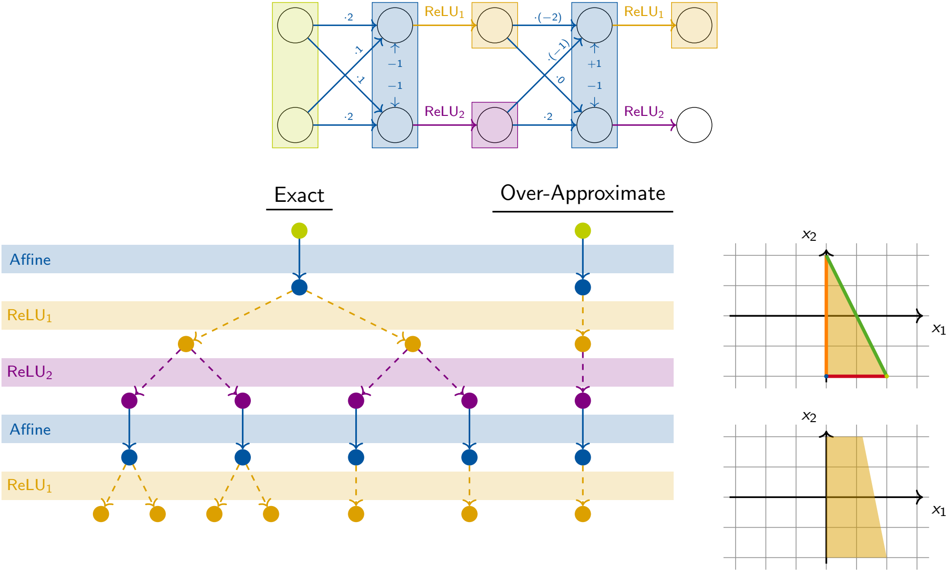


Reachability Analysis [Tran et al., 2019]

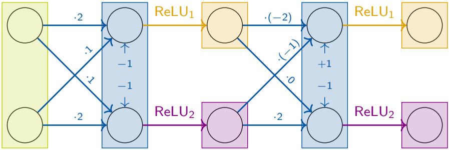


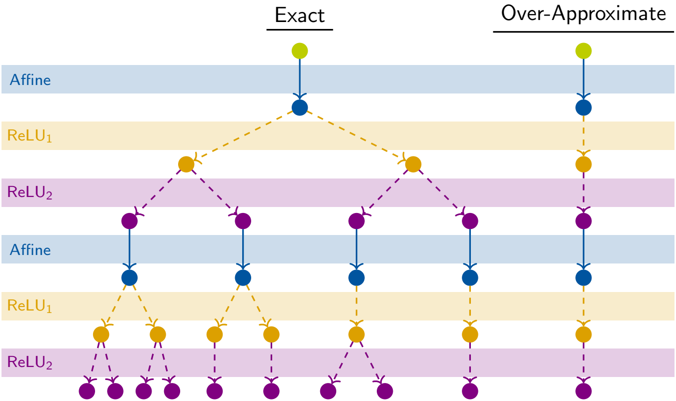


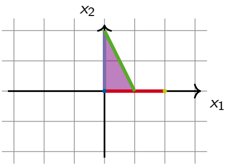
Reachability Analysis [Tran et al., 2019]

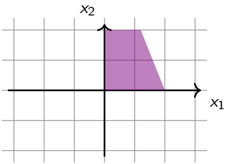


Reachability Analysis [Tran et al., 2019]









