OVERTPOLY

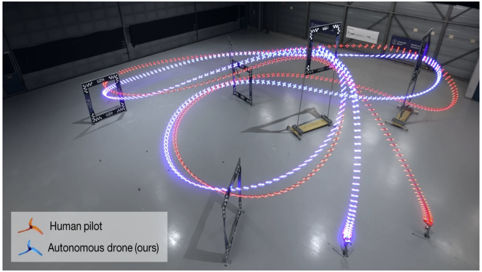
An Algorithm for Forward Reachability Analysis of Neural Feedback Systems

Samuel (Ifeoluwa) Akinwande
Stanford University

samakin@stanford.edu
03/17/2025

Introduction

Neural networks have found recent success as controllers for dynamical systems

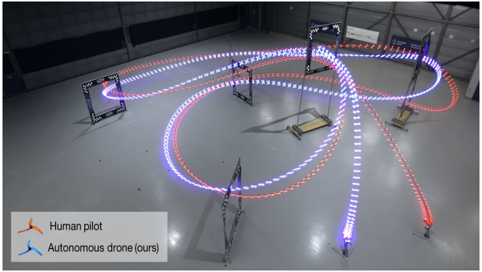


(a) Drone Racing [1]

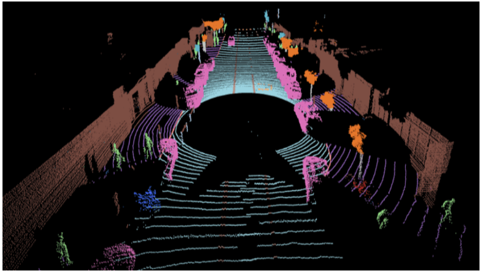
[1] Credit: (Kaufmann et al. 2023)
[2] Credit: (Ettinger et al. 2021)

Introduction

Neural networks have found recent success as controllers for dynamical systems



(a) Drone Racing [1]



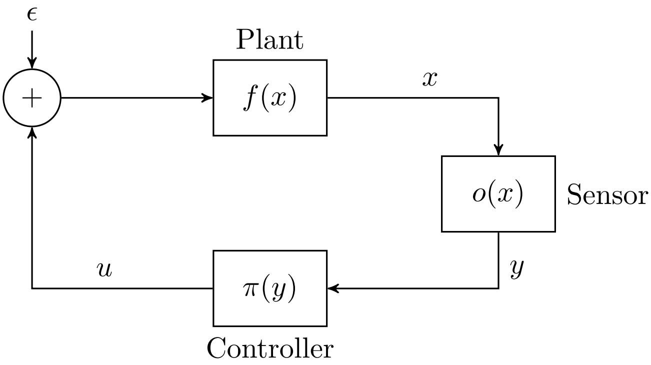
(b) Autonomous Driving [2]

[1] Credit: (Kaufmann et al. 2023)

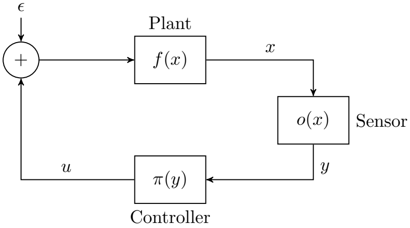
[2] Credit: (Ettinger et al. 2021)

Neural Feedback Systems

The resulting systems are what we call *Neural Feedback Systems* (NFS)

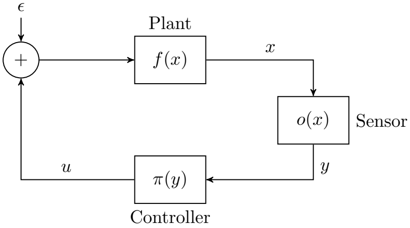


Neural Feedback Systems



Assume $x \in \mathbb{R}^n$, $f(x) = [f_1(x), \ldots, f_n(x)]$, where each $f_i : \mathbb{R}^n \to \mathbb{R}$, $\epsilon \in E$, π is a neural network

Neural Feedback Systems



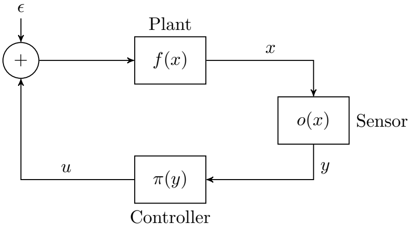
Assume $x \in \mathbb{R}^n, f(x) = [f_1(x), \ldots, f_n(x)]$, where each $f_i : \mathbb{R}^n \to \mathbb{R}$, $\epsilon \in E$, π is a neural network

The system ($\mathcal{D}$) evolves such that for each $i \in [1..n]$

$$next^{\mathcal{D}}(x)_i = \left\{ x_i + \left(f_i(x) + \pi(o(x))_i + \epsilon \right) \cdot \delta | \epsilon \in E \right\}$$

where δ is the time step size, and E the error set

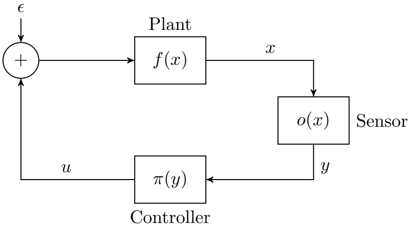
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system

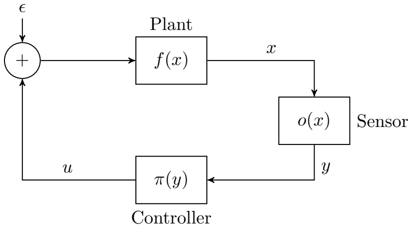
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states

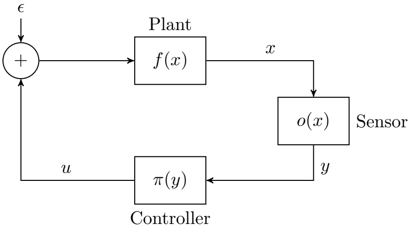
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i

