Verifying Nonlinear Neural Feedback Systems using Polyhedral Enclosures

I. Samuel Akinwande SAMAKIN@STANFORD.EDU
Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, USA

Chelsea Sidrane CHELSE@KTH.SE
Division of Robotics, Perception and Learning, Intelligent Systems Department, School of Electrical Engineering & Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden

Mykel J. Kochenderfer MYKEL@STANFORD.EDU
Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, USA

Clark Barrett BARRETTC@STANFORD.EDU
Department of Computer Science, Stanford University, Stanford, CA, USA

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

As dynamical systems controlled by neural networks become increasingly prevalent, it is critical to ensure their safe operation. Although efficient techniques exist to handle neural systems with *linear* transition functions, few scalable methods address the *nonlinear* case. We propose a novel algorithm for verifying nonlinear neural feedback systems using forward reachability analysis. Our algorithm leverages the structure of the nonlinear transition functions to compute tight linear abstractions which we call polyhedral enclosures. These are then encoded as mixed-integer linear programs (MILPs) and solved to yield a sound over-approximation of the forward-reachable set. We evaluate our algorithm on representative benchmarks and demonstrate significant improvements over the previous state of the art.

Keywords: Formal Verification, Neural Networks, Control Systems, Feedback Systems

1. Introduction

The success of neural networks in the fields of drone racing (Kaufmann et al., 2023), autonomous driving (Ettinger et al., 2021), and system identification (Duong et al., 2024) exemplifies the growing interest in using neural networks as controllers for dynamical systems. We refer to these as *neural feedback systems*, and many of their potential uses are in safety-critical settings. In order to realize this potential, it is crucial to develop verification techniques to ensure their safety. Although there is a rich body of work on verification for classical control systems (Tomlin et al., 2003; Chen et al., 2013; Bansal et al., 2017), these techniques are often ill-suited for neural feedback systems due to nonlinearities and the common practice of viewing neural networks as black boxes. To address this gap, verification approaches based on reachability analysis have been developed (Ivanov et al., 2020; Everett et al., 2021; Sidrane et al., 2022; Wang et al., 2023; Zhang and Xu, 2023; Kochdumper and Althoff, 2023). These methods can be broadly classified into two categories: propagation-based methods and combinatorial methods (Everett, 2021), each offering distinct advantages and disadvantages.

Following the terminology of Everett (2021), we define *propagation-based methods* as those that systematically propagate an initial set through both the dynamical system and the neural network.

© 2026 I.S. Akinwande, C. Sidrane, M.J. Kochenderfer & C. Barrett.

These methods often rely on abstractions, such as Taylor models (Dutta et al., 2019; Huang et al., 2022), Bernstein polynomials (Fan et al., 2020), zonotopes (Schilling et al., 2022), and polynomial zonotopes (Kochdumper et al., 2023), to achieve efficient computations. CORA is a prominent tool for verifying neural feedback systems using abstraction propagation (Althoff and Kochdumper, 2016). It computes reachable sets by combining approximations of the system dynamics with non-convex network abstractions (e.g., polynomial zonotopes).

In contrast, *combinatorial methods* verify properties by solving combinatorial problems. Existing approaches include techniques that model neural feedback systems as hybrid systems (Ivanov et al., 2019) or as hybrid zonotopes (Siefert et al., 2023; Zhang and Xu, 2023), as well as methods that encode the problem as marching trees (Vincent and Schwager, 2021) or integer linear programs (Sidrane et al., 2022). By emphasizing the combinatorial structure of the neural network controller, these methods trade off computational complexity for precision. A notable example is the OVERTVerify algorithm (Sidrane et al., 2022), which computes reachable sets for nonlinear neural