Outline

Overview: Verification with CEGAR

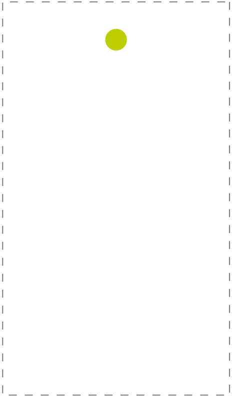
FNN
Input Set
Safety Specification

Overview: Verification with CEGAR

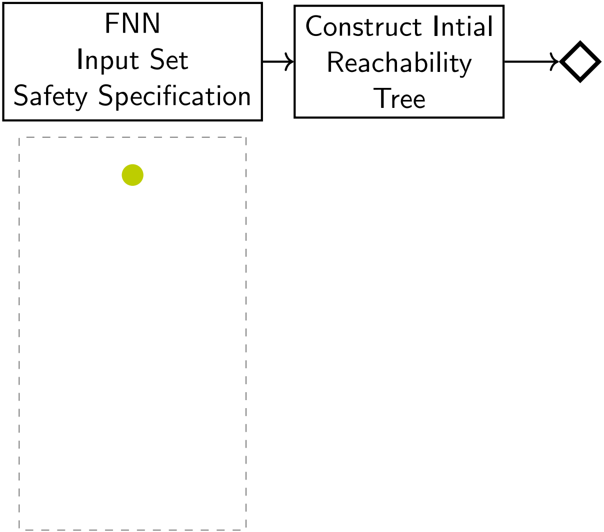
FNN
Input Set
Safety Specification

→

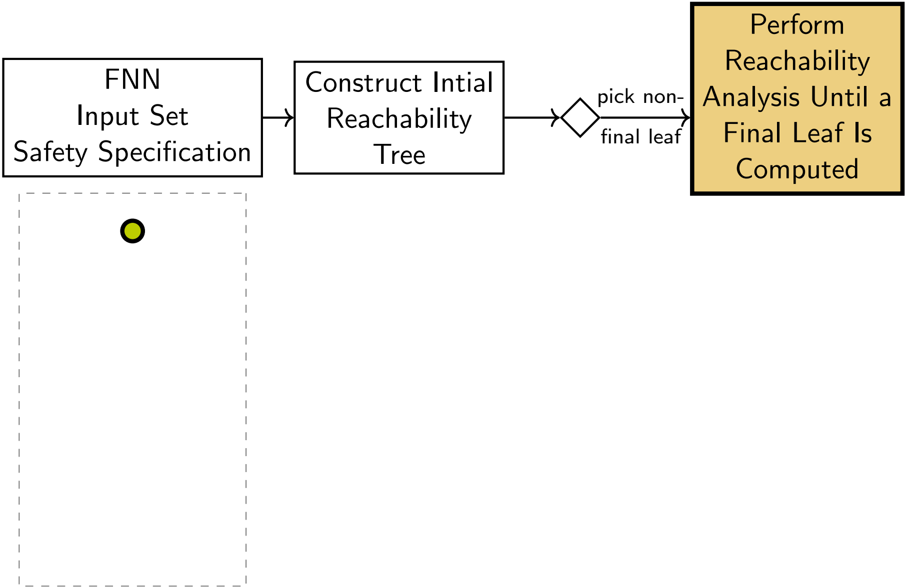
Construct Intial
Reachability
Tree



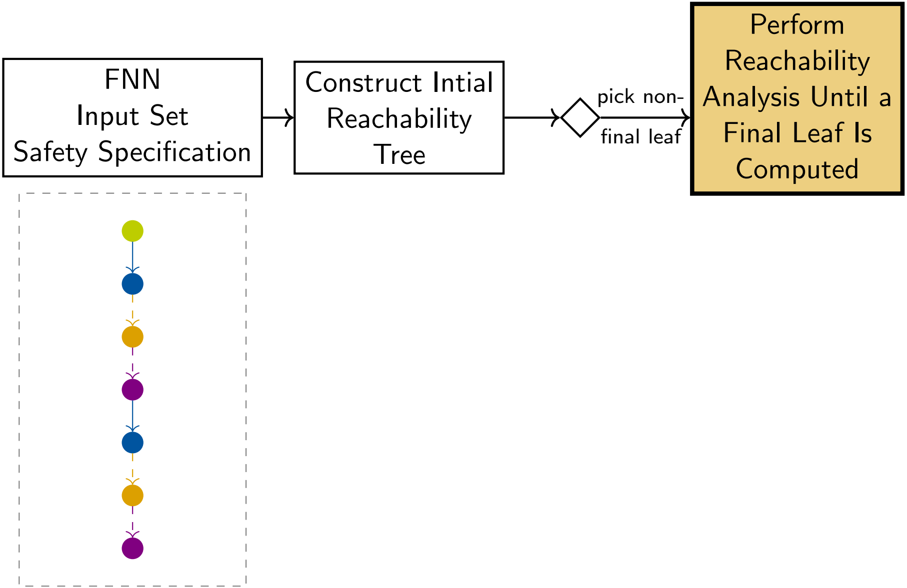
Overview: Verification with CEGAR



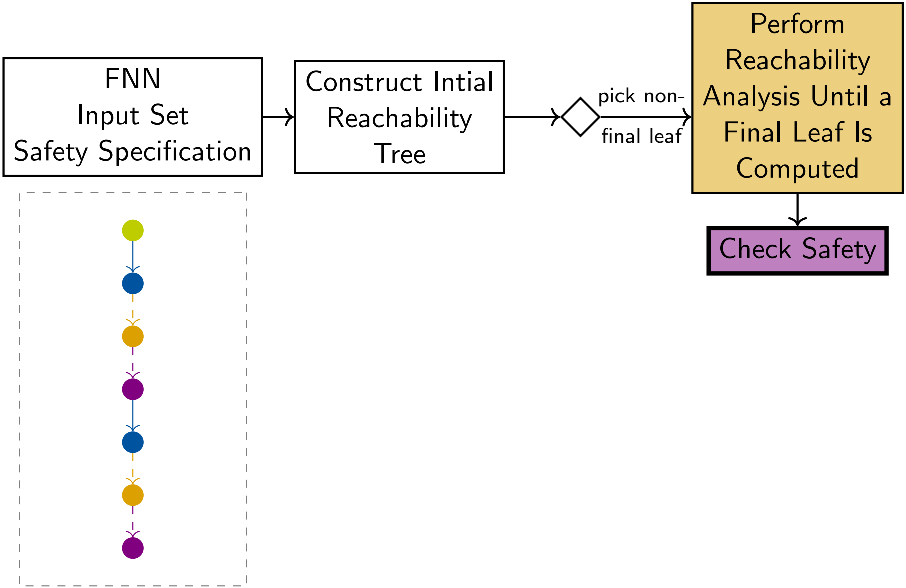
Overview: Verification with CEGAR



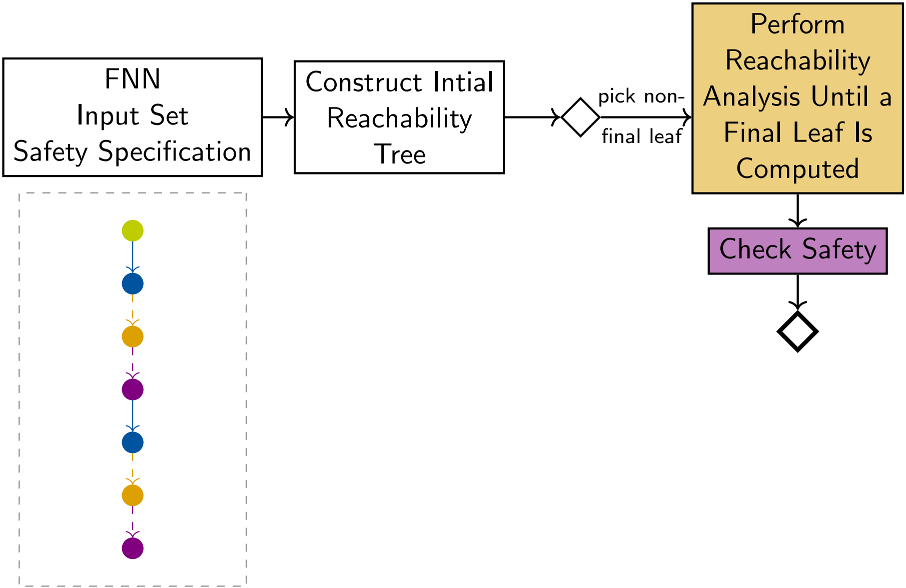
Overview: Verification with CEGAR



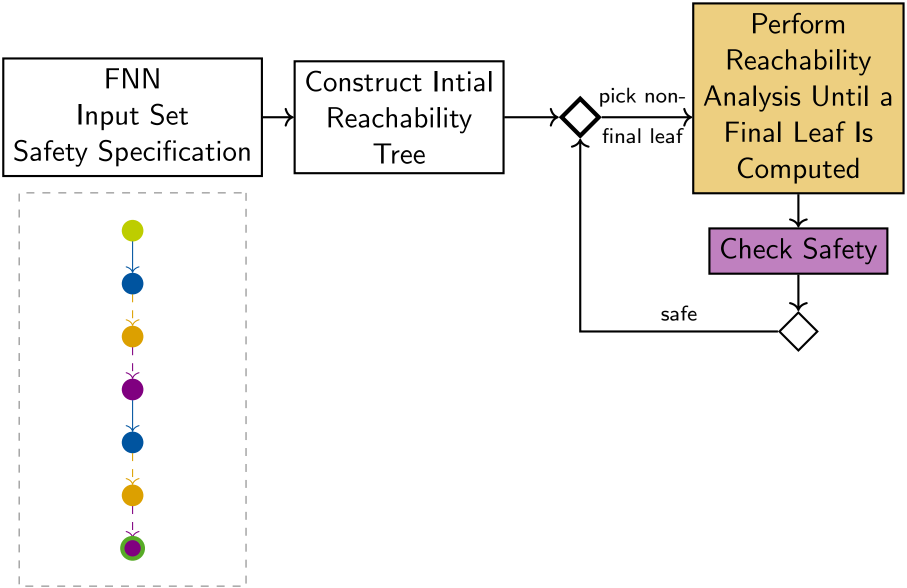
Overview: Verification with CEGAR



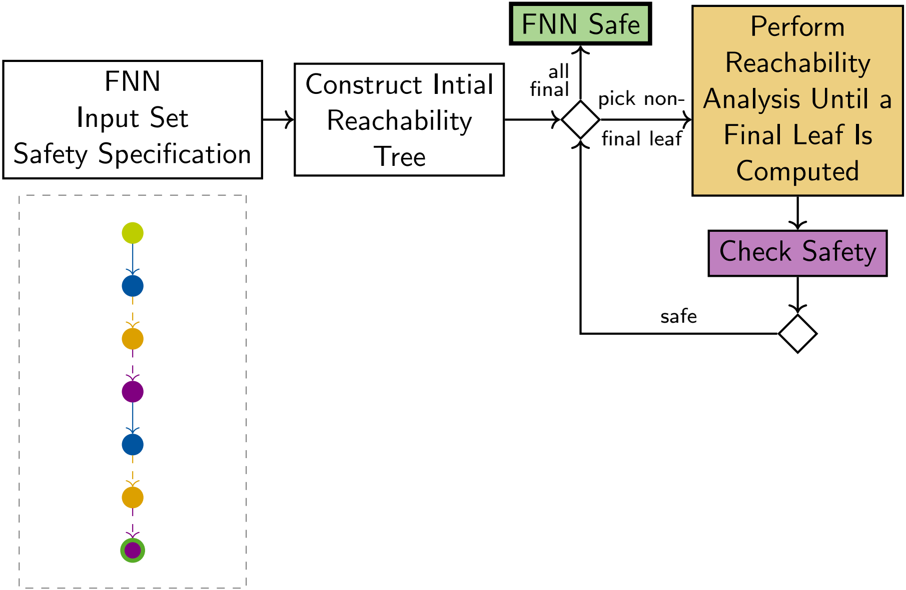
Overview: Verification with CEGAR



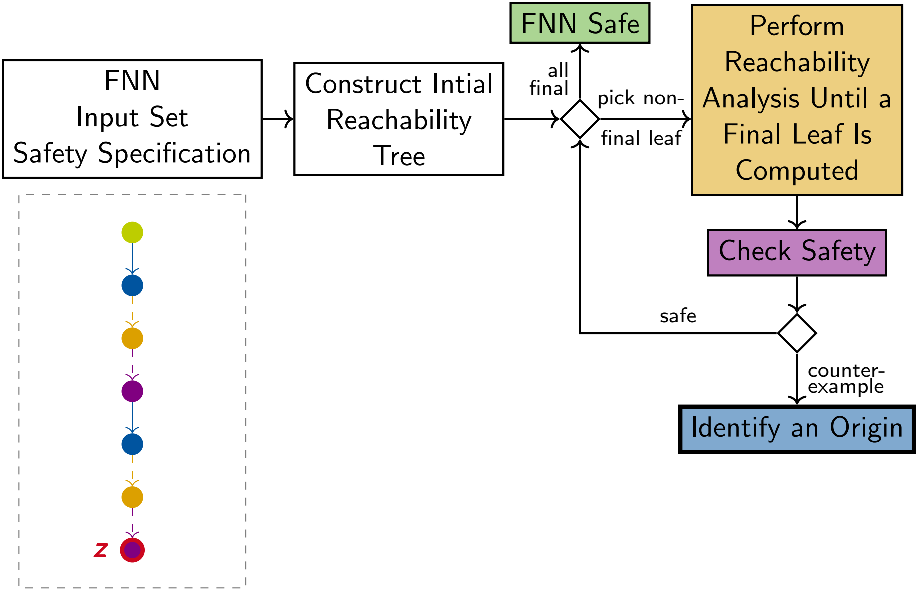
Overview: Verification with CEGAR



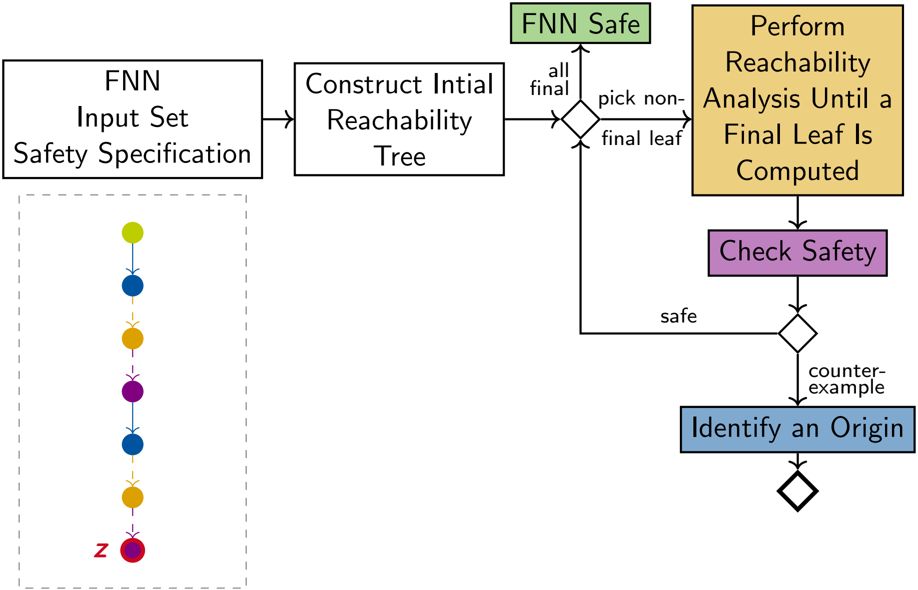
Overview: Verification with CEGAR



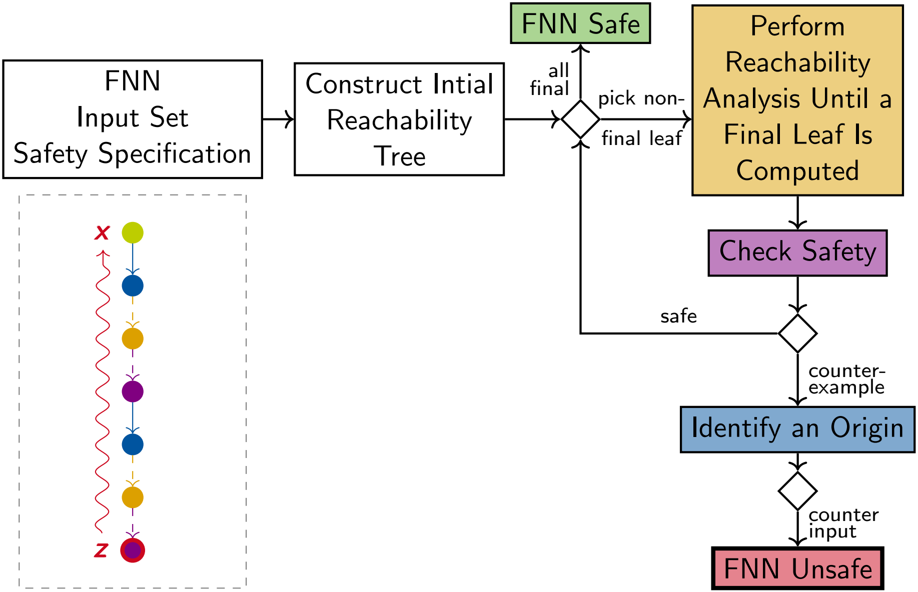
Overview: Verification with CEGAR



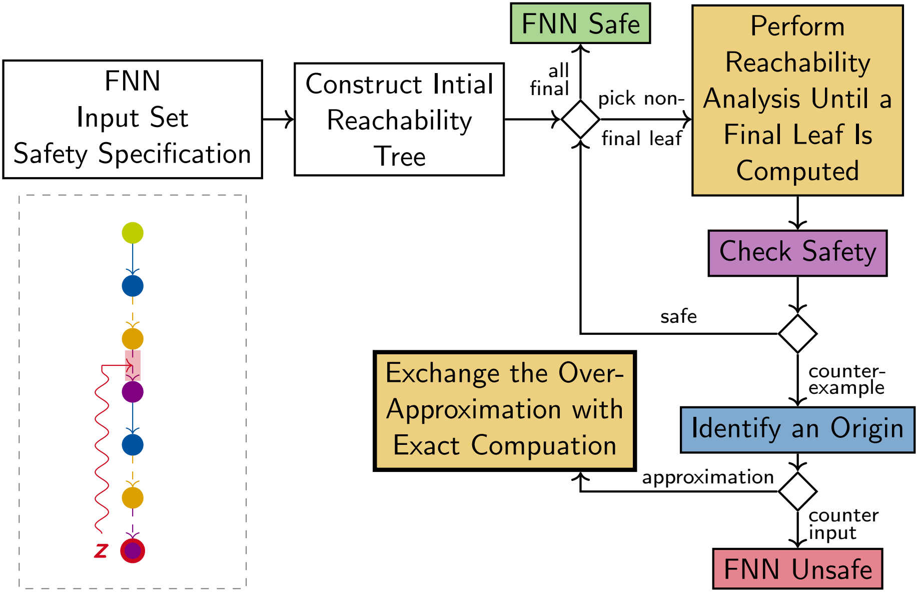
Overview: Verification with CEGAR



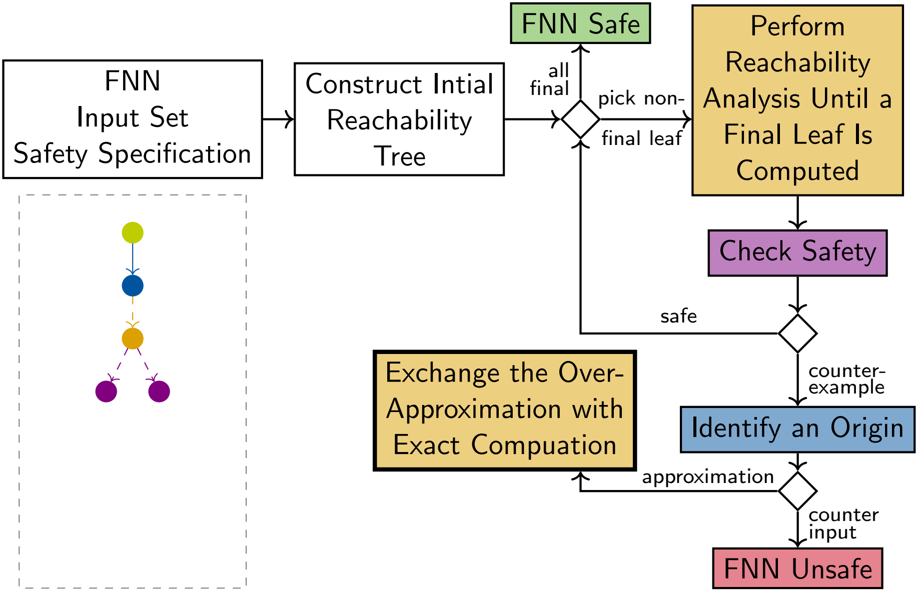
Overview: Verification with CEGAR



Overview: Verification with CEGAR

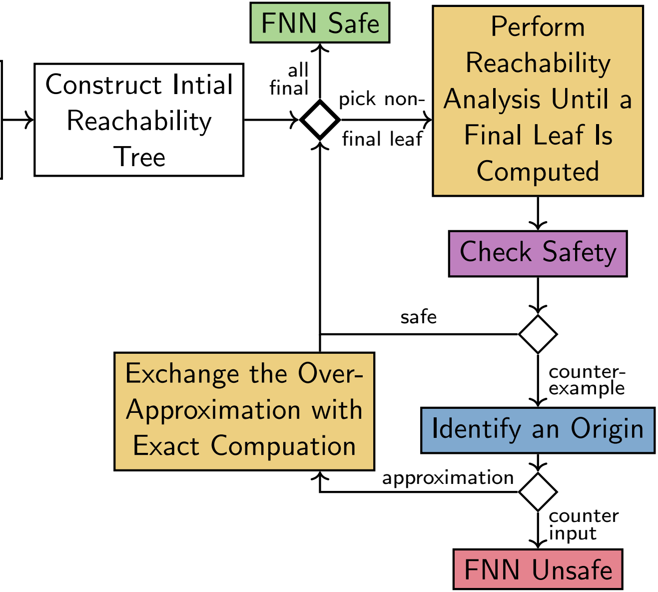


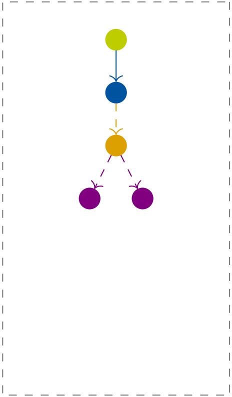
Overview: Verification with CEGAR



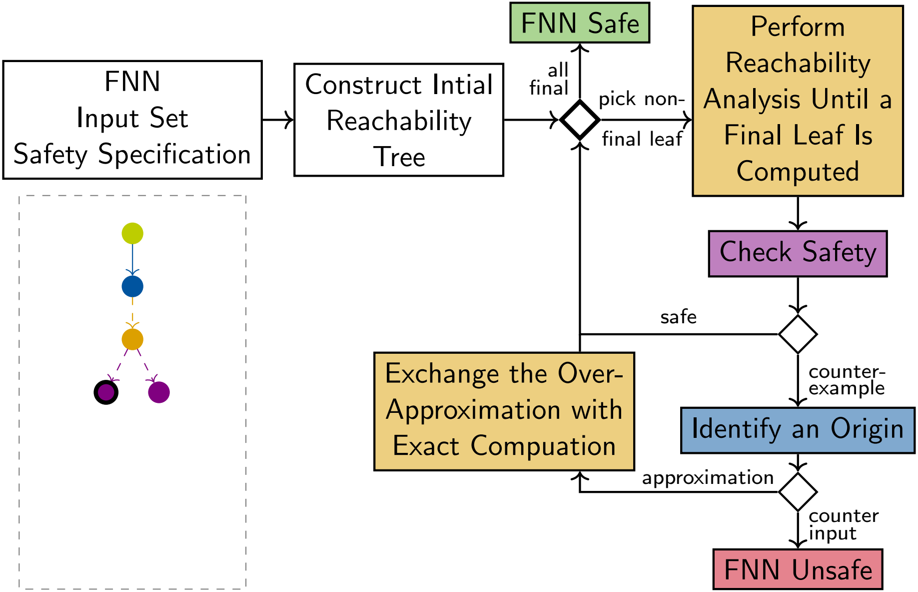
Overview: Verification with CEGAR

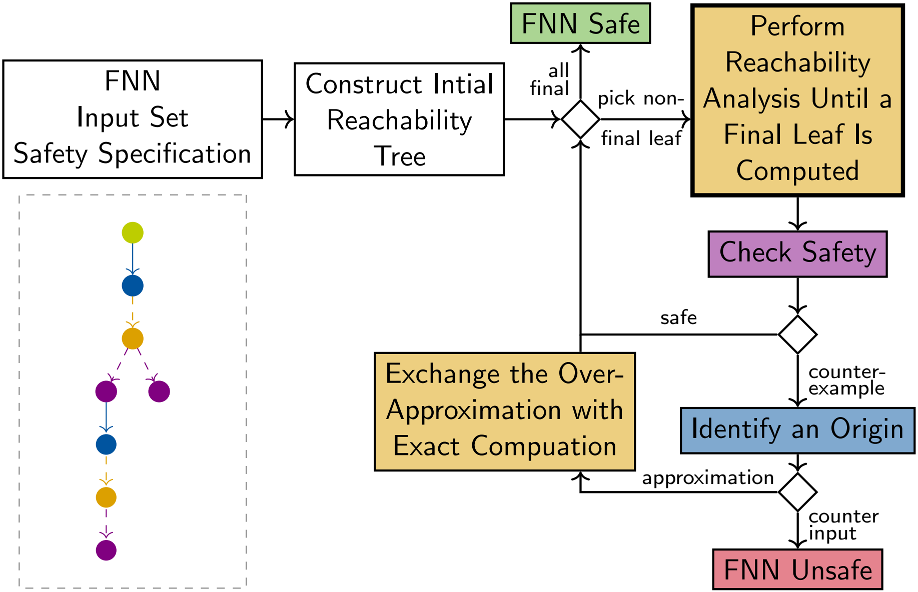
FNN
Input Set
Safety Specification



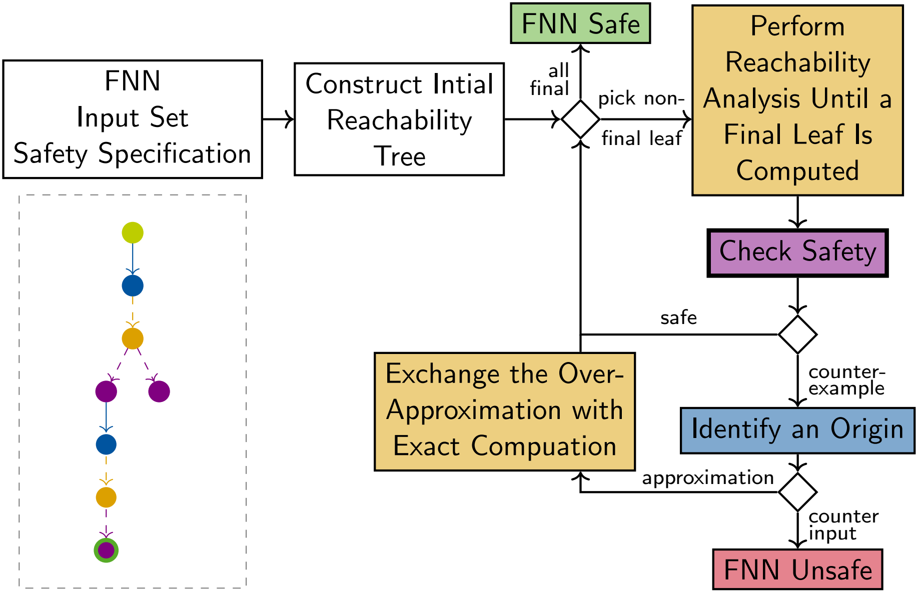


Overview: Verification with CEGAR

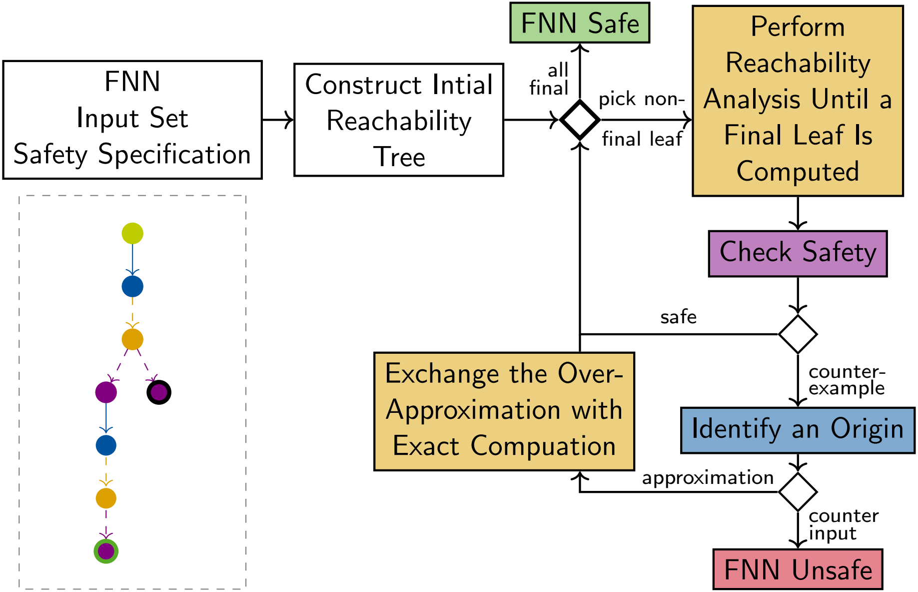


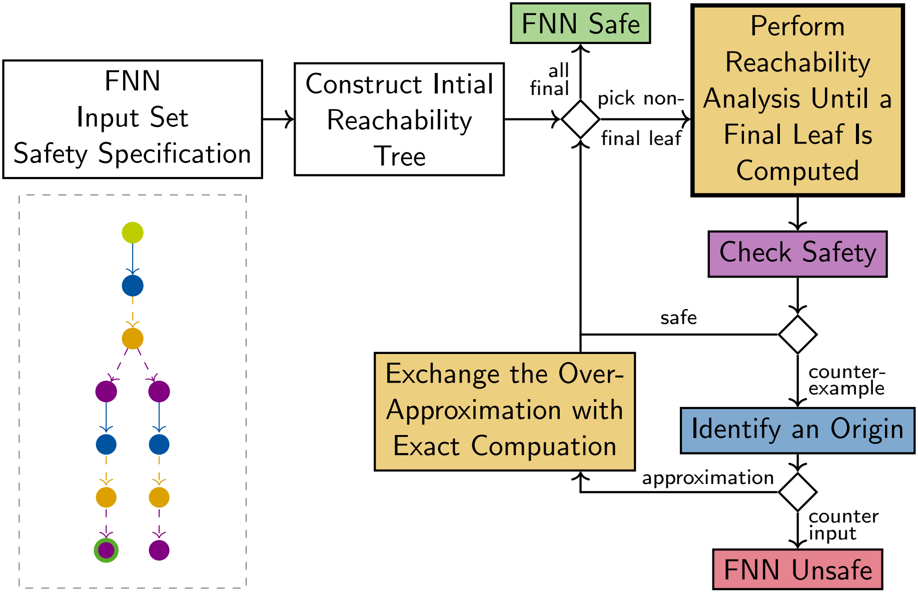

FNN
Input Set
Safety Specification
Construct Intial Reachability Tree
all final
FNN Safe
pick non-final leaf
Perform Reachability Analysis Until a Final Leaf Is Computed
Check Safety
safe
counter-example
Identify an Origin
approximation
Exchange the Over-Approximation with Exact Compuation
counter input
FNN Unsafe