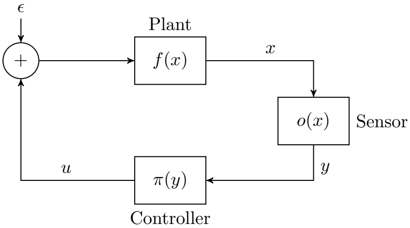
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set

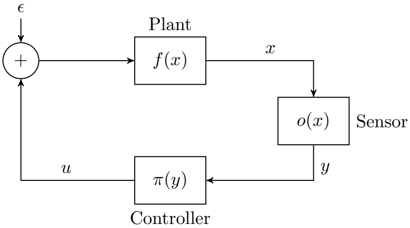
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set
- u is the neural network controller

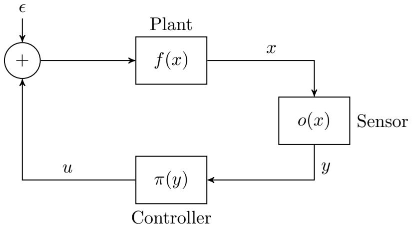
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set
- u is the neural network controller
- δ is the time step size

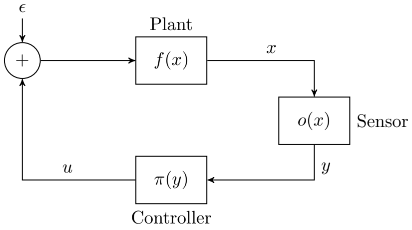
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set
- u is the neural network controller
- δ is the time step size
- T is the time horizon

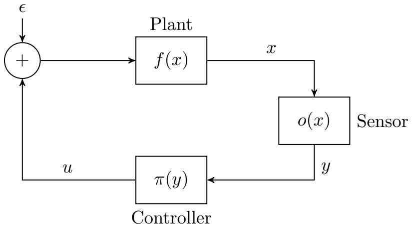
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set
- u is the neural network controller
- δ is the time step size
- T is the time horizon
- G is a set of goal states

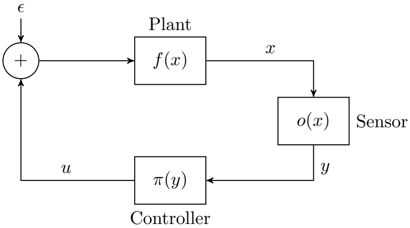
Neural Feedback Systems



A NFS is the tuple $\langle n, I, F, E, u, \delta, T, G, A \rangle$, where

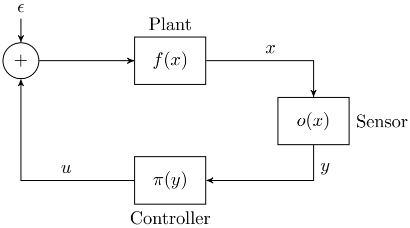
- n is the dimensionality of the system
- I is a set of initial states
- F is the set of transition functions f_i
- E is a bounded error set
- u is the neural network controller
- δ is the time step size
- T is the time horizon
- G is a set of goal states
- A is the set of unsafe states at each time step

Reach-Avoid Properties



A *trajectory* $\tau^{\mathcal{D}}(\mathcal{X}_0)$ is a sequence of state sets $(\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_T)$, where $\mathcal{X}_0 \subseteq I$, and for each $t \in [1..T]$, $\mathcal{X}_t = next^{\mathcal{D}}(\mathcal{X}_{t-1})$

Reach-Avoid Properties



A *trajectory* $\tau^{\mathcal{D}}(\mathcal{X}_0)$ is a sequence of state sets $(\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_T)$, where $\mathcal{X}_0 \subseteq I$, and for each $t \in [1..T]$, $\mathcal{X}_t = next^{\mathcal{D}}(\mathcal{X}_{t-1})$

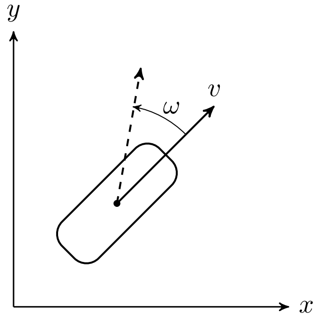
A system ($\mathcal{D}$) is *safe* if for all trajectories $\tau^{\mathcal{D}}(\mathcal{X}_0)$,

$$\forall x_0 \in I \,.\exists t \in [0..T] \,.\, \tau^{\mathcal{D}}(\{x_0\})_t \subseteq G, \tag{1}$$

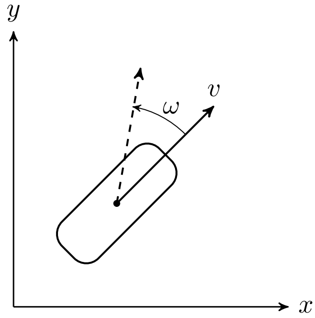
$$\forall t \in [0..T], \tau^{\mathcal{D}}(I)_t \cap A(t) = \emptyset \tag{2}$$

Unicycle Car Model

The state variables are $\mathbf{x} = (x, y, \omega, v)$.