Overview: Verification with CEGAR

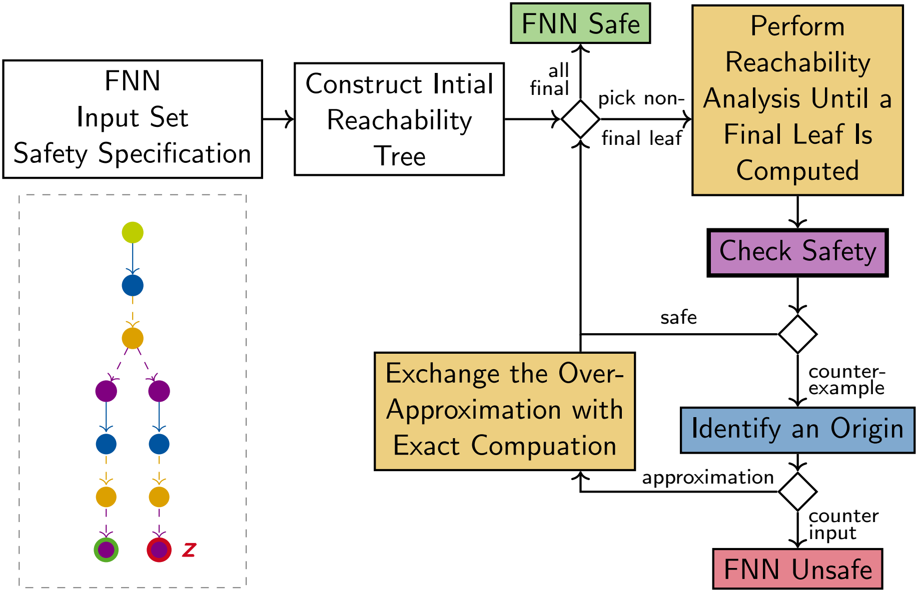


Overview: Verification with CEGAR

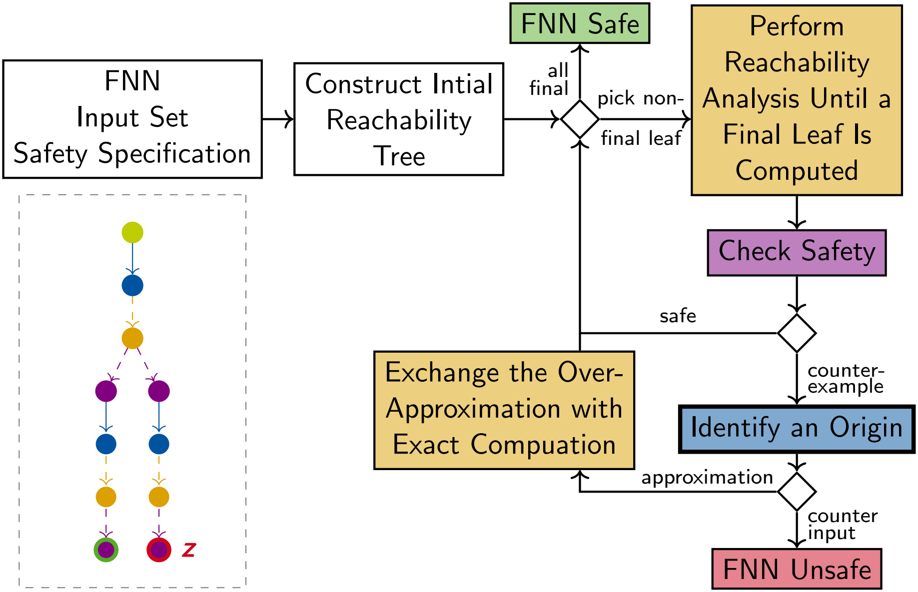



FNN
Input Set
Safety Specification
Construct Intial Reachability Tree
all final
FNN Safe
pick non-final leaf
Perform Reachability Analysis Until a Final Leaf Is Computed
Check Safety
safe
counter-example
Identify an Origin
approximation
counter input
FNN Unsafe
Exchange the Over-Approximation with Exact Compuation

Overview: Verification with CEGAR



Overview: Verification with CEGAR



Outline

Safety Checking

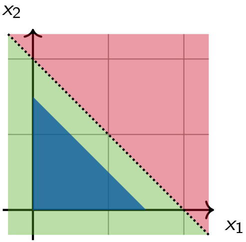
- Reachable set $\Theta \subseteq F(\mathcal{I})$

Safety Checking

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

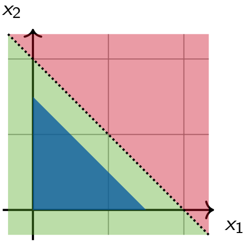
Safety Checking

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

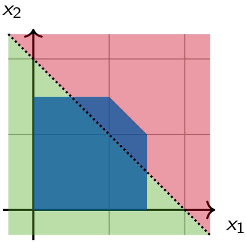


Safety Checking

- Reachable set $\Theta \subseteq F(\mathcal{I})$

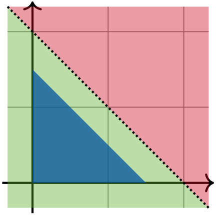


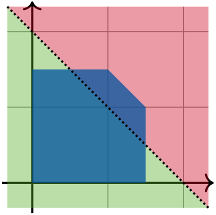
- Safety Specification S



- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

Θ is unsafe

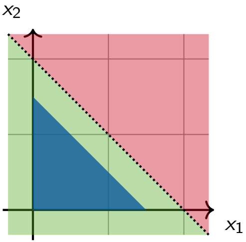


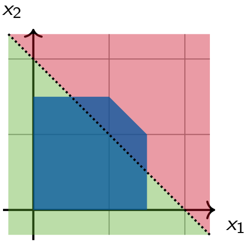


Safety Checking

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

$$\Theta \text{ is unsafe} \quad \Leftrightarrow \quad \text{A counterexample exists in } \Theta$$

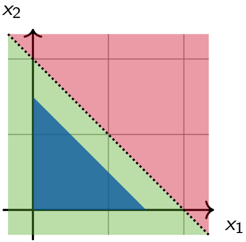


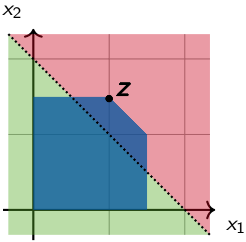


Safety Checking

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

$$\begin{aligned} \Theta \text{ is unsafe} \quad &\Leftrightarrow \quad \text{A counterexample exists in } \Theta \\ &\Leftrightarrow \quad \boldsymbol{z} \in \Theta \wedge \boldsymbol{z} \notin S \end{aligned}$$



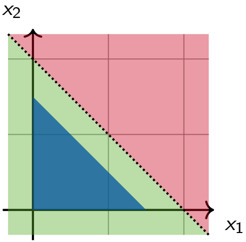


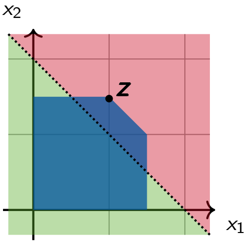
Safety Checking

Definition: A star $\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, \mathcal{P} \rangle$ represents $\{\boldsymbol{c} + \boldsymbol{V}\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathcal{P}\}$

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

$$\begin{aligned} \Theta \text{ is unsafe} \quad &\Leftrightarrow \quad \text{A counterexample exists in } \Theta \\ &\Leftrightarrow \quad \boldsymbol{z} \in \Theta \wedge \boldsymbol{z} \notin S \end{aligned}$$



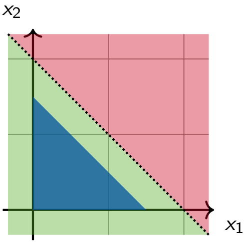


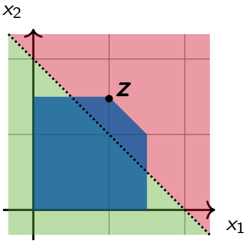
Safety Checking

Definition: A star $\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, \mathcal{P} \rangle$ represents $\{\boldsymbol{c} + \boldsymbol{V}\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathcal{P}\}$

- Reachable set $\Theta \subseteq F(\mathcal{I})$
- Safety Specification S

$$\begin{aligned} \Theta \text{ is unsafe} \quad &\Leftrightarrow \quad \text{A counterexample exists in } \Theta \\ &\Leftrightarrow \quad \boldsymbol{z} \in \Theta \wedge \boldsymbol{z} \notin S \\ &\Leftrightarrow \quad \boldsymbol{\alpha} \in \mathcal{P} \wedge (\boldsymbol{c} + \boldsymbol{V}\boldsymbol{\alpha}) \notin S \end{aligned}$$





Methods for Safety Checking

Basic Method: Define $\psi(\boldsymbol{z})$ such that satisfying $\boldsymbol{z}$ are counterexamples

Methods for Safety Checking

Basic Method: Define $\psi(\boldsymbol{z})$ such that satisfying $\boldsymbol{z}$ are counterexamples

Heuristics:

- Reuse previous counterexamples

Methods for Safety Checking

Basic Method: Define $\psi(\boldsymbol{z})$ such that satisfying $\boldsymbol{z}$ are counterexamples

Heuristics:

- Reuse previous counterexamples

Methods for Safety Checking

Basic Method: Define $\psi(\boldsymbol{z})$ such that satisfying $\boldsymbol{z}$ are counterexamples

Heuristics:

- Reuse previous counterexamples
- Reduce the representation $z_i = \frac{n_i}{d_i}$

Methods for Safety Checking

Basic Method: Define $\psi(\boldsymbol{z})$ such that satisfying $\boldsymbol{z}$ are counterexamples

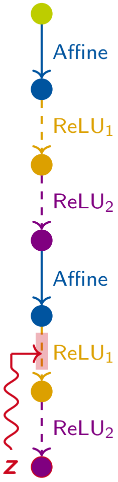
Heuristics:

- Reuse previous counterexamples
- Reduce the representation $z_i = \frac{n_i}{d_i}$

Outline

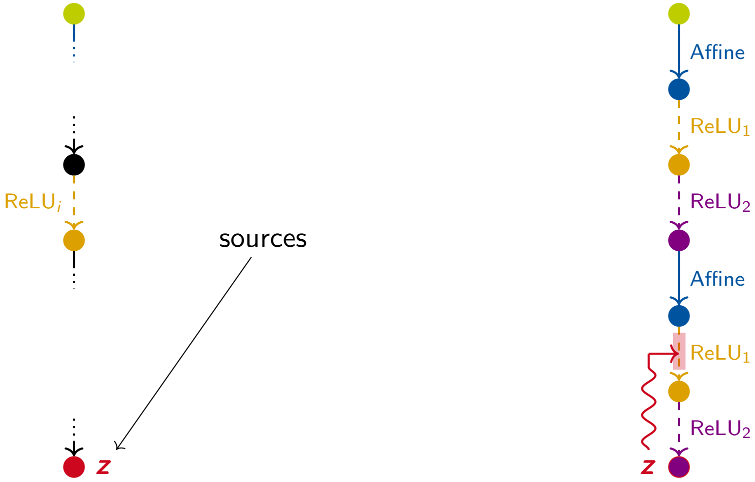
1. Preliminaries
   - Feedforward Neural Networks
   - Reachability Analysis
2. Verification with CEGAR
   - Safety Checking
   - Tracing a Counterexample
   - Refinement Heuristic
3. Performance
4. Conclusion

Origins of Counterexamples

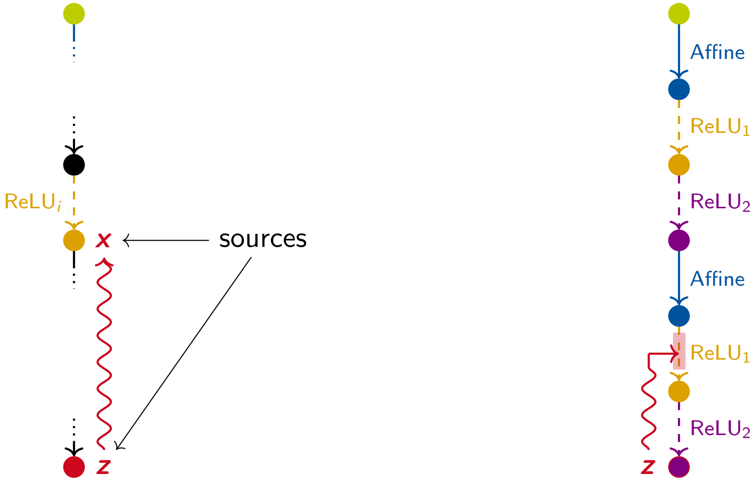


Origins of Counterexamples

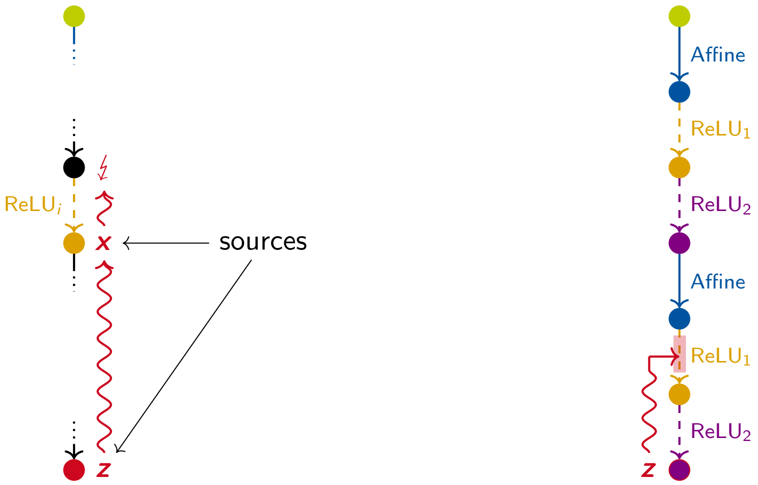
Origins of Counterexamples



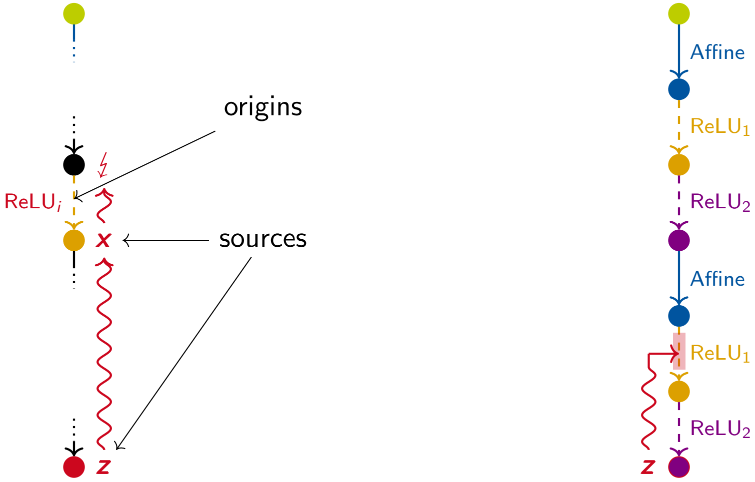
Origins of Counterexamples



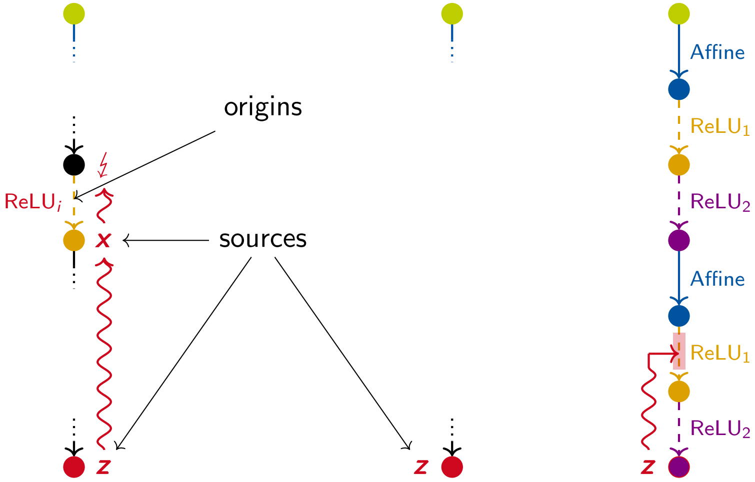
Origins of Counterexamples



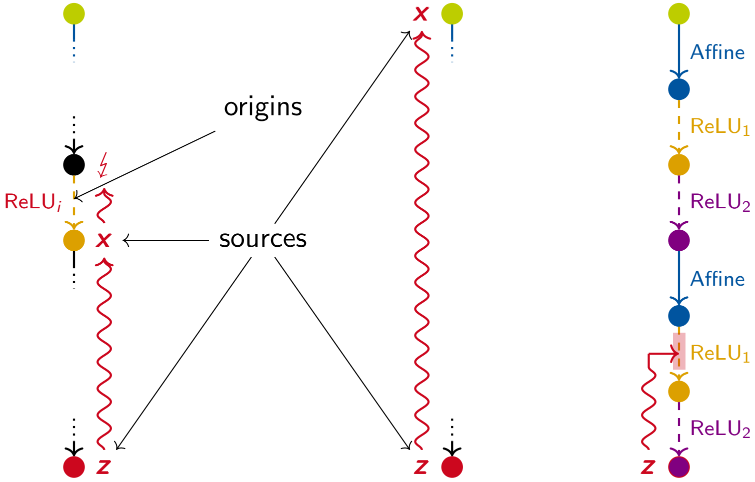
Origins of Counterexamples



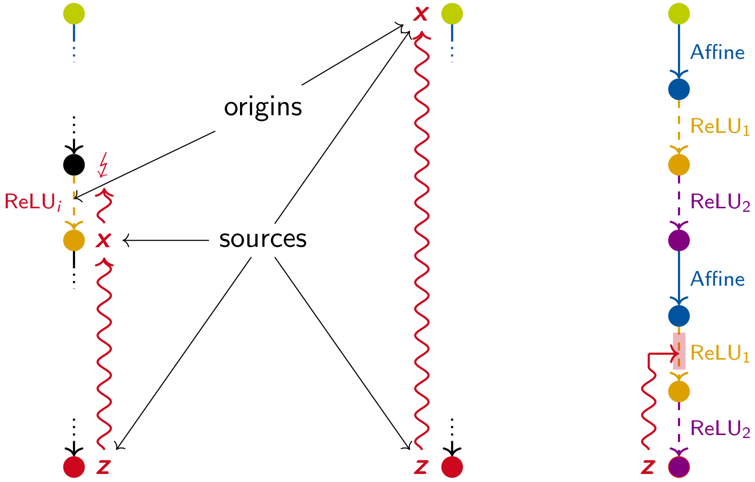
Origins of Counterexamples



Origins of Counterexamples

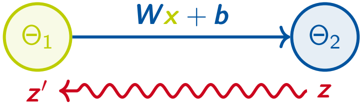


Origins of Counterexamples



Affine Tracing

Affine Tracing



Affine Tracing

$$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P} \rangle = \Theta_1 \xrightarrow{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}} \Theta_2 = \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P} \rangle$$

$$\boldsymbol{z}' \leftsquigarrow \boldsymbol{z}$$

Affine Tracing

$$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P} \rangle = \Theta_1 \xrightarrow{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}} \Theta_2 = \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P} \rangle$$

$$\boldsymbol{z}', \alpha' \leftsquigarrow \boldsymbol{z}, \alpha$$

Affine Tracing

$$\begin{tikzcd}
\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P} \rangle = \Theta_1 \arrow[r, "\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}"] & \Theta_2 = \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P} \rangle \\
\boldsymbol{z}', \alpha' & \boldsymbol{z}, \alpha \arrow[l, squiggly]
\end{tikzcd}$$

$$\boldsymbol{z}' = \boldsymbol{V}_1 \alpha' + \boldsymbol{c}_1 \qquad \boldsymbol{z} = \boldsymbol{V}_2 \alpha + \boldsymbol{c}_2$$

Affine Tracing

$$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P}\rangle = \Theta_1 \xrightarrow{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}} \Theta_2 = \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P}\rangle$$

$$\boldsymbol{z}', \alpha' \leftsquigarrow \boldsymbol{z}, \alpha$$

$\boldsymbol{z}' = \boldsymbol{V}_1\alpha' + \boldsymbol{c}_1$

$\boldsymbol{z} = \boldsymbol{V}_2\alpha + \boldsymbol{c}_2$

$$\begin{aligned} &\boldsymbol{z} \\ = \quad &\boldsymbol{V}_2\alpha + \boldsymbol{c}_2 \end{aligned}$$

Affine Tracing

$$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P}\rangle = \Theta_1 \xrightarrow{\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}} \Theta_2 \begin{aligned} &= \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P}\rangle \\ &= \langle \boldsymbol{W}\boldsymbol{c}_1 + \boldsymbol{b}, \boldsymbol{W}\boldsymbol{V}_1, \mathcal{P}\rangle \end{aligned}$$

$$\boldsymbol{z}', \alpha' \leftsquigarrow \boldsymbol{z}, \alpha$$

$$\boldsymbol{z}' = \boldsymbol{V}_1\alpha' + \boldsymbol{c}_1 \qquad \boldsymbol{z} = \boldsymbol{V}_2\alpha + \boldsymbol{c}_2$$

$$\begin{aligned} &\boldsymbol{z} \\ = \quad &\boldsymbol{V}_2\alpha + \boldsymbol{c}_2 \end{aligned}$$

Affine Tracing

$$
\begin{tikzcd}
\langle \mathbf{c}_1, \mathbf{V}_1, \mathcal{P} \rangle = \Theta_1 \arrow[r, "\mathbf{W}\mathbf{x} + \mathbf{b}"] & \Theta_2 = \langle \mathbf{c}_2, \mathbf{V}_2, \mathcal{P} \rangle = \langle \mathbf{W}\mathbf{c}_1 + \mathbf{b}, \mathbf{W}\mathbf{V}_1, \mathcal{P} \rangle \\
\mathbf{z}', \alpha' & \mathbf{z}, \alpha \arrow[l, squiggly]
\end{tikzcd}
$$

$$\mathbf{z}' = \mathbf{V}_1 \alpha' + \mathbf{c}_1 \qquad \mathbf{z} = \mathbf{V}_2 \alpha + \mathbf{c}_2$$

$$
\begin{aligned}
& \mathbf{z} \\
= \quad & \mathbf{V}_2 \alpha + \mathbf{c}_2 \\
= \quad & \mathbf{W}\mathbf{V}_1 \alpha + \mathbf{W}\mathbf{c}_1 + \mathbf{b}
\end{aligned}
$$

Affine Tracing

$$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P}\rangle = \Theta_1 \xrightarrow{\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}} \Theta_2 = \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P}\rangle = \langle \boldsymbol{W}\boldsymbol{c}_1+\boldsymbol{b}, \boldsymbol{W}\boldsymbol{V}_1, \mathcal{P}\rangle$$

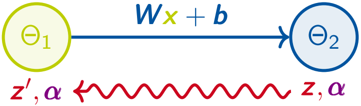
$$\boldsymbol{z}', \alpha' \leftsquigarrow \boldsymbol{z}, \alpha$$

$$\boldsymbol{z}' = \boldsymbol{V}_1\alpha' + \boldsymbol{c}_1 \qquad \boldsymbol{z} = \boldsymbol{V}_2\alpha + \boldsymbol{c}_2$$

$$\begin{aligned} &\boldsymbol{z} \\ = \quad & \boldsymbol{V}_2\alpha + \boldsymbol{c}_2 \\ = \quad & \boldsymbol{W}\boldsymbol{V}_1\alpha + \boldsymbol{W}\boldsymbol{c}_1 + \boldsymbol{b} \\ = \quad & \boldsymbol{W}\left(\boldsymbol{V}_1\alpha + \boldsymbol{c}_1\right) + \boldsymbol{b} \end{aligned}$$

Affine Tracing

$\langle \boldsymbol{c}_1, \boldsymbol{V}_1, \mathcal{P} \rangle =$



$= \langle \boldsymbol{c}_2, \boldsymbol{V}_2, \mathcal{P} \rangle$

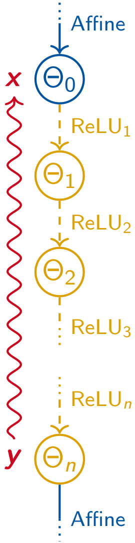
$= \langle \boldsymbol{W}\boldsymbol{c}_1 + \boldsymbol{b}, \boldsymbol{W}\boldsymbol{V}_1, \mathcal{P} \rangle$

$\boldsymbol{z}' = \boldsymbol{V}_1 \alpha + \boldsymbol{c}_1$

$\boldsymbol{z} = \boldsymbol{V}_2 \alpha + \boldsymbol{c}_2$

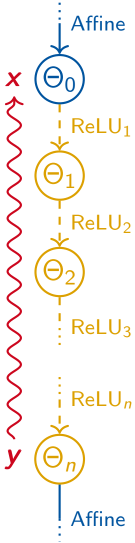
$$
\begin{aligned}
& \boldsymbol{z} \\
= \quad & \boldsymbol{V}_2 \alpha + \boldsymbol{c}_2 \\
= \quad & \boldsymbol{W}\boldsymbol{V}_1 \alpha + \boldsymbol{W}\boldsymbol{c}_1 + \boldsymbol{b} \\
= \quad & \boldsymbol{W}\left(\boldsymbol{V}_1 \alpha + \boldsymbol{c}_1\right) + \boldsymbol{b} \\
= \quad & \boldsymbol{W}\boldsymbol{z}' + \boldsymbol{b}
\end{aligned}
$$

Tracing for ReLU



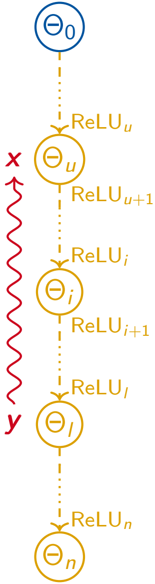
Tracing for ReLU

$$\boldsymbol{x} \in \Theta_0 \wedge \bigwedge_{i=1}^{n} \mathrm{ReLU}(x_i) = y_i$$



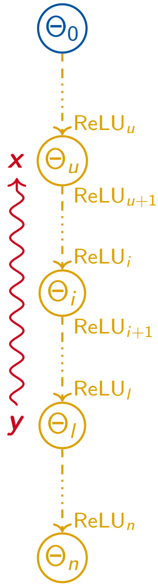
Tracing for ReLU

$$\boldsymbol{x} \in \Theta_u \wedge \bigwedge_{j=1}^{u} x_j = y_j \wedge \bigwedge_{i=u+1}^{l} \mathrm{ReLU}(x_i) = y_i \wedge \bigwedge_{j=l+1}^{n} x_j = y_j$$



Tracing for ReLU

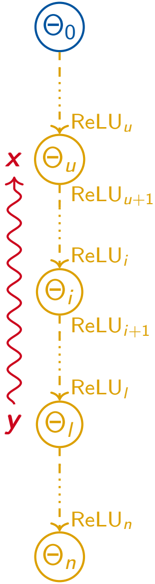
$$\boldsymbol{x} \in \Theta_u \wedge \bigwedge_{j=1}^{u} x_j = y_j \wedge \bigwedge_{i=u+1}^{l} \underbrace{\mathrm{ReLU}(x_i) = y_i}_{\varphi_i} \wedge \bigwedge_{j=l+1}^{n} x_j = y_j$$



UNSAT Core: An already unsatisfiable subformula

Tracing for ReLU

$$\boldsymbol{x} \in \Theta_u \wedge \bigwedge_{j=1}^{u} x_j = y_j \wedge \bigwedge_{i=u+1}^{l} \underbrace{\mathrm{ReLU}(x_i) = y_i}_{\varphi_i} \wedge \bigwedge_{j=l+1}^{n} x_j = y_j$$



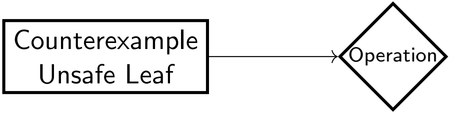
<u>UNSAT Core</u>: An already unsatisfiable subformula

For our method: Smallest i with φ_i in the UNSAT core

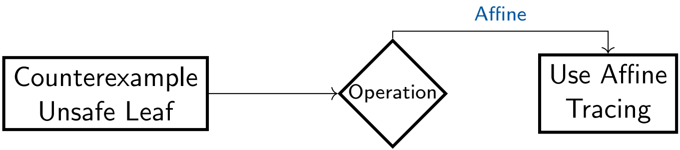
UNSAT Core Based Tracing

Counterexample
Unsafe Leaf

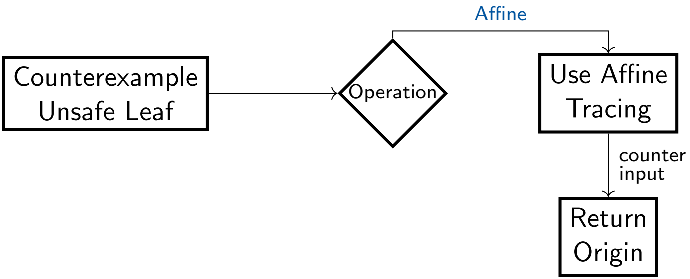
UNSAT Core Based Tracing



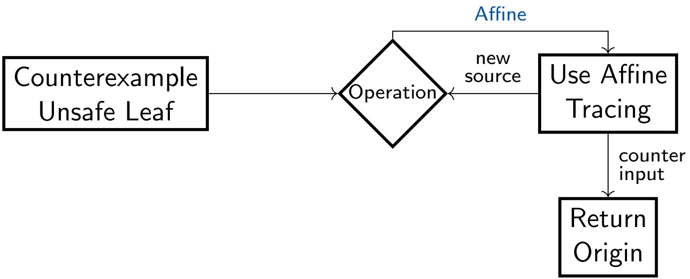
UNSAT Core Based Tracing



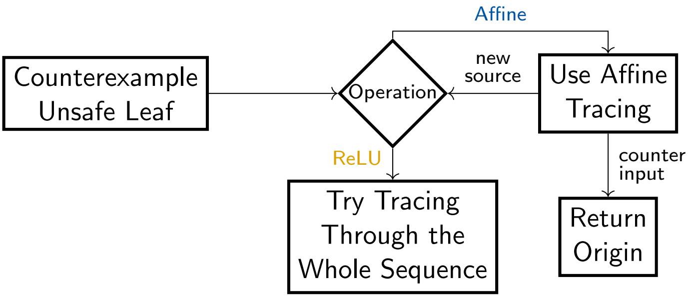
UNSAT Core Based Tracing



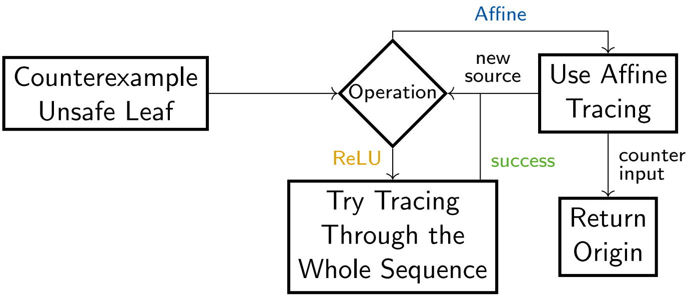
UNSAT Core Based Tracing



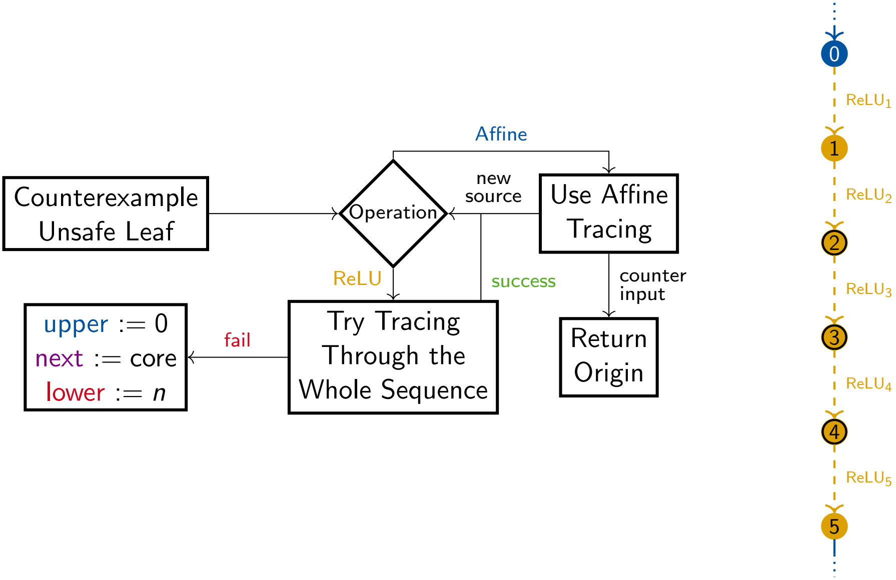
UNSAT Core Based Tracing



UNSAT Core Based Tracing

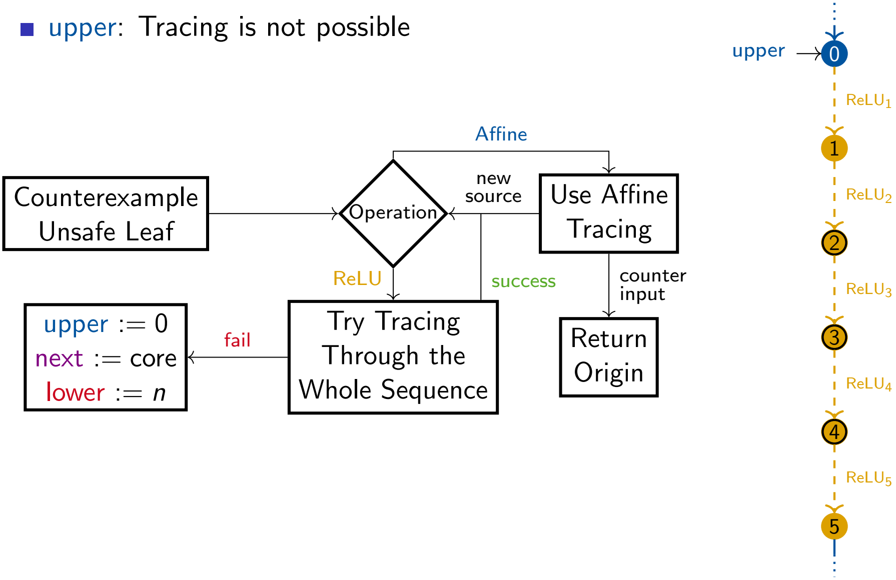


UNSAT Core Based Tracing



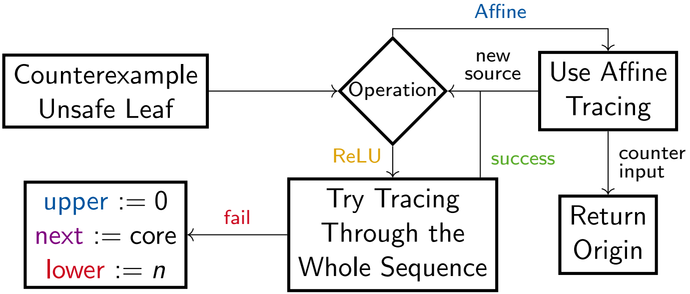
UNSAT Core Based Tracing

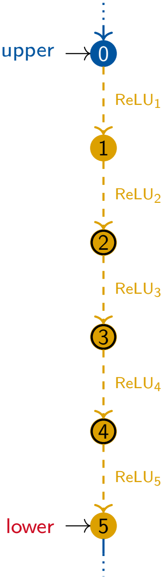
- upper: Tracing is not possible



UNSAT Core Based Tracing

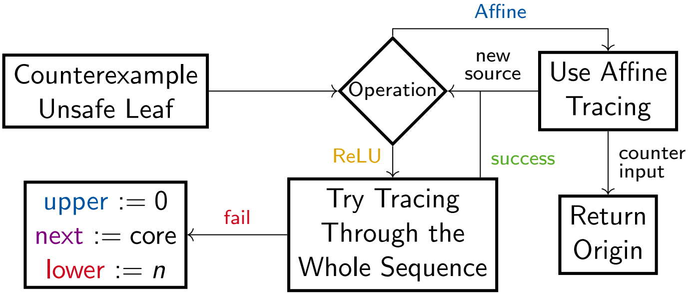
- upper: Tracing is not possible
- lower: Tracing is possible

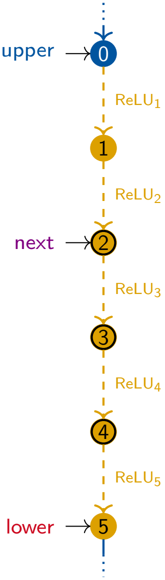




UNSAT Core Based Tracing

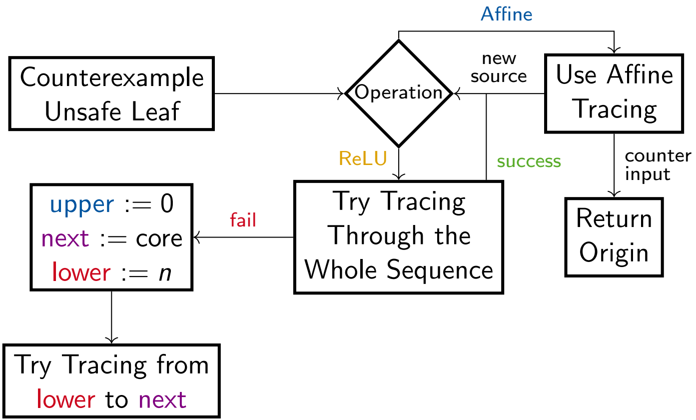
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

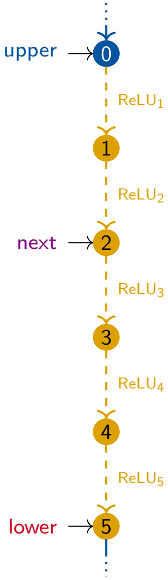




UNSAT Core Based Tracing

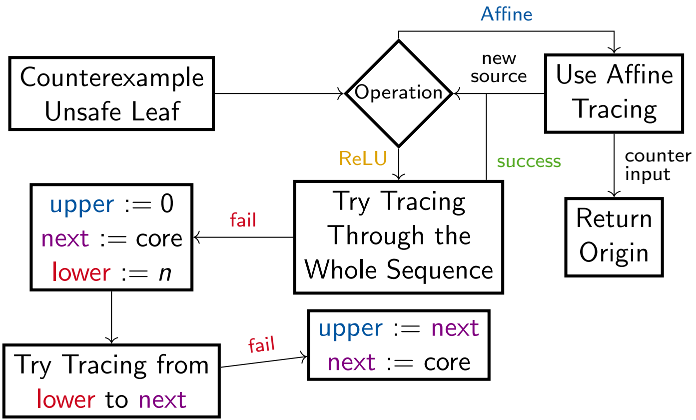
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