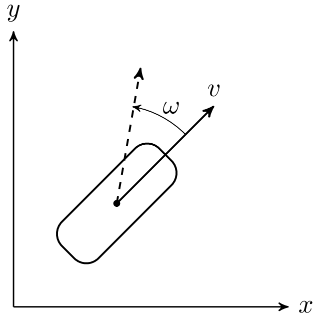


Unicycle Car Model



The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

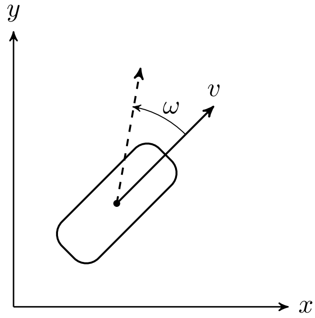
Unicycle Car Model



The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

Unicycle Car Model

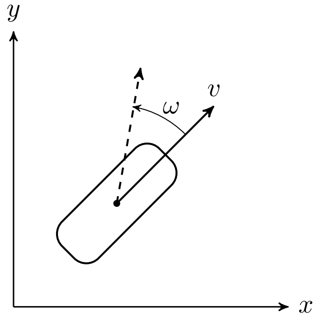


The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

$$I^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51],$$

Unicycle Car Model



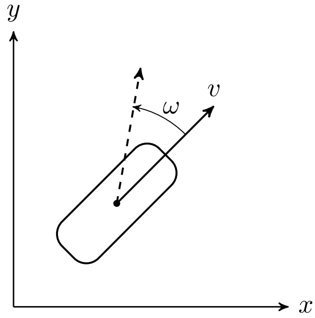
The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

$$I^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51],$$

$$F^{\mathcal{U}} = (\mathbf{x}_4 \cos(\mathbf{x}_3), \quad \mathbf{x}_4 \sin(\mathbf{x}_3), \quad 0, \quad 0)$$

Unicycle Car Model



The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

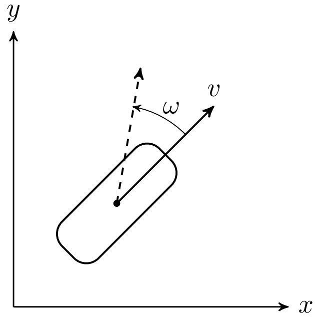
$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

$$I^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51],$$

$$F^{\mathcal{U}} = (\mathbf{x}_4 \cos(\mathbf{x}_3), \quad \mathbf{x}_4 \sin(\mathbf{x}_3), \quad 0, \quad 0)$$

$$E^{\mathcal{U}} = \{0\} \times \{0\} \times \{0\} \times [-10^{-4}, 10^{-4}],$$

Unicycle Car Model



The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

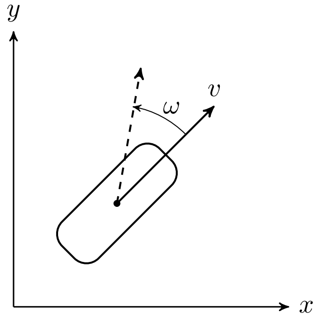
$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

$$I^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51],$$

$$F^{\mathcal{U}} = (\mathbf{x}_4 \cos(\mathbf{x}_3), \quad \mathbf{x}_4 \sin(\mathbf{x}_3), \quad 0, \quad 0)$$

$$E^{\mathcal{U}} = \{0\} \times \{0\} \times \{0\} \times [-10^{-4}, 10^{-4}],$$
$$G^{\mathcal{U}} = [-0.6, 0.6] \times [-0.2, 0.2] \times [-0.06, 0.06] \times [-0.3, 0.3],$$

Unicycle Car Model



The state variables are $\mathbf{x} = (x, y, \omega, v)$.We define the neural feedback system $\mathcal{U}$ as follows:

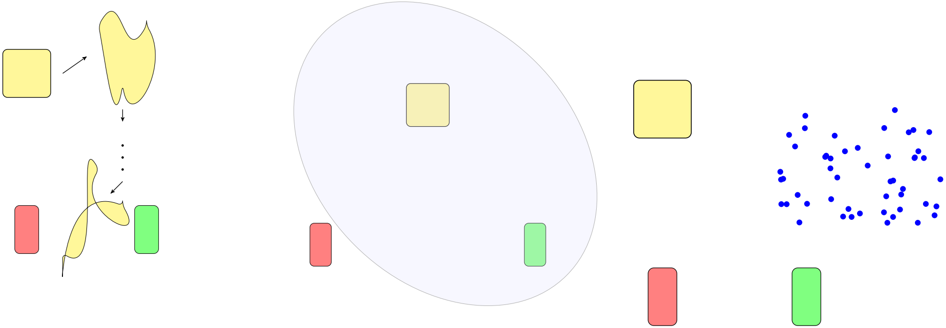
$$\langle 4, I^{\mathcal{U}}, F^{\mathcal{U}}, E^{\mathcal{U}}, u^{\mathcal{U}}, 0.2, 50, G^{\mathcal{U}}, \emptyset \rangle$$

$I^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51]$,

$$F^{\mathcal{U}} = (\mathbf{x}_4 \cos(\mathbf{x}_3), \quad \mathbf{x}_4 \sin(\mathbf{x}_3), \quad 0, \quad 0)$$

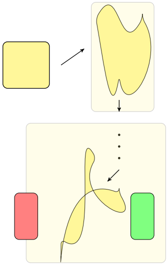
$E^{\mathcal{U}} = \{0\} \times \{0\} \times \{0\} \times [-10^{-4}, 10^{-4}]$,
$G^{\mathcal{U}} = [-0.6, 0.6] \times [-0.2, 0.2] \times [-0.06, 0.06] \times [-0.3, 0.3]$,
$u^{\mathcal{U}}$ is a ReLU neural network with one hidden layer, 500 neurons, and four outputs.

Verification Approaches

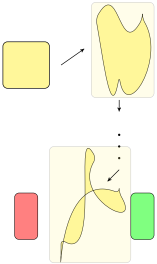


(a) Reachability Analysis (b) Invariance Analysis (c) Falsification

Reachability Approaches

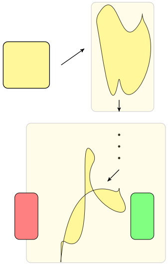


(a) Abstraction Propagation



(b) Combinatorial Optimization

Abstraction Propagation



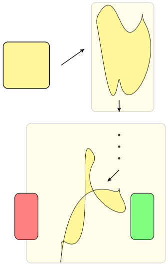
Tools using these methods propagate an abstract representation of the system to compute reachable sets.

Representations include: Taylor models, Bernstein polynomials, zonotopes, and polytopes.

The CORA tool[a] is a representative example of this approach.

[a] Kochdumper and Althoff 2023.