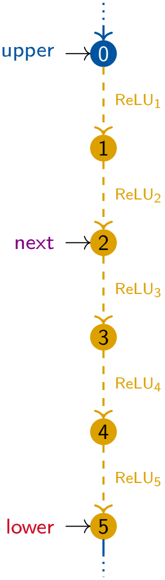




UNSAT Core Based Tracing

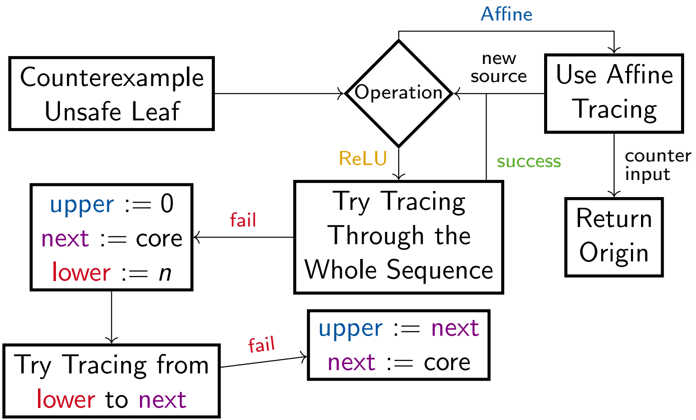
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

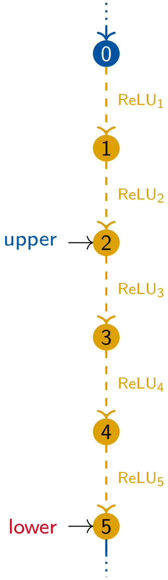




UNSAT Core Based Tracing

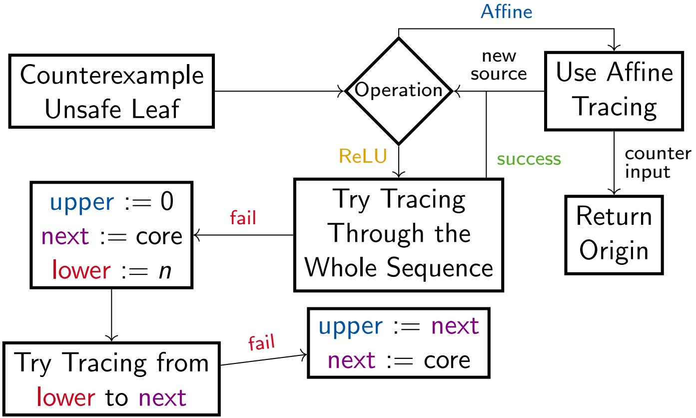
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

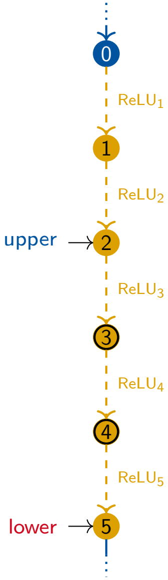




UNSAT Core Based Tracing

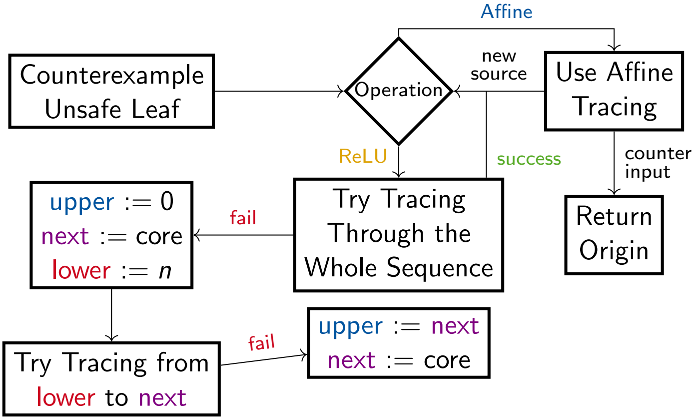
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

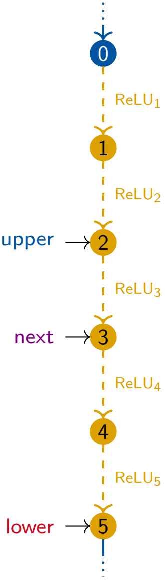




UNSAT Core Based Tracing

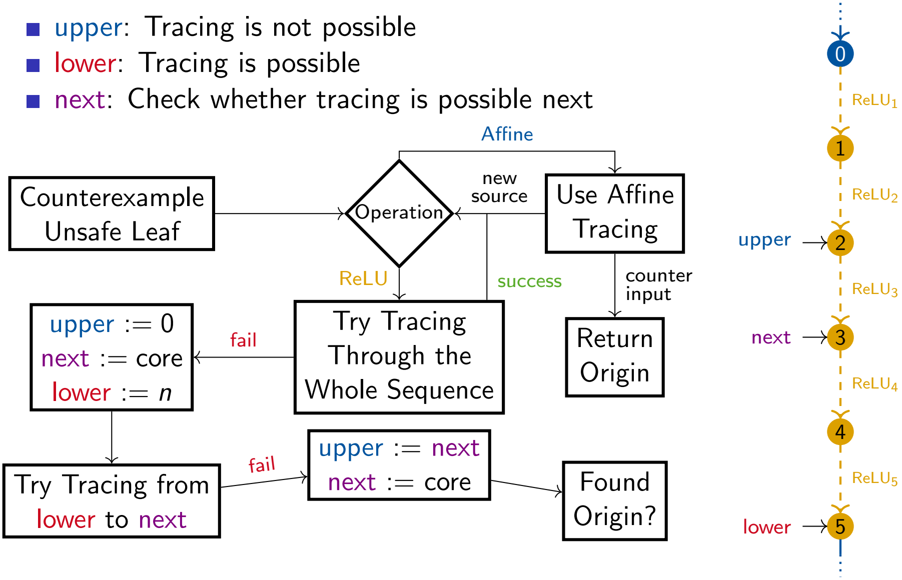
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next





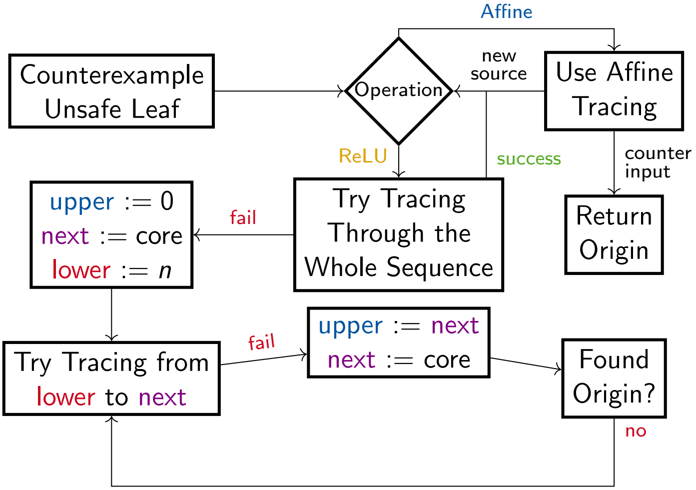
UNSAT Core Based Tracing

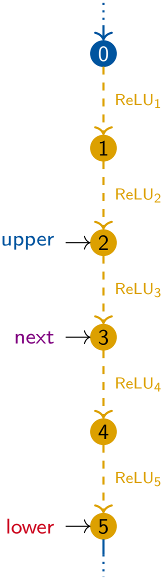
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next



UNSAT Core Based Tracing

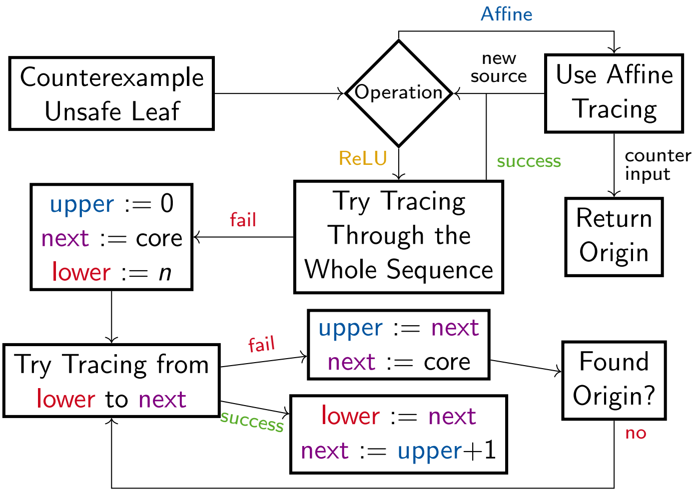
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

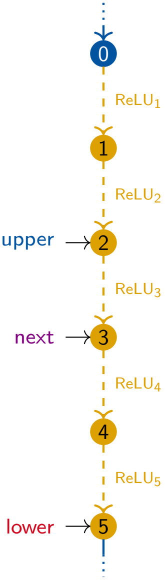




UNSAT Core Based Tracing

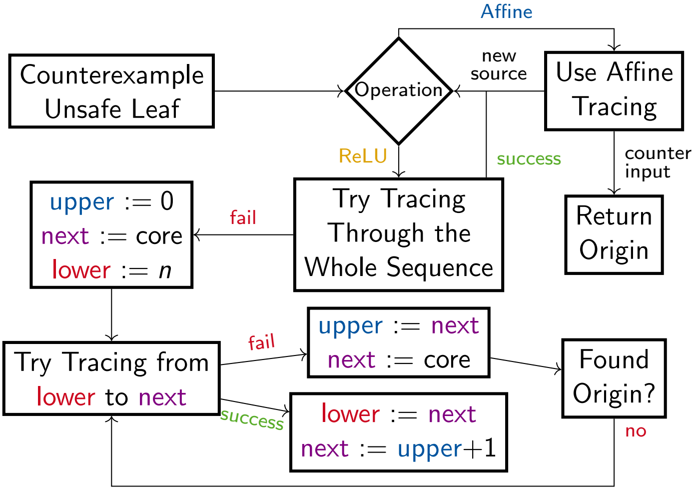
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next

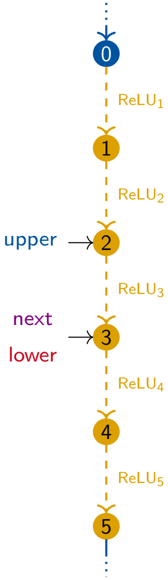




UNSAT Core Based Tracing

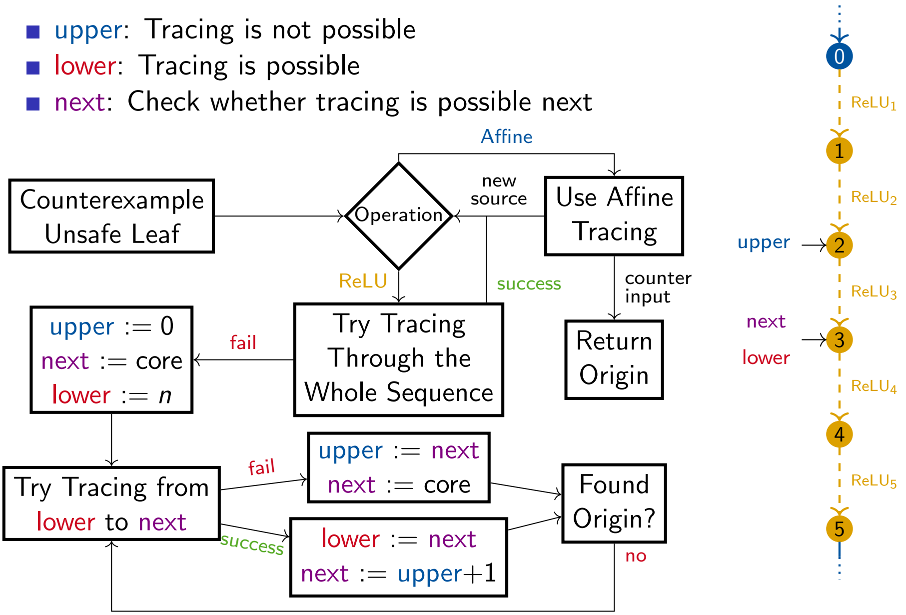
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next





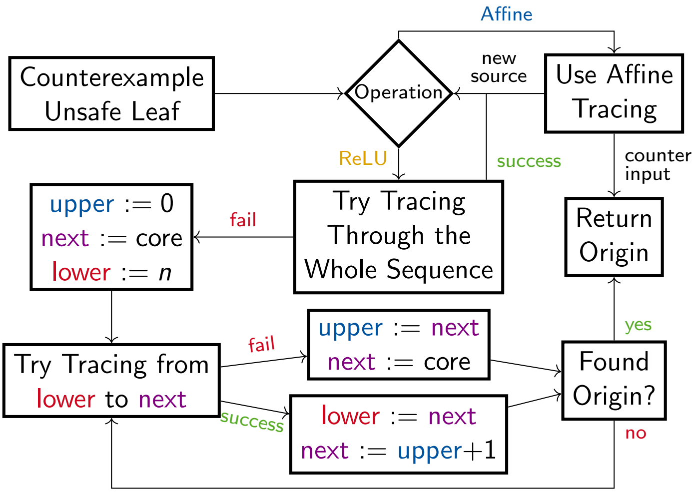
UNSAT Core Based Tracing

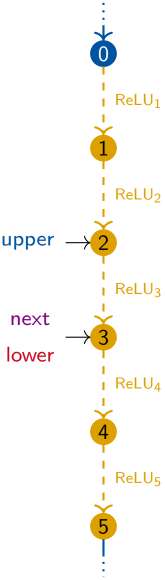
- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next



UNSAT Core Based Tracing

- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next





Tracing Methods

Goal: Identification of an origin

Tracing Methods

Goal: Identification of an origin

Methods:

- Tracing based on UNSAT cores

Tracing Methods

Goal: Identification of an origin

Methods:

- Tracing based on UNSAT cores
- Repeatedly split the search space in half

Tracing Methods

Goal: Identification of an origin

Methods:

- Tracing based on UNSAT cores
- Repeatedly split the search space in half
- Check previous origins first

Tracing Methods

Goal: Identification of an origin

Methods:

- Tracing based on UNSAT cores
- Repeatedly split the search space in half
- Check previous origins first
- Check if the parent node contains a new source

Outline

Refinement Methods

<u>Goal</u>: Construct and refine reachability trees to prove (un-)safety

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary
- Construct all final leaves and refine if necessary

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins
- Always use the exact method for previous origins

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins
- Always use the exact method for previous origins
- Use the safety of subtrees to reduce computation

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- Construct one additional final leaf and refine if necessary
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins
- Always use the exact method for previous origins
- Use the safety of subtrees to reduce computation (no implementation)

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- **Construct one additional final leaf and refine if necessary**
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins
- **Always use the exact method for previous origins**
- Use the safety of subtrees to reduce computation (no implementation)

Refinement Methods

Goal: Construct and refine reachability trees to prove (un-)safety

Methods:

- **Construct one additional final leaf and refine if necessary**
- Construct all final leaves and refine if necessary

Heuristics:

- Check if the exact method should be used for previous origins
- **Always use the exact method for previous origins**
- *Use the safety of subtrees to reduce computation (no implementation)*

Refinement Heuristic: Exact Origins

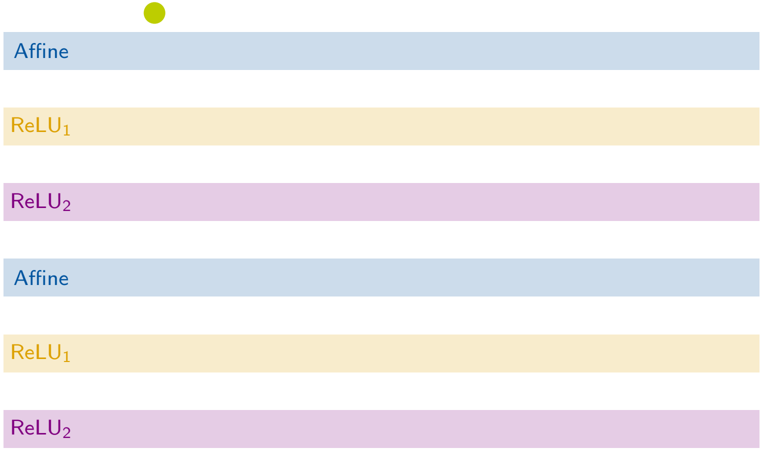
- Keep track of all previous origins

Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method

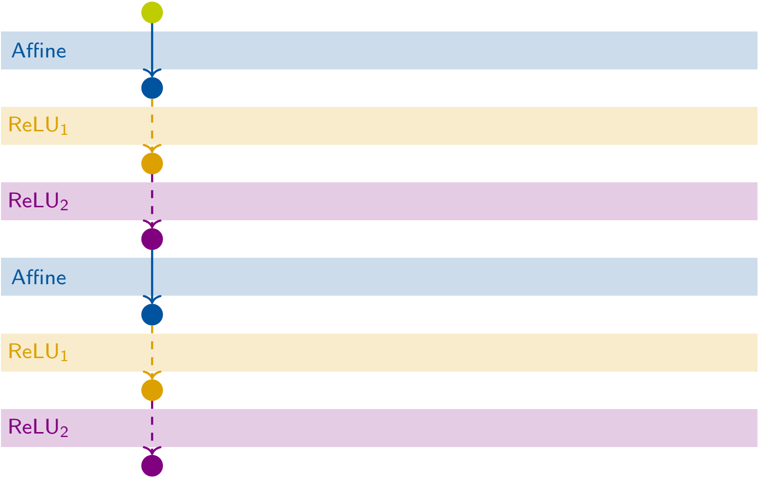
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



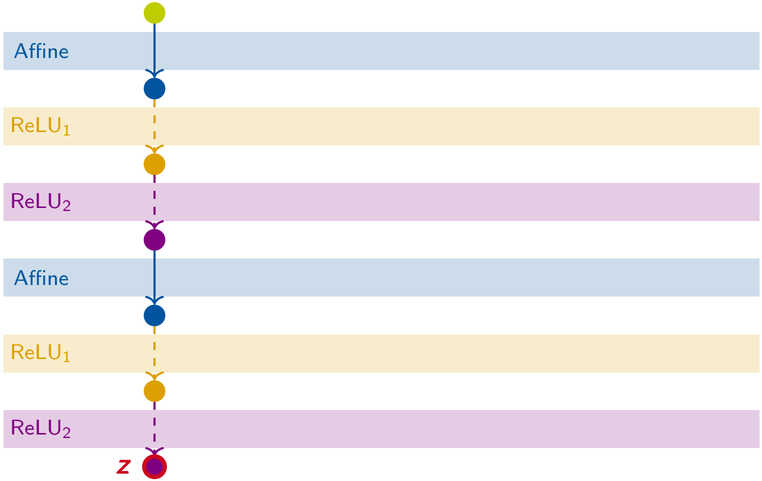
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



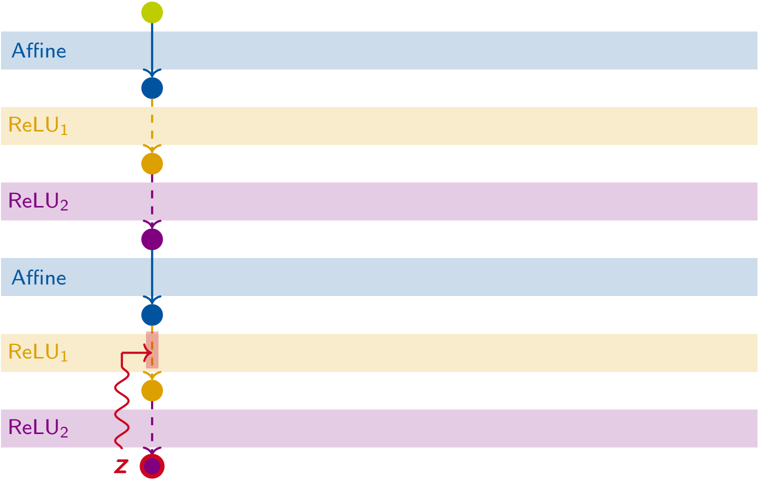
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



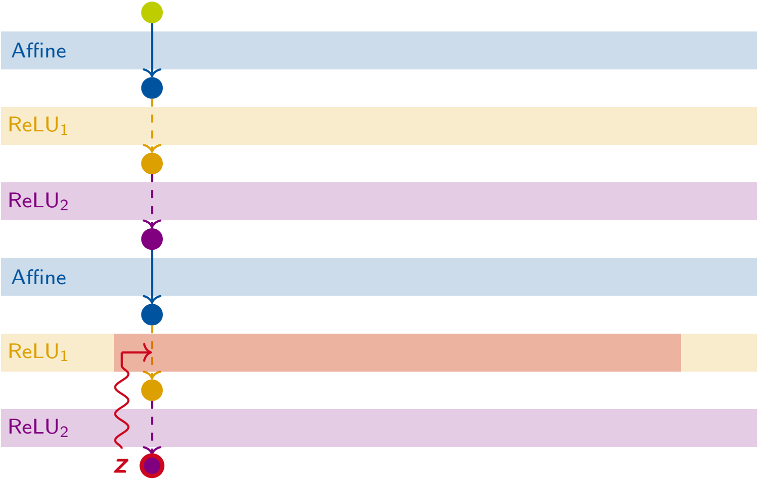
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



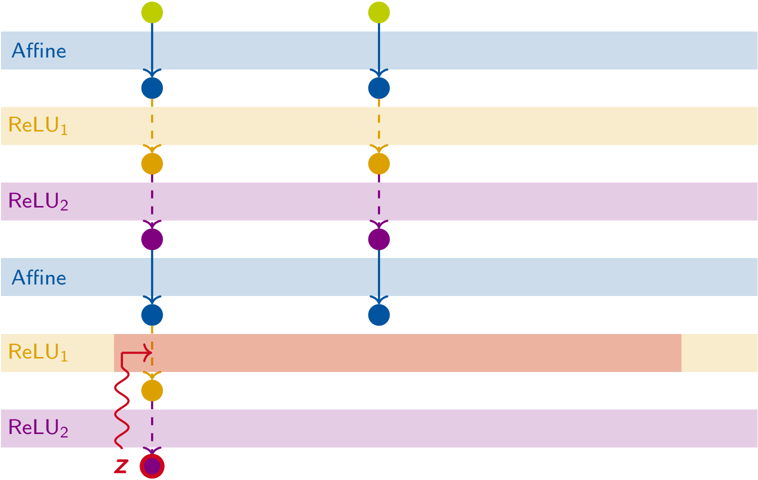
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



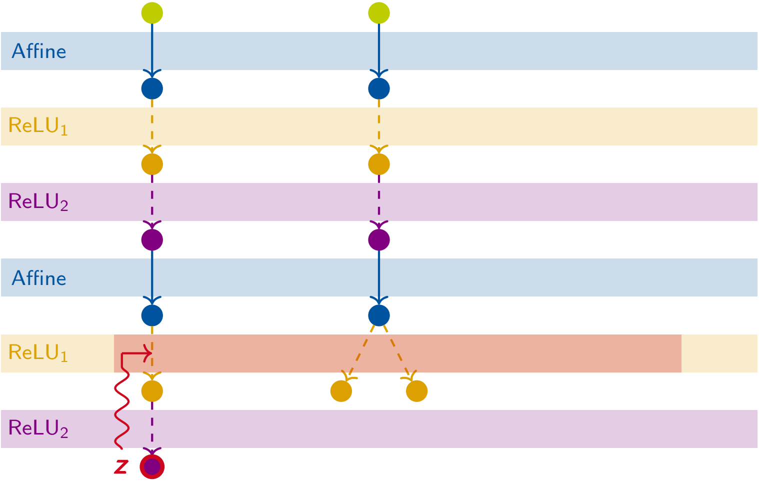
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



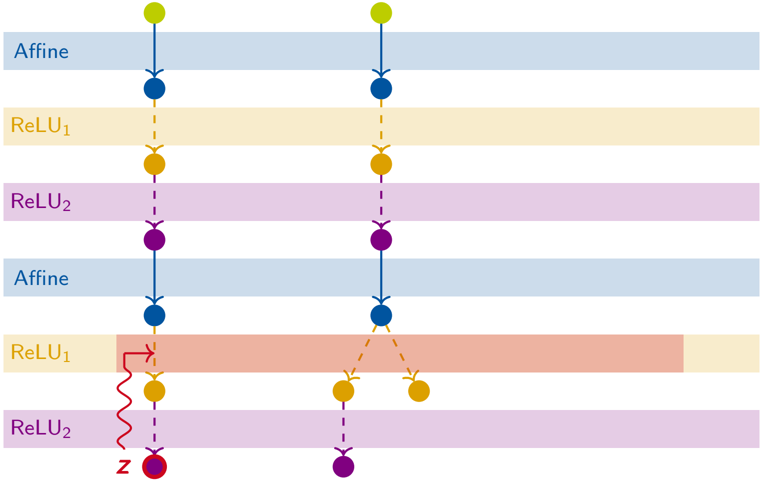
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



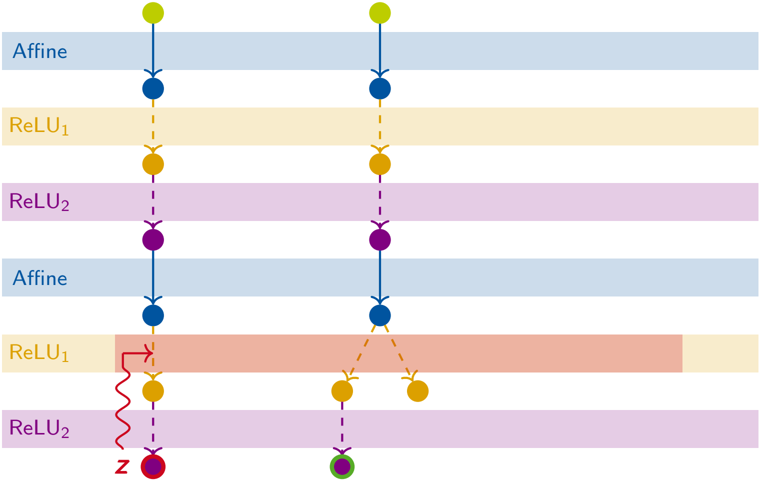
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



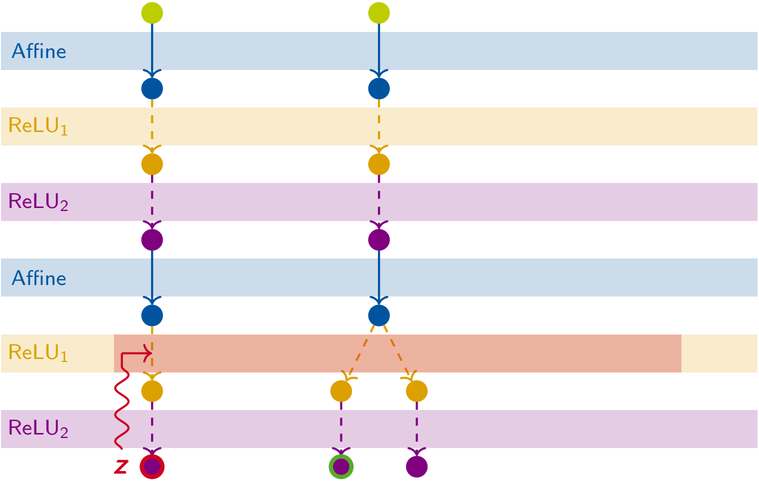
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



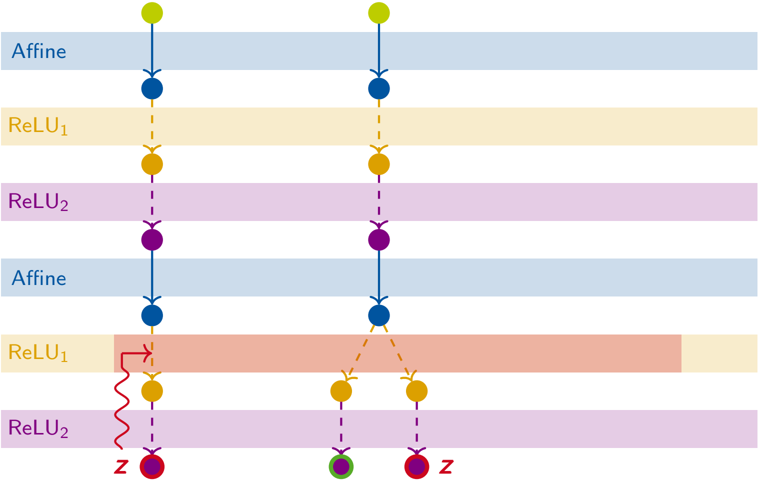
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



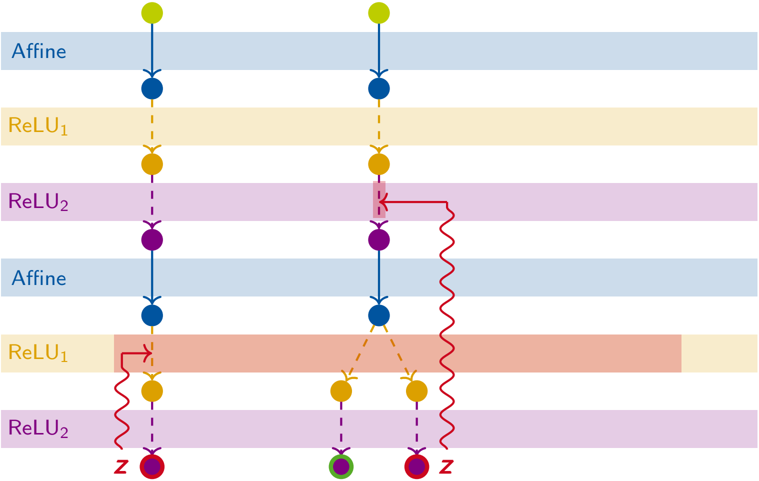
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



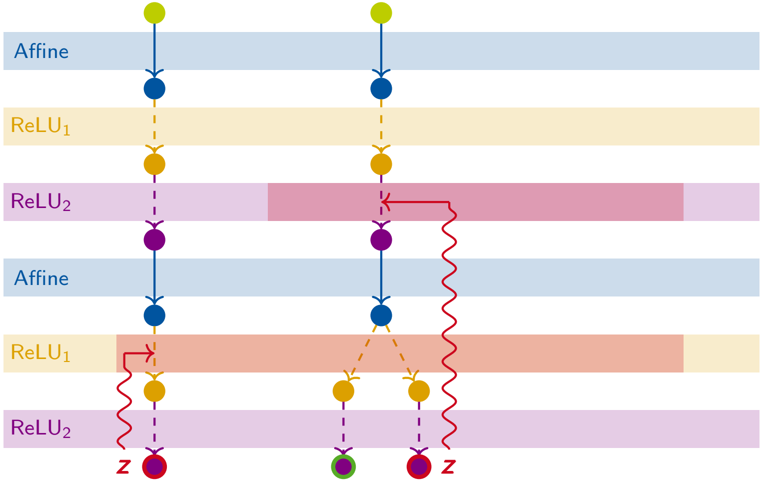
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



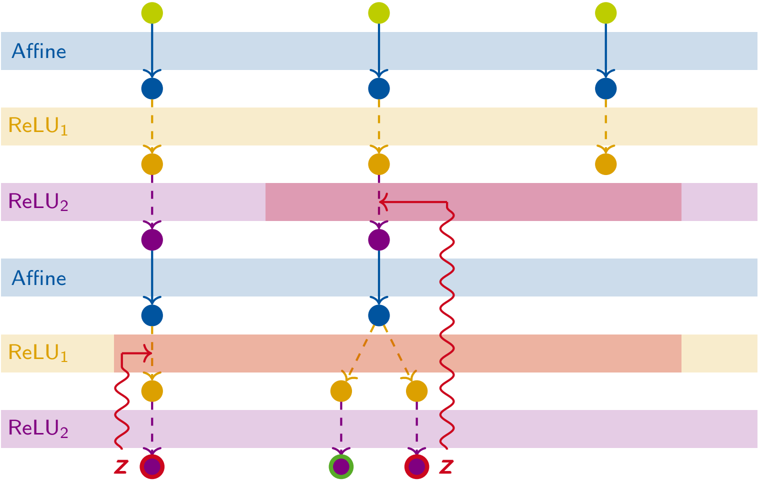
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



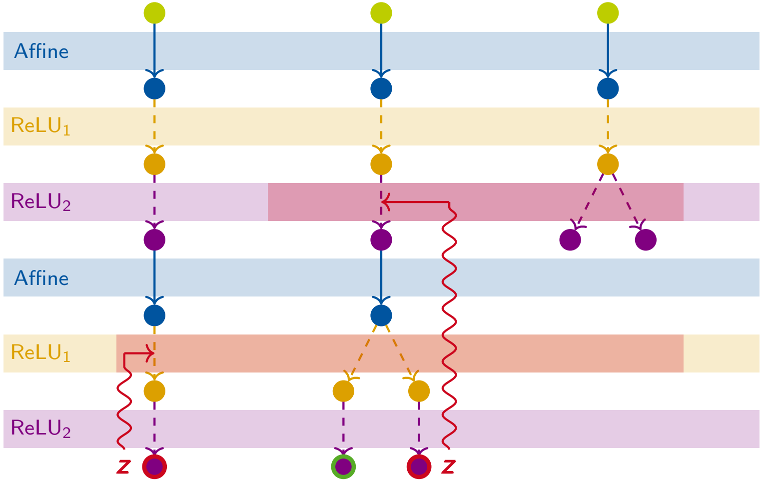
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



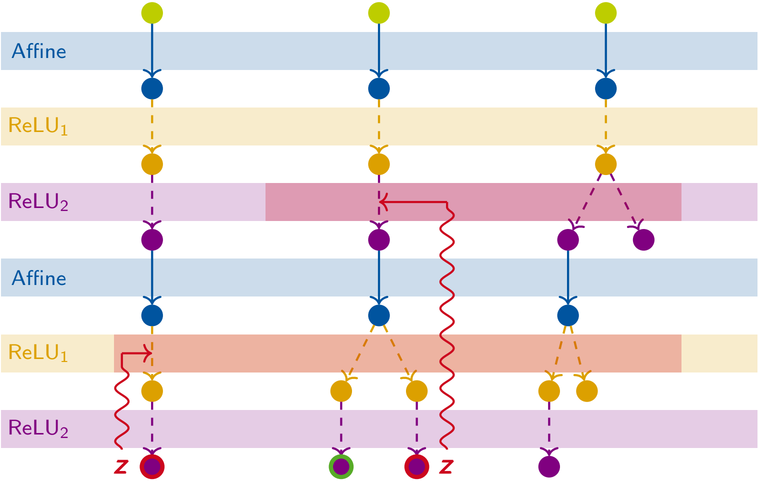
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



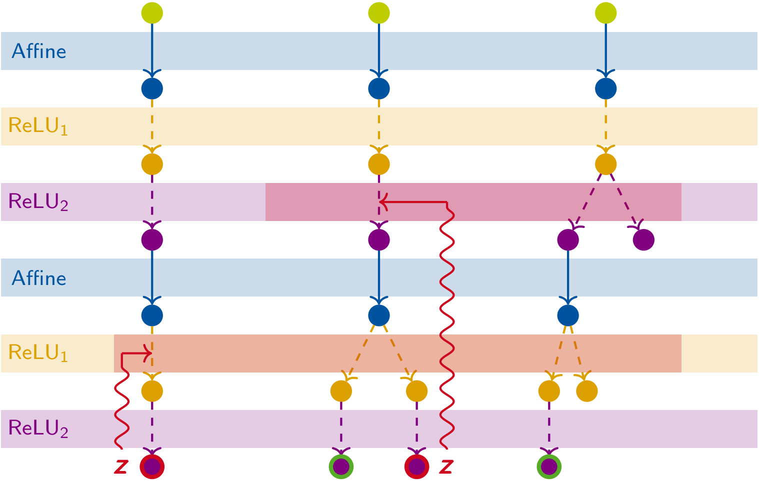
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



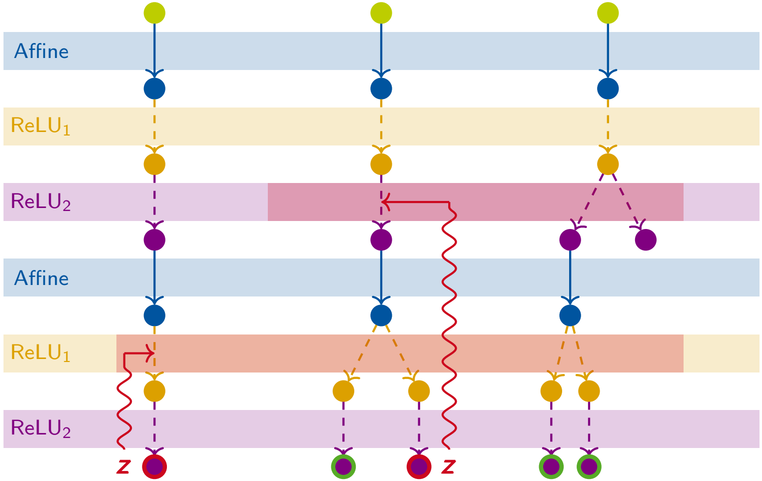
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