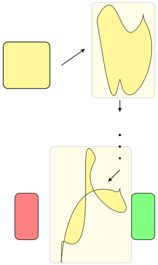
Limitations: Abstraction Propagation



Abstraction propagation can be computationally efficient, but the inexactness of the abstraction often leads to excess conservatism

This especially affects systems with nonlinear dynamics, and long time horizons.

Combinatorial Optimization



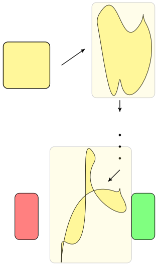
Tools using these methods solve combinatorial problems to compute reachable sets.

They often represent the system as integer programs, hybrid zonotopes, or marching trees.

The OVERTVerify tool[a] is a representative example of this approach.

[a] Sidrane et al. 2022.

Limitations: Combinatorial Optimization



While combinatorial optimization can be arbitrarily precise, computing reachable sets is computationally expensive.

The problem quickly becomes intractable when the system dynamics are nonlinear, or the time horizon is long.

OvertPoly

OvertPoly is a **combinatorial algorithm** for forward reachability analysis of Neural Feedback Systems with **computational efficiency** comparable to abstraction propagation methods.

OvertPoly

Our contributions are:

- We introduce a **novel combinatorial abstraction** for nonlinear dynamical systems

OvertPoly

Our contributions are:

- We introduce a **novel combinatorial abstraction** for nonlinear dynamical systems
- Using our abstraction, we define an **efficient representation** of nonlinear neural feedback systems

OvertPoly

Our contributions are:

- We introduce a **novel combinatorial abstraction** for nonlinear dynamical systems
- Using our abstraction, we define an **efficient representation** of nonlinear neural feedback systems
- We use this representation to define novel algorithms for forward reachability analysis

OvertPoly

Our contributions are:

- We introduce a **novel combinatorial abstraction** for nonlinear dynamical systems
- Using our abstraction, we define an **efficient representation** of nonlinear neural feedback systems
- We use this representation to define novel algorithms for forward reachability analysis
- We demonstrate an **order of magnitude improvement** in performance compared to the current state-of-the-art

OvertPoly

Assumptions:

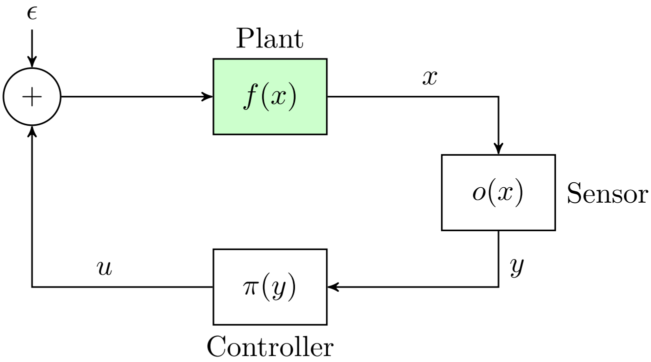
- The nonlinear dynamics are from the class of Extended Algebraic functions

OVERTPOLY

Assumptions:

- The nonlinear dynamics are from the class of Extended Algebraic functions
- The controller is a ReLU neural network

OvertPoly



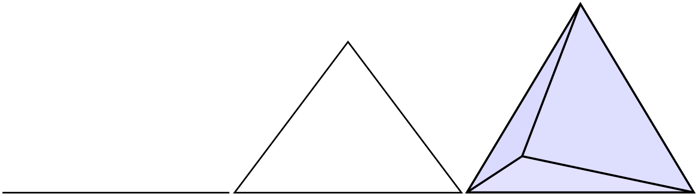
Polyhedra: Simplices

Definition (k-Simplex)

A polyhedron $\mathcal{S}$ is a k-simplex if it is the convex hull of $k+1$ affinely independent points in $\mathbb{R}^n$. A polyhedron is a simplex if it is a k-simplex for some k, and k is called its *dimension*.

Polyhedra: Simplices

Definition
A polyhedron $\mathcal{S}$ is a k-simplex if it is the convex hull of $k+1$ affinely independent points in $\mathbb{R}^n$.



(a) 1-Simplex (b) 2-Simplex (c) 3-Simplex

Polyhedra: Complexes

Definition (Simplicial k-Complex)

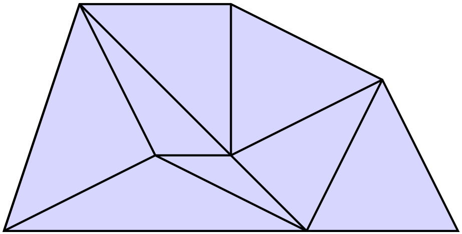
A *simplicial complex* $\mathcal{C}$ is a set of simplices such that:

- Every face of a simplex in $\mathcal{C}$ is also in $\mathcal{C}$
- Every non-empty intersection of two simplices $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{C}$ is a face of both $\mathcal{S}_1$ and $\mathcal{S}_2$

Polyhedra: Complexes

Definition

- Every face of a simplex in $\mathcal{C}$ is also in $\mathcal{C}$
- Every non-empty intersection of two simplices $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{C}$ is a face of both $\mathcal{S}_1$ and $\mathcal{S}_2$



Polyhedra: Triangulations

Definition (Point Set Triangulation)

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of* P if $P = \bigcup_{\mathcal{S} \in \mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S} \in \mathcal{S}} \mathcal{S}$.

Polyhedra: Triangulations

Definition (Point Set Triangulation)

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of P* if $P = \bigcup_{\mathcal{S}\in\mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S}\in\mathcal{S}} \mathcal{S}$.

Let C be a closed n-ball. We call C^O its open n-ball, and C^S the hypersphere forming its surface. We call $V_P(C) = C \cap P$ the vertices of C.

Polyhedra: Triangulations

Definition (Point Set Triangulation)

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of* P if $P = \bigcup_{\mathcal{S} \in \mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S} \in \mathcal{S}} \mathcal{S}$.

Let C be a closed n-ball. We call C^O its open n-ball, and C^S the hypersphere forming its surface. We call $V_P(C) = C \cap P$ the vertices of C.

If $\mathcal{C}$ is a point set triangulation of P, and $\mathcal{S}$ is a n-simplex in $\mathcal{C}$, then $C(\mathcal{S})$ is the smallest closed n-ball C such that $\mathcal{S} \subseteq C$ and $C^O \cap \mathbf{vert}(\mathcal{S}) = \emptyset$.

Polyhedra: Triangulations

Definition (Point Set Triangulation)

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of* P if $P = \bigcup_{\mathcal{S}\in\mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S}\in\mathcal{S}} \mathcal{S}$.

Let C be a closed n-ball. We call C^O its open n-ball, and C^S the hypersphere forming its surface. We call $V_P(C) = C \cap P$ the vertices of C.

If $\mathcal{C}$ is a point set triangulation of P, and $\mathcal{S}$ is a n-simplex in $\mathcal{C}$, then $C(\mathcal{S})$ is the smallest closed n-ball C such that $\mathcal{S} \subseteq C$ and $C^O \cap \mathbf{vert}(\mathcal{S}) = \emptyset$.

$\mathcal{S}$ satisfies the *Delaunay condition* and is called a *Delaunay simplex of* P if $V_P(C(\mathcal{S})^O) = \emptyset$.

Polyhedra: Triangulations

Definition (Point Set Triangulation)

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of* P if $P = \bigcup_{\mathcal{S}\in\mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S}\in\mathcal{S}} \mathcal{S}$.

Let C be a closed n-ball. We call C^O its open n-ball, and C^S the hypersphere forming its surface. We call $V_P(C) = C \cap P$ the vertices of C.

If $\mathcal{C}$ is a point set triangulation of P, and $\mathcal{S}$ is a n-simplex in $\mathcal{C}$, then $C(\mathcal{S})$ is the smallest closed n-ball C such that $\mathcal{S} \subseteq C$ and $C^O \cap \mathbf{vert}(\mathcal{S}) = \emptyset$.

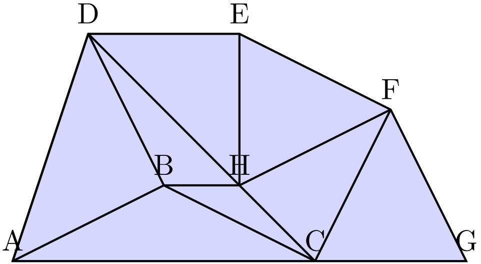
$\mathcal{S}$ satisfies the *Delaunay condition* and is called a *Delaunay simplex of* P if $V_P(C(\mathcal{S})^O) = \emptyset$.

$\mathcal{C}$ is a *Delaunay triangulation* if every n-simplex in $\mathcal{C}$ satisfies the Delaunay condition.

Polyhedra: Triangulations

Definition

If P is a finite set of points in $\mathbb{R}^n$, then a pure simplicial n-complex $\mathcal{C}$ is a *point set triangulation of* P if $P = \bigcup_{\mathcal{S}\in\mathcal{C}} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S}\in\mathcal{S}} \mathcal{S}$.

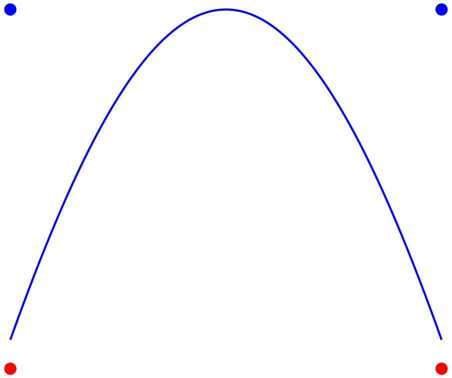


Polyhedra: Bounding Set

Definition (Bounding Set)

A *bounding set* is a tuple $\mathcal{B} = \langle n, P, L, U \rangle$, where $n \in \mathbb{N}$, P is finite a set of points in $\mathbb{R}^n$, and L and U are functions from P to $\mathbb{R}$, such that $L(p) \leq U(p)$ for all $p \in P$. The *domain* of $\mathcal{B}$ is defined as $\mathbf{dom}(\mathcal{B}) = \mathbf{conv}(P)$.

Polyhedra: Bounding Set



Definition

A *bounding set* is a tuple $\mathcal{B} = \langle n, P, L, U \rangle$. The *domain* of $\mathcal{B}$ is defined as $\mathbf{dom}(\mathcal{B}) = \mathbf{conv}(P)$.

Definition (Polyhedron formed by Bounding Set)

Let $\mathcal{B} = \langle n, P, L, U \rangle$ be a bounding set. We define the *vertices* of the bounding set as:

$$V(\mathcal{B}) := \{(p, L(p)) : p \in P\} \cup \{(p, U(p)) : p \in P\}.$$

We define the ($n+1$-dimensional) *polyhedron formed by $\mathcal{B}$* as

$$\mathcal{P}(\mathcal{B}) := \mathbf{conv}(V(\mathcal{B})).$$

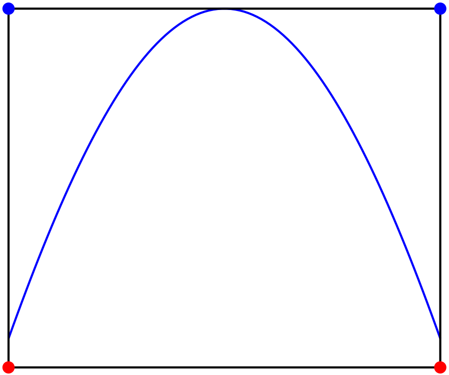
Polyhedra

Definition

$V(\mathcal{B}) := \{(p, L(p)) : p \in P\} \cup \{(p, U(p)) : p \in P\}.$

We define the ($n+1$-dimensional) *polyhedron formed by* $\mathcal{B}$ as

$$\mathcal{P}(\mathcal{B}) := \mathbf{conv}(V(\mathcal{B})).$$



Polyhedral Enclosures

Definition (Polyhedral Enclosure)