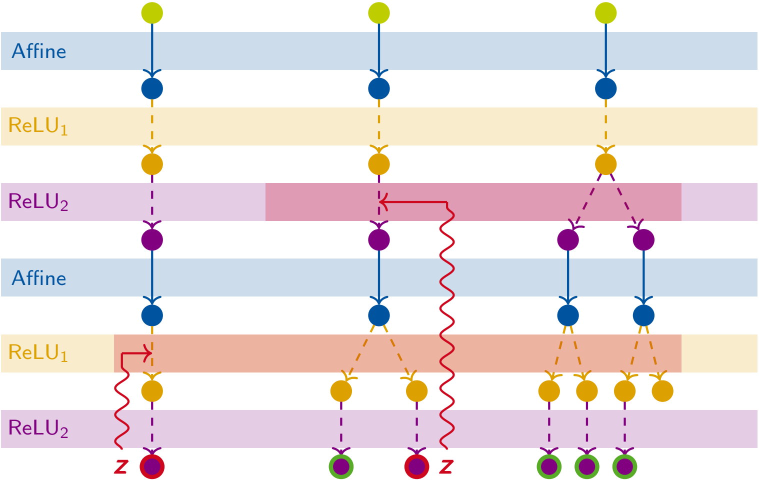
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



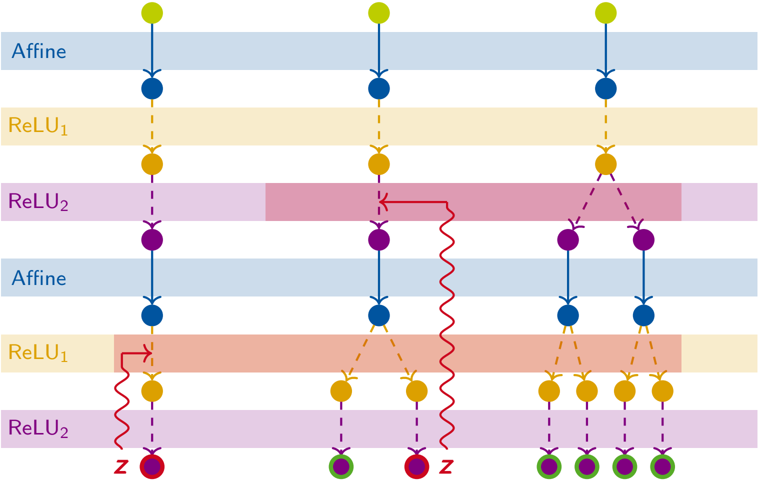
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



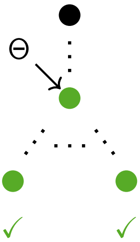
Refinement Heuristic: Exact Origins

- Keep track of all previous origins
- Always compute a previous origin with the exact method



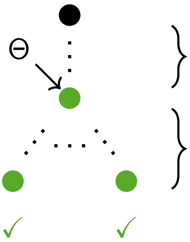
Safe History

Iteration i



Safe History

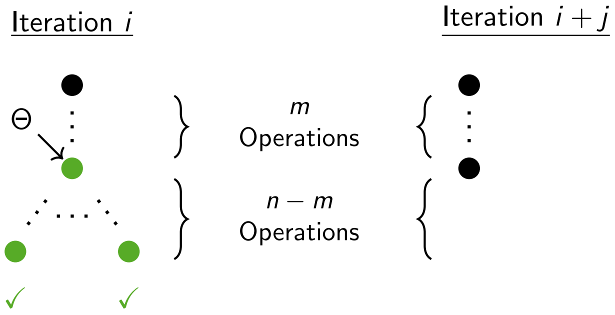
Iteration i



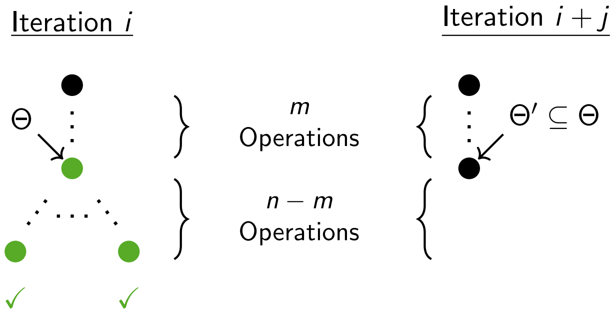
m
Operations

$n-m$
Operations

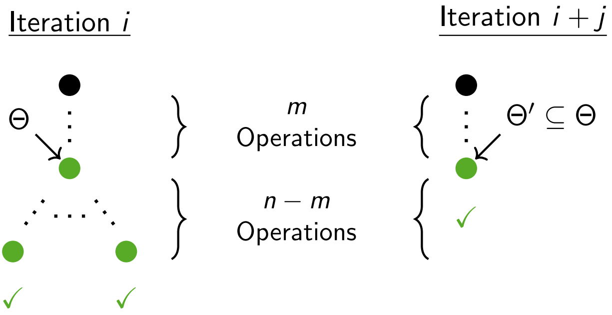
Safe History



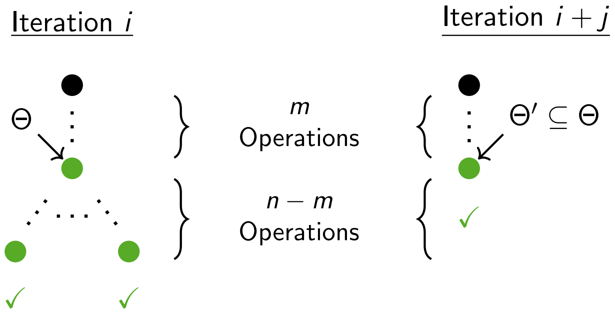
Safe History



Safe History



Safe History

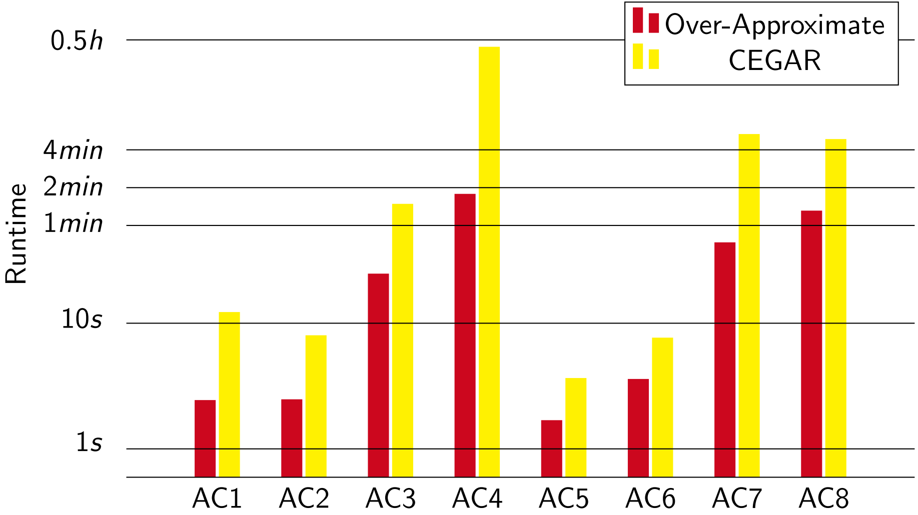


Proposition

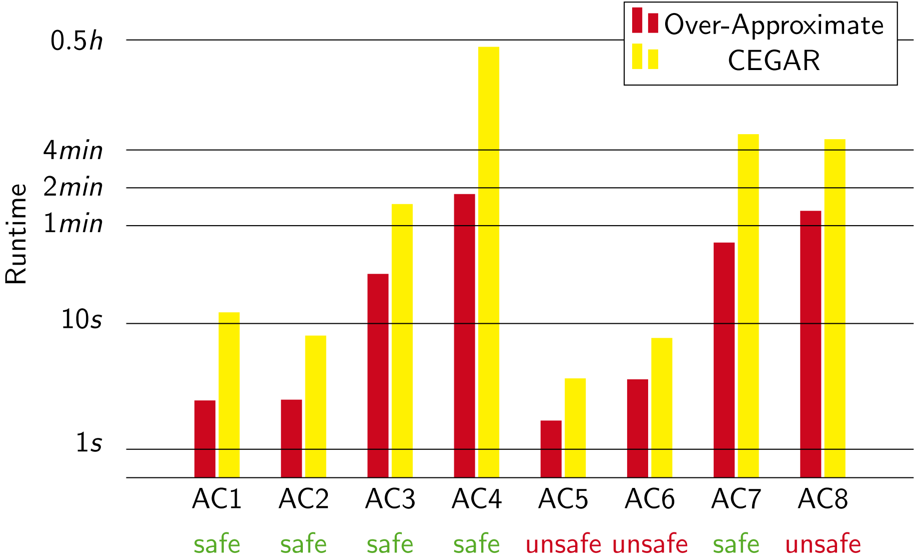
For two paths $\Theta_0, \ldots, \Theta_m$ and $\Theta'_0, \ldots, \Theta'_m$ in reachability trees corresponding to the same FNN with the same root $\Theta_0 = \Theta'_0$, $\Theta'_m \subseteq \Theta_m$ if the operations applied in the first path are at least as tight as the operations in the second path.

Outline

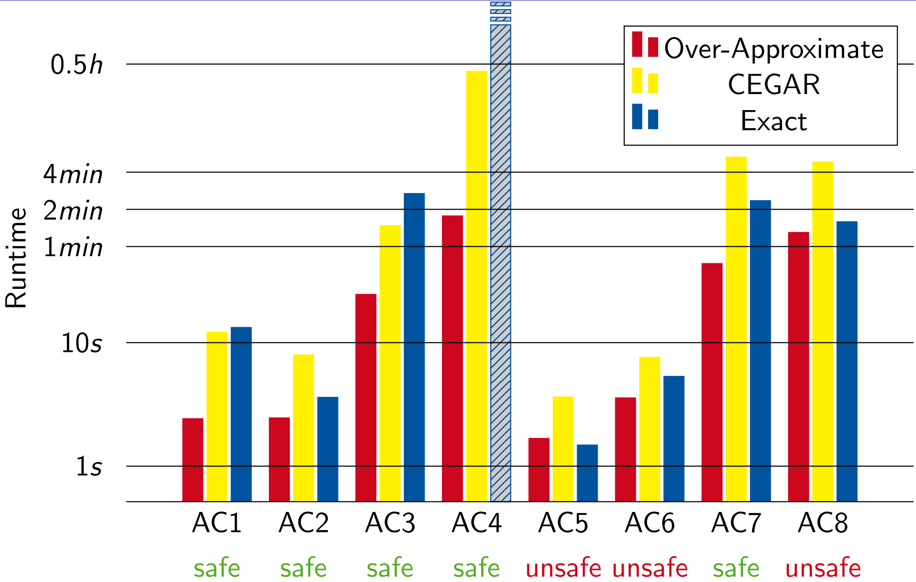
Drones Benchmark



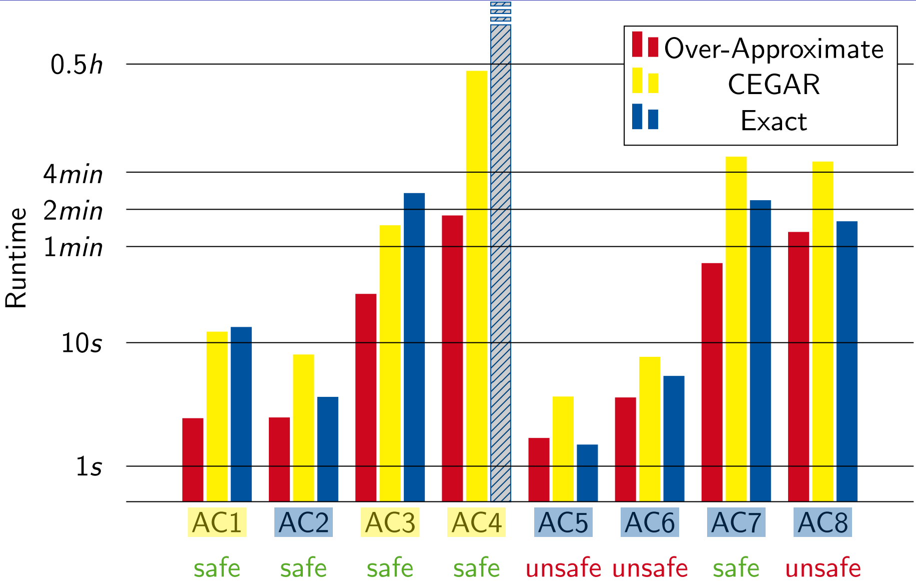
Drones Benchmark



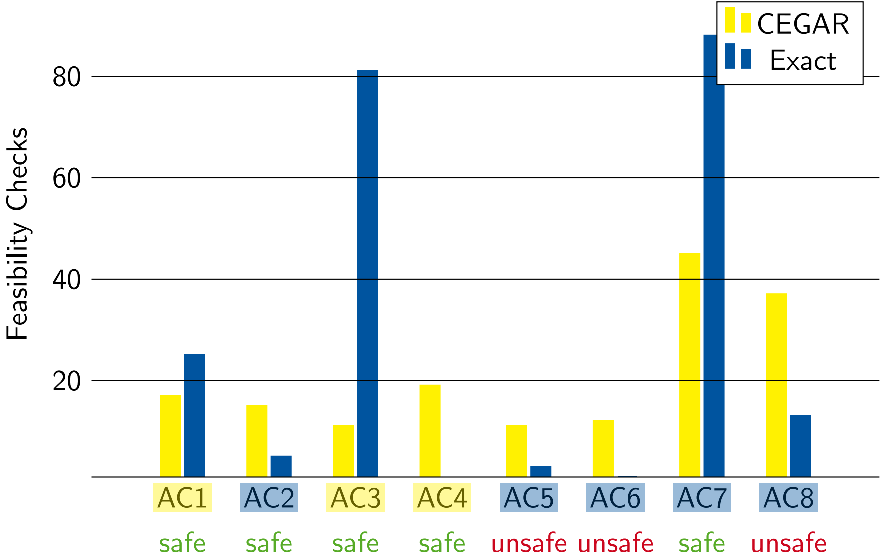
Drones Benchmark



Drones Benchmark



Drones Benchmark



Conclusion

A combination of exact and over-approximate reachability analysis

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin
4) Refine: Apply more exact computation

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin
4) Refine: Apply more exact computation

<u>Conclusion</u>

- Verification with CEGAR is complete

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin
4) Refine: Apply more exact computation

<u>Conclusion</u>

- Verification with CEGAR is complete
- CEGAR can be faster than exact computation

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin
4) Refine: Apply more exact computation

<u>Conclusion</u>

- Verification with CEGAR is complete
- CEGAR can be **faster** than exact computation, but also **slower**

Conclusion

A combination of exact and over-approximate reachability analysis

1) Highly over-approximate reachability analysis
2) Identify a counterexample
3) Trace it to an origin
4) Refine: Apply more exact computation

<u>Conclusion</u>

- Verification with CEGAR is complete
- CEGAR can be **faster** than exact computation, but also **slower**
- Reason: Amount and complexity of feasibility problems

Stars [Tran et al., 2019]

An (n, m)-dimensional *star* is a tuple $\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, P \rangle$ with

- $\boldsymbol{c} \in \mathbb{R}^n$
- $\boldsymbol{V} \in \mathbb{R}^{n \times m}$
- $P = \{\boldsymbol{\alpha} \in \mathbb{R}^m \mid \boldsymbol{A} \cdot \boldsymbol{\alpha} \leq \boldsymbol{b}\}$

Representing the set $\{\boldsymbol{c} + \boldsymbol{V}\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in P\}$

Affine Mapping with $\boldsymbol{W} \in \mathbb{R}^{k \times n}$ and $\boldsymbol{b} \in \mathbb{R}^k$:

$$\boldsymbol{W}\Theta + \boldsymbol{b} = \langle \boldsymbol{W}\boldsymbol{c} + \boldsymbol{b}, \boldsymbol{W}\boldsymbol{V}, P \rangle$$

Intersection with halfspace $\mathcal{H} = \left\{\boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \boldsymbol{h}^T\boldsymbol{x} \leq d\right\}$:

$$\Theta \cap \mathcal{H} = \left\langle \boldsymbol{c}, \boldsymbol{V}, P \cap \left\{\boldsymbol{\alpha} \in \mathbb{R}^m \;\middle|\; \boldsymbol{h}^T\left(\boldsymbol{V}\boldsymbol{\alpha} + \boldsymbol{c}\right) \leq d\right\}\right\rangle$$

Bounds of Θ in dimension i:

$$c_i + \min\left\{\boldsymbol{V}_{\_,i}\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in P\right\} \qquad c_i + \max\left\{\boldsymbol{V}_{\_,i}\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in P\right\}$$

Exact ReLU Operation

Input: Star $\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, \mathcal{P} \rangle$, activation function act, and dimension j

Extract the calculation to apply:

1) Halfspaces $\mathcal{H}_{1,1}, \dots, \mathcal{H}_{1,k_1}$ and affine mapping $\boldsymbol{W}_1 + \boldsymbol{b}_1$

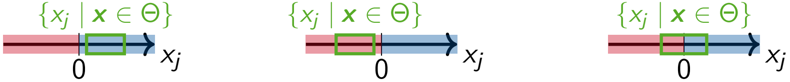
$\vdots$

k) Halfspaces $\mathcal{H}_{k,1}, \dots, \mathcal{H}_{k,k_k}$ and affine mapping $\boldsymbol{W}_k + \boldsymbol{b}_k$

Output: $\Theta_1, \dots, \Theta_k$ with $\Theta_i := \boldsymbol{W}_i \left(\Theta \cap \bigcap_{\iota=1}^{k_i} \mathcal{H}_{i,\iota} \right) + \boldsymbol{b}_i$

$$\text{ReLU} : \mathbb{R} \to \mathbb{R}, x \mapsto \begin{cases} x & , x \geq 0 \\ 0 & , x < 0 \end{cases}$$

$$\boldsymbol{I} = \begin{pmatrix} \boldsymbol{e}_1 & \cdots & \boldsymbol{e}_j & \cdots & \boldsymbol{e}_{\boldsymbol{n}} \end{pmatrix} \qquad \boldsymbol{M} = \begin{pmatrix} \boldsymbol{e}_1 & \cdots & 0 & \cdots & \boldsymbol{e}_{\boldsymbol{n}} \end{pmatrix}$$