Let $\mathcal{B} = \langle n, P, L, U \rangle$ be a bounding set, let Δ be a Delaunay triangulation of P, and let Δ_n be the set of all n-simplices in Δ. We then define the bounding set associated with a simplex $\mathcal{S} \in \Delta_n$ as:

$$\mathcal{B}_{\mathcal{S}} := \langle n, \mathbf{vert}(\mathcal{S}), L^{\mathbf{vert}(\mathcal{S})}, U^{\mathbf{vert}(\mathcal{S})} \rangle,$$

We define the *polyhedral enclosure* formed by $\mathcal{B}$ and Δ as:

$$\mathcal{E}(\mathcal{B}, \Delta) := \bigcup_{S \in \Delta_n} \mathcal{P}(\mathcal{B}_S).$$

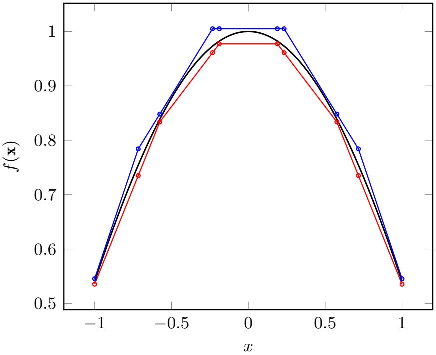
Polyhedral Enclosures

Definition

$$\mathcal{B}_{\mathcal{S}} := \langle n, \mathbf{vert}(\mathcal{S}), L^{\mathbf{vert}(\mathcal{S})}, U^{\mathbf{vert}(\mathcal{S})} \rangle,$$

We define the *polyhedral enclosure* formed by $\mathcal{B}$ and Δ as:

$$\mathcal{E}(\mathcal{B}, \Delta) := \bigcup_{S \in \Delta_n} \mathcal{P}(\mathcal{B}_S).$$



Polyhedral Enclosures: Composition

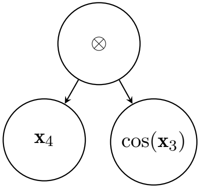
Definition (Bounding Set Composition)

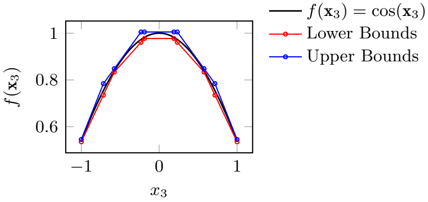
Let $\mathcal{B}^f = \langle n, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n, P, L^g, U^g \rangle$ be bounding sets with the same dimension and over the same set of points P. For $\bowtie \in \{+, -, \times, /\}$, we define $\mathcal{B}^f \bowtie \mathcal{B}^g := \langle n, P, L, U \rangle$, where, for all $\mathbf{x} \in P$,

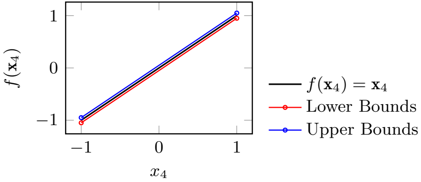
$$L(\mathbf{x}), U(\mathbf{x}) = \begin{cases} L^f(\mathbf{x}) + L^g(\mathbf{x}), U^f(\mathbf{x}) + U^g(\mathbf{x}) & \text{if } \bowtie = +, \\ L^f(\mathbf{x}) - L^g(\mathbf{x}), U^f(\mathbf{x}) - U^g(\mathbf{x}) & \text{if } \bowtie = -, \\ \min(Bounds), \max(Bounds) & \text{if } \bowtie \in \{\times, \div\}, \end{cases}$$

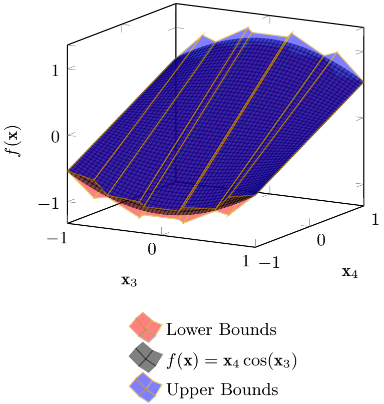
where $Bounds = \{h_1(x) \bowtie h_2(x) \mid h_1 \in \{L^f, U^f\}, h_2 \in \{L^g, U^g\}\}$ is the set of potential bounds when using multiplication or division.

Polyhedral Enclosures: Composition









Polyhedral Enclosures

Polyhedral enclosures are **an efficient combinatorial abstraction** for nonlinear dynamical systems.

Convex Combination Encoding

The **structure** of polyhedral enclosures permits the use of efficient encodings to represent dynamical systems as mixed integer linear programs (MILPs).

We adapt the convex combination encoding[1] to represent each nonlinear transition function as an MILP.

[1] GeiSSler et al. 2011.

Convex Combination Encoding

Each transition function is represented as an MILP of the form

$$\sum_{j=1}^{d} \lambda_j = 1, \quad \lambda \succcurlyeq 0, \quad \sum_{i=1}^{m} b_i \leq 1, \quad b \in \{0,1\}^m \tag{3a}$$

$$\lambda_j \leq \sum_{\{i \mid \mathbf{p}_j \in \mathbf{vert}(S_i)\}} b_i \quad \text{for } \; j = 1, \ldots, d \tag{3b}$$

$$\mathbf{x} = \sum_{j=1}^{d} \lambda_j \cdot \mathbf{p}_j \tag{3c}$$

$$\overline{y} = \sum_{j=1}^{d} \lambda_j \cdot \overline{f}(\mathbf{p}_j), \quad \underline{y} = \sum_{j=1}^{d} \lambda_j \cdot \underline{f}(\mathbf{p}_j) \tag{3d}$$

$$\underline{y} \leq y \leq \overline{y} \tag{3e}$$

Convex Combination Encoding

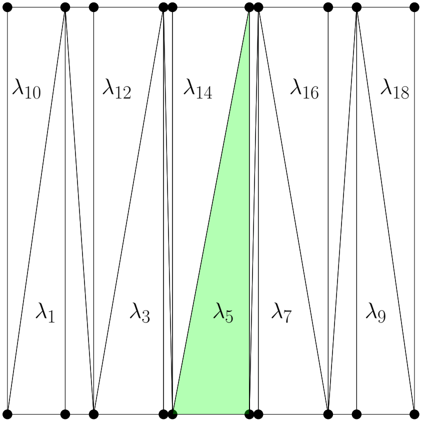
$$\sum_{j=1}^{d} \lambda_j = 1, \quad \lambda \succcurlyeq 0, \quad \sum_{i=1}^{m} b_i \leq 1, \quad b \in \{0,1\}^m \tag{4a}$$

$$\lambda_j \leq \sum_{\{i \mid \mathbf{p}_j \in \mathbf{vert}(S_i)\}} b_i \quad \text{for} \quad j = 1, \ldots, d \tag{4b}$$

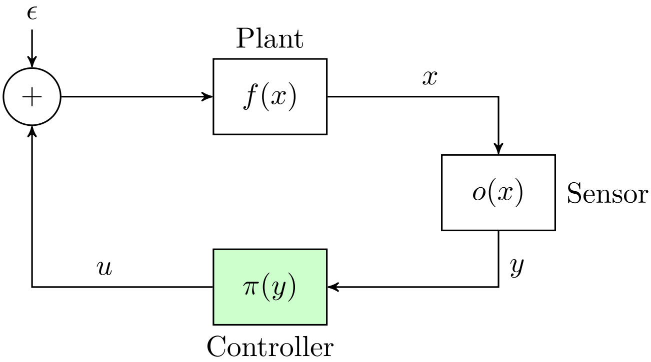
$$\mathbf{x} = \sum_{j=1}^{d} \lambda_j \cdot \mathbf{p}_j \tag{4c}$$

$$\overline{y} = \sum_{j=1}^{d} \lambda_j \cdot \overline{f}(\mathbf{p}_j), \quad \underline{y} = \sum_{j=1}^{d} \lambda_j \cdot \underline{f}(\mathbf{p}_j) \tag{4d}$$

$$\underline{y} \leq y \leq \overline{y} \tag{4e}$$



OvertPoly



ReLU Networks as MILPs

A neural network is a composition of affine transformations and nonlinear activation functions.

[2] Tjeng, Xiao, and Tedrake 2017.

ReLU Networks as MILPs

A neural network is a composition of affine transformations and nonlinear activation functions.
When the activation function is the rectified linear unit (ReLU), it results in a piecewise linear function.

[2] Tjeng, Xiao, and Tedrake 2017.

ReLU Networks as MILPs

A neural network is a composition of affine transformations and nonlinear activation functions.
When the activation function is the rectified linear unit (ReLU), it results in a piecewise linear function.
We adapt the MIPVerify[2] encoding to represent ReLU networks as MILPs. A single ReLU yields

[2] Tjeng, Xiao, and Tedrake 2017.

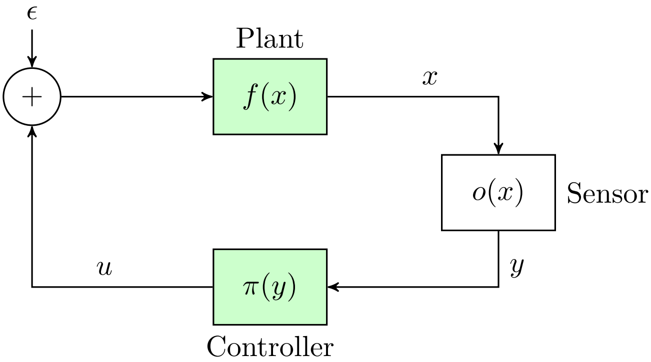
ReLU Networks as MILPs

A neural network is a composition of affine transformations and nonlinear activation functions.
When the activation function is the rectified linear unit (ReLU), it results in a piecewise linear function.
We adapt the MIPVerify[2] encoding to represent ReLU networks as MILPs. A single ReLU yields

$$
\begin{aligned}
l, u \in \mathbb{R}, \quad l \leq x \leq u, \quad l \leq 0 \leq u, \quad \mathbb{1}_{x \geq 0} \\
y \leq x - l(1 - \mathbb{1}_{x \geq 0}), \quad y \geq x, \quad y \leq u \cdot \in x \geq 0, \quad y \geq 0.
\end{aligned}
$$

[2] Tjeng, Xiao, and Tedrake 2017.

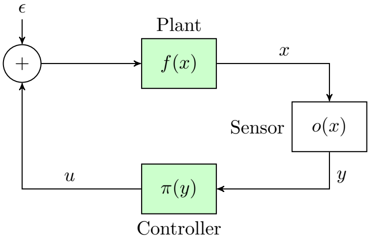
OvertPoly



Forward Reachability Analysis

Since both the plant and controller are represented as MILPs, we can compute tight bounds on the forward reachable set of the system by solving a series of MILPs.