$\boldsymbol{I}\Theta + \boldsymbol{0}$

$\boldsymbol{M}\Theta + \boldsymbol{0}$

$\boldsymbol{I}(\Theta \cap (x_j \geq 0)) + \boldsymbol{0}$

$\boldsymbol{M}(\Theta \cap (x_j \leq 0)) + \boldsymbol{0}$

Over-Approximate Operation

Input: Star $\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, \mathcal{P} \rangle$, activation function act, and dimension j
Method:

(a) Apply exact method if no split is necessary
(b) Extract convex relaxation and apply to Θ

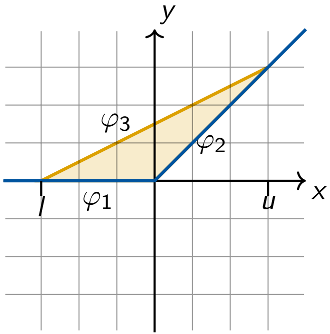
Output for (b): $\Theta' = \langle \boldsymbol{Mc} + \boldsymbol{b}, (\boldsymbol{MV} \quad \boldsymbol{e}_j), \mathcal{P}' \rangle$ where

- $\mathcal{P} = \{\boldsymbol{\alpha} \in \mathbb{R}^m \mid \psi_{\mathcal{P}}(\boldsymbol{\alpha})\}$
- $\mathcal{P}' = \left\{\boldsymbol{\alpha}' \in \mathbb{R}^{m+1} \;\middle|\; \psi_{\mathcal{P}}(\boldsymbol{\alpha}') \bigwedge_{i=1}^{k} \psi_i(\boldsymbol{\alpha}')\right\}$
- $\psi_{\mathcal{P}}$ is a formula defining $\mathcal{P}$ over variables $\alpha_1, \ldots, \alpha_m$
- $\varphi_1, \ldots, \varphi_k$ defines the convex relaxation over variables $\alpha_1, \ldots, \alpha_m, \alpha_{m+1}$

Convex Relaxation for ReLU in Dimension j

$$\Theta = \langle \boldsymbol{c}, \boldsymbol{V}, \{\boldsymbol{\alpha} \in \mathbb{R}^m \mid \psi_{\mathcal{P}}(\boldsymbol{\alpha})\}\rangle$$

$$\boxed{x_j = c_j + \sum_{i=1}^{m} v_{ji}\alpha'_i}$$



ReLU

$$\varphi_1(\boldsymbol{\alpha}') := \left(0 \leq \alpha'_{m+1}\right)$$

$$\varphi_2(\boldsymbol{\alpha}') := \left(x_j \leq \alpha'_{m+1}\right)$$

$$\varphi_3(\boldsymbol{\alpha}') := \left(\alpha'_{m+1} \leq \frac{u(x_j - l)}{u - l}\right)$$

$$\Theta' = \left\langle \boldsymbol{Mc} + \boldsymbol{b}, \begin{pmatrix}\boldsymbol{MV} & \boldsymbol{e}_j\end{pmatrix}, \left\{\boldsymbol{\alpha}' \in \mathbb{R}^{m+1} \,\middle|\, \psi_{\mathcal{P}}(\boldsymbol{\alpha}') \wedge \varphi_1 \wedge \varphi_2 \wedge \varphi_3\right\}\right\rangle$$

Refinement: Local Removal of Counterexamples

Proposition: Refinement

Let Θ_o be the star resulting from an over-approximate activation function application to the star Θ. If $\boldsymbol{z} \in \Theta_o$ is the source of a counterexample and the application is an origin of $\boldsymbol{z}$, then $\boldsymbol{z} \notin \bigcup_{i=1}^{k} \Theta_i$ where $\Theta_1, \dots, \Theta_k$ are the result of the corresponding exact activation function application.

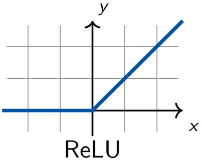
Proof

Let $\mathrm{act}(\cdot)$ be the activation function.
Assume $\boldsymbol{z} \in \Theta_i$ for an $i \in \{1, \dots, k\}$.
Then a $\boldsymbol{z}' \in \Theta$ exists such that $\mathrm{act}(\boldsymbol{z}') = \boldsymbol{z}$. Therefore, a new source of $\boldsymbol{z} \in \Theta_o$ exists in Θ. Thus, the over-approximate activation function application is not an origin. This is a contradiction, which implies $\boldsymbol{z} \notin \bigcup_{i=1}^{k} \Theta_i$.

Tracing for ReLU - Extended

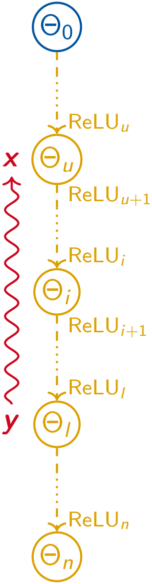
$$\boldsymbol{x} \in \Theta_u \wedge \bigwedge_{j=1}^{u} x_j = y_j \wedge \bigwedge_{i=u+1}^{l} \underbrace{\mathrm{ReLU}(x_i) = y_i}_{\varphi_i(x_i, y_i)} \wedge \bigwedge_{j=l+1}^{n} x_j = y_j$$

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{, if } x > 0 \\ 0 & \text{, if } x \leq 0 \end{cases}$$



Case $y_i > 0$: $\varphi_i(x_i, y_i) := (x_i = y_i)$

Case $y_i = 0$: $\varphi_i(x_i, y_i) := (x_i \leq 0)$



Safe History: Safe Subtrees - Proof 1

Proposition: Safe Subtrees

Let Θ_i be a star resulting from the application of the first $i \in \{0, \dots, n\}$ operations of a reachability analysis of a FNN F with n total operations. Let Θ'_i also be the result of the first i reachability analysis operations of F. If all reachable sets resulting from Θ_i are safe and $\Theta'_i \subseteq \Theta_i$, then all reachable sets resulting from Θ'_i are also safe.

Proof

Let S be the safety specification used for safety checking, and Θ_i, Θ'_i defined as before.

- If $r = 0$ operations remain, then Θ_i corresponds to a final leaf and $\Theta_i \subseteq S$. Because $\Theta'_i \subseteq \Theta_i$, Θ'_i is also safe.
- ...

Safe History: Safe Subtrees - Proof 2

Proposition: Safe Subtrees

Let Θ_i be a star resulting from the application of the first $i \in \{0, \dots, n\}$ operations of a reachability analysis of a FNN F with n total operations. Let Θ_i' also be the result of the first i reachability analysis operations of F. If all reachable sets resulting from Θ_i are safe and $\Theta_i' \subseteq \Theta_i$, then all reachable sets resulting from Θ_i' are also safe.

Proof (continued)

- Assume $r + 1$ remaining operations and an affine operation next. Let the affine mapping with $\boldsymbol{W}$ and $\boldsymbol{b}$ be applied and let $\boldsymbol{x} \in (\boldsymbol{W}\Theta_i' + \boldsymbol{b})$. Then $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b}$ for $\boldsymbol{x}' \in \Theta_i'$. Thus, $\boldsymbol{x}' \in \Theta_i$ and therefore $\boldsymbol{x} \in (\boldsymbol{W}\Theta_i + \boldsymbol{b})$. This implies $(\boldsymbol{W}\Theta_i' + \boldsymbol{b}) \subseteq (\boldsymbol{W}\Theta_i + \boldsymbol{b})$.
- ...

Safe History: Safe Subtrees - Proof 3

Proof (continued)

- Assume $r + 1$ remaining operations and an exact activation function application next.
  Assume this application on Θ_i results in $k \in \mathbb{N}_+$ reachable sets $\Theta_1^e, \ldots, \Theta_k^e$. Let Θ_j^e result from the intersection with $\mathcal{H}_{j,1}, \ldots, \mathcal{H}_{j,k_j}$ halfspaces for $k_j \in \mathbb{N}$ and the affine mapping with $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$.
  Because $\Theta_i' \subseteq \Theta_i$, these or less halfspaces and affine mappings are used to apply the activation function to Θ_i'. It holds that $\Theta_i' \cap \bigcap_{j=1}^{k_j} \subseteq \Theta_i \cap \bigcap_{j=1}^{k_j}$. Based on the previous case, the subset property is preserved by affine mappings. Thus, the union of reachable sets resulting form Θ_i' is a subset of the union of reachable sets resulting from Θ_i.

Therefore, the proposition holds.

Safe History: Safe Paths

Proposition

For two paths $\Theta_0, \ldots, \Theta_m$ and $\Theta'_0, \ldots, \Theta'_m$ in reachability trees corresponding to the same FNN with the same root $\Theta_0 = \Theta'_0$, $\Theta'_m \subseteq \Theta_m$ if the following conditions hold for every $i \in \{1, \ldots, m\}$:

- If Θ_i is the result of an affine operation applied to Θ_i, then Θ'_i is the result of the same affine operation applied to Θ'_{i-1}.
- If Θ_i is the result of an exact activation function operation applied to Θ_{i-1}, then Θ'_i is the result of the same halfspace intersections and affine mapping.
- If Θ_i is the result of an over-approximated activation function operation applied to Θ_{i-1}, then Θ'_i is the result of a corresponding exact or over-approximate operation applied to Θ'_{i-1}.

Safe History: Safe Paths - Proof 1

Let $\Theta_0, \dots, \Theta_m$ and $\Theta'_0, \dots, \Theta'_m$ be two paths in reachability trees corresponding to the same FNN with the same root $\Theta_0 = \Theta'_0$ satisfying the conditions of the proposition. If $i = 0$, then $\Theta'_0 \subseteq \Theta_0$ because $\Theta'_0 = \Theta_0$. If otherwise $i+1$ operations are applied to generate Θ_{i+1} and Θ'_{i+1} and $\Theta'_i \subseteq \Theta_i$, split cases based on the operation applied to Θ_i and Θ'_i.

- Affine Operation with $\boldsymbol{W}$ and $\boldsymbol{b}$:
$$\Theta'_{i+1} = \{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \mid \boldsymbol{x} \in \Theta'_i\} \subseteq \{\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \mid \boldsymbol{x} \in \Theta_i\} = \Theta_{i+1}$$
- The same exact activation function operation intersecting with $\mathcal{H}_1, \dots, \mathcal{H}_k$ for $k \in \mathbb{N}$ and applying the affine mapping with $\boldsymbol{W}$ and $\boldsymbol{b}$. Then, $\Theta'_i \cap \bigcap_{j=1}^{k} \mathcal{H}_j \subseteq \Theta_i \cap \bigcap_{j=1}^{k} \mathcal{H}_j$. Because the affine mapping preserves the subset property, $\Theta'_{i+1} \subseteq \Theta_{i+1}$.
- ...

Safe History: Safe Paths - Proof 2

- ...
- An exact activation function operation is applied to Θ'_i and an over-approximated activation function operation is applied to Θ_i. Based on the previous case, applying the exact activation function used to compute Θ'_{i+1} to Θ_i preserves the subset property. The result of an over-approximated activation function application is a superset of the union of the stars resulting from exact activation function application. Therefore, $\Theta'_{i+1} \subseteq \Theta_{i+1}$.
- The same over-approximate activation function operation: Because $\Theta'_i \subseteq \Theta_i$, the bounds in the dimension of application are tighter for Θ'_i than for Θ_i. Thus, $\Theta'_{i+1} \subseteq \Theta_{i+1}$, because the tightest convex over-approximation is applied.

Thus, the proposition holds.

Safety Checking: Smaller Representation

Basic formula: $\boldsymbol{\alpha} \in \mathcal{P} \wedge (\boldsymbol{c} + \boldsymbol{V\alpha}) \notin \mathcal{S}$

Problem: Z3 uses an exact number representation that gets large

$$\alpha_i \in \mathbb{Q} \rightsquigarrow \alpha_i = \frac{a_i}{b_i} \text{ for } a_i, b_i \in \mathbb{Z}$$

Solution: Minimize $|a_1| + |b_1| + \cdots + |a_n| + |b_n|$

↳ Z3 supports minimizing sums

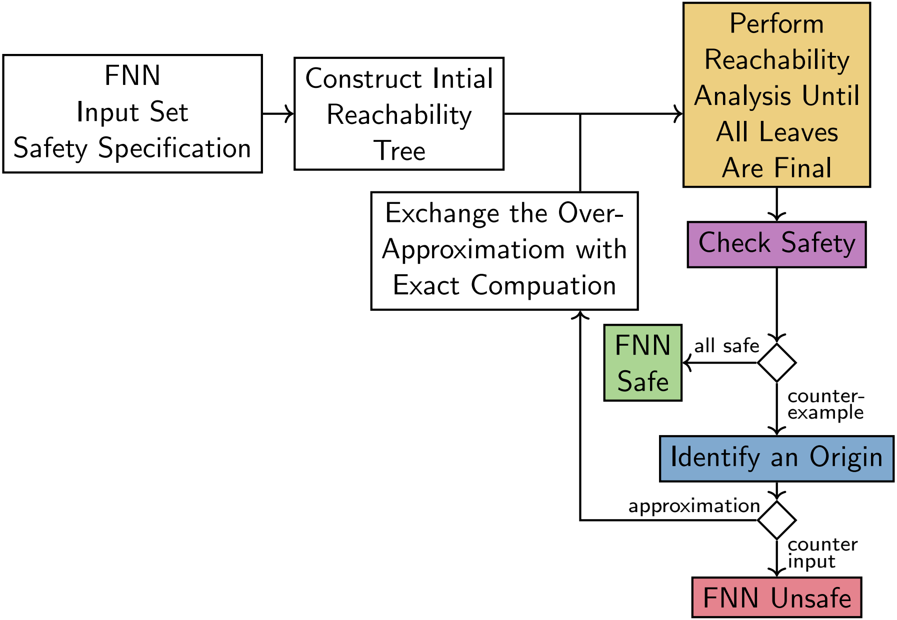
↳ But neither absolute values nor access to a_i, b_i

Approach 1: Add constraints $\alpha_i = s_i \cdot \frac{a_i}{b_i}$ for

- Real variables α_i
- Integer variables s_i constraint to $\{1, -1\}$
- Integer variables a_i constraint to $\mathbb{N}$
- Integer variables b_i constraint to $\mathbb{N}_+$

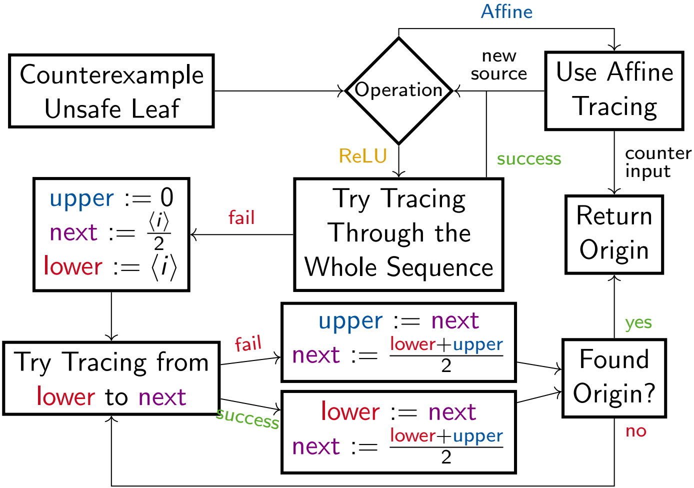
Approach 2: Replace α_i with $s_i \cdot \frac{a_i}{b_i}$

Full Refinement

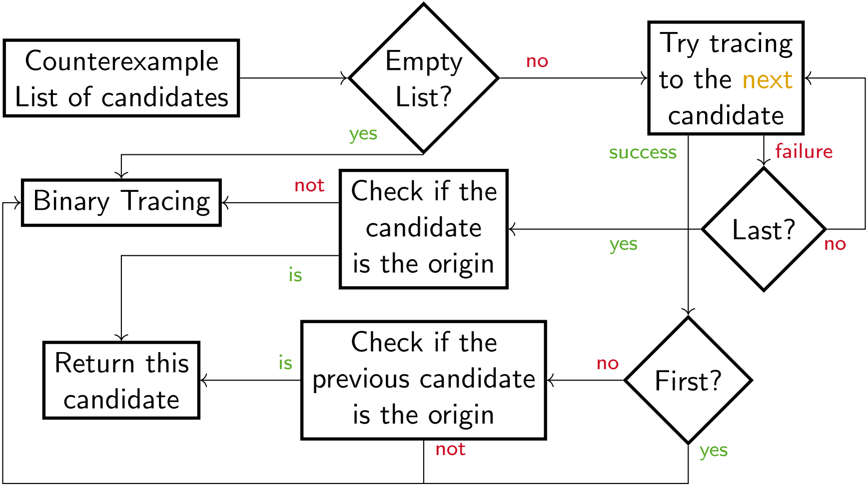


Binary Tracing

- upper: Tracing is not possible
- lower: Tracing is possible
- next: Check whether tracing is possible next



Remembering Tracing



Drones Benchmark - $\delta = 0.1$

	Time (ms)		
	Approximate	CEGAR	Exact
AC1	2398	12036	13129
AC2	2432	7864	3568
AC3	24334	87434	159360
AC4	104998	1555611	-
AC5	1659	3592	1467
AC6	3531	7529	5275
AC7	43185	314285	139500
AC8	77222	286673	94180

Drones Benchmark - $\delta = 0.1$

	Time (ms)		Checks		Variables		Constraints	
	Exact	CEGAR	Exact	CEGAR	Exact	CEGAR	Exact	CEGAR
AC1	13129	**12036**	25	**17**	12	[15, 18]	[28, 29]	[35, 42]
AC2	**3568**	7864	**5**	15	12	[13, 15]	[25, 27]	[29, 33]
AC3	159360	**87434**	81	**11**	12	[19, 20]	[27, 32]	[46, 48]
AC4	-	**1555611**	-	**19**	-	[21, 23]	-	[53, 57]
AC5	**1467**	3592	**3**	11	12	[14, 15]	[25, 27]	[31, 33]
AC6	**5275**	7529	**1**	12	12	[13, 14]	[26, 26]	[28, 30]
AC7	**139500**	314285	88	**45**	12	[13, 20]	[29, 32]	[31, 48]
AC8	**94180**	286673	**13**	37	12	[18, 22]	[28, 30]	[45, 54]

Drones Benchmark - $\delta = 0.01$

	Time (ms)		Checks		Variables		Constraints	
	Exact	CEGAR	Exact	CEGAR	Exact	CEGAR	Exact	CEGAR
AC1	**2276649**	-	**2660**	-	12	$[31, 36]$	$[28, 41]$	$[83, 96]$
AC2	-	-	-	-	-	-	-	-
AC3	-	-	-	-	-	-	-	-
AC4	-	-	-	-	-	-	-	-
AC5	-	-	-	-	-	-	-	-
AC6	-	-	-	-	-	-	-	-
AC7	-	-	-	-	-	-	-	-
AC8	-	-	-	-	-	-	-	-