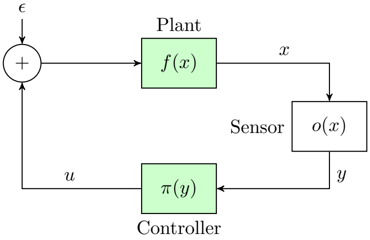
Forward Reachability Analysis



Recall that the system evolves such that for each $i \in [1..n]$

$$next^{\mathcal{D}}(x)_i = \left\{ x_i + \big(f_i(x) + \pi(o(x))_i + \epsilon\big) \cdot \delta | \epsilon \in E \right\}$$

Forward Reachability Analysis

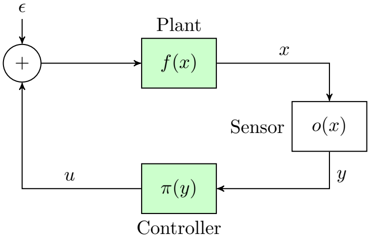


We can compute tight bounds for each $i \in [1..n]$ by solving

$$\min_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\underline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x})_i) + \epsilon) \cdot \delta \tag{5a}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{5b}$$

Forward Reachability Analysis



We can compute tight bounds for each $i \in [1..n]$ by solving

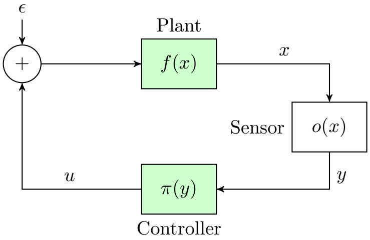
$$\min_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\underline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x})_i) + \epsilon) \cdot \delta \tag{5a}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{5b}$$

$$\max_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\overline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x}))_i + \epsilon) \cdot \delta \tag{5c}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{5d}$$

Forward Reachability Analysis: Symbolic



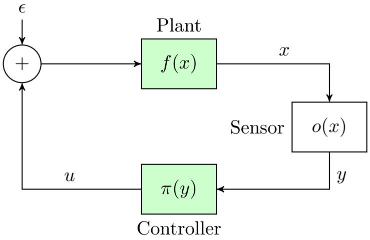
We can compute tight bounds for each $i \in [1..n]$ by solving

$$\min_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\underline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x})_i) + \epsilon) \cdot \delta \tag{6a}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{6b}$$

$$\hat{\mathcal{X}}_t = next^{\mathcal{D}}(\hat{\mathcal{X}}_{t-1}) \tag{6c}$$

Forward Reachability Analysis: Symbolic



We can compute tight bounds for each $i \in [1..n]$ by solving

$$\min_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\underline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x})_i) + \epsilon) \cdot \delta \tag{6a}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{6b}$$

$$\hat{\mathcal{X}}_t = next^{\mathcal{D}}(\hat{\mathcal{X}}_{t-1}) \tag{6c}$$

$$\max_{\mathbf{x} \in \hat{\mathcal{X}}_t} \mathbf{x}_i + (\overline{f_i}(\mathbf{x}) + \pi(o(\mathbf{x}))_i + \epsilon) \cdot \delta \tag{6d}$$

$$\pi(o(\mathbf{x})) = \mathcal{M}_0(\mathbf{x}), \quad \epsilon \in E \tag{6e}$$

$$\hat{\mathcal{X}}_t = next^{\mathcal{D}}(\hat{\mathcal{X}}_{t-1}) \tag{6f}$$

Experiments: Performance

	OvertPoly		OVERTVerify		CORA	
	Time (s)	Volume	Time (s)	Volume	Time (s)	Volume
Single Pendulum	**1.6431**	**5.533E−2**	1.9232	**5.53E−2**	2.919	7.6388
ACC	**23.1960**	**3.819E−3**	345.0036	1.119E−2	25.4478	6.1596E+5
TORA	**1461.5640**	**6.434E−1**	13620.0628	**6.434E−1**	×	6.0325E+2
Unicycle	**7217.153**	**8.979E−6**	40109.9396	9.865E−6	×	×

Table: Benchmark computation time (s) and set volumes. Computation times listed for verified instances, and × for unverified instances. All available set volumes listed. Best performance is highlighted in bold.

Experiments: Scalability

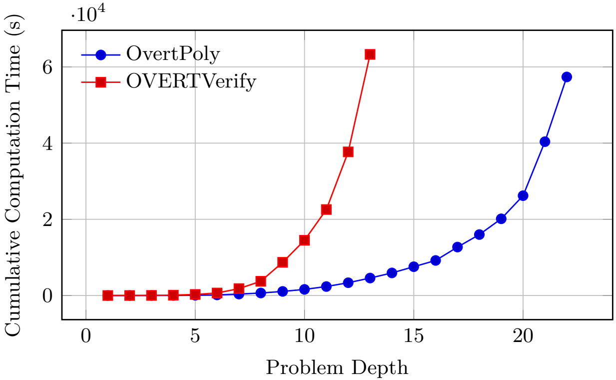


Figure: Pure symbolic reachability comparison between OverPoly and OVERTVerify using the Unicycle benchmark. Each tool was used used to compute as many pure symbolic reachable steps as possible with a (per optimizer call) timeout of 3600 seconds.

Summary

We introduced OvertPoly, a combinatorial algorithm for forward reachability analysis of neural feedback systems.

Summary

We introduced OvertPoly, a combinatorial algorithm for forward reachability analysis of neural feedback systems.

We demonstrated an **order of magnitude improvement** in performance compared to the current state-of-the-art.

Thank you!

References

Ettinger, Scott et al. (2021). “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9710–9719.

GeiSSler, Björn et al. (2011). “Using piecewise linear functions for solving MINLPs”. In: *Mixed integer nonlinear programming*. Springer, pp. 287–314.

Kaufmann, Elia et al. (2023). “Champion-level drone racing using deep reinforcement learning”. In: *Nature* 620.7976, pp. 982–987.

Kochdumper, Niklas and Matthias Althoff (2023). “Constrained polynomial zonotopes”. In: *Acta Informatica* 60.3, pp. 279–316.

Sidrane, Chelsea et al. (2022). “Overt: An algorithm for safety verification of neural network control policies for nonlinear systems”. In: *Journal of Machine Learning Research* 23.117, pp. 1–45.

Tjeng, Vincent, Kai Xiao, and Russ Tedrake (2017). “Evaluating robustness of neural networks with mixed integer programming”. In: *arXiv preprint arXiv:1711.07356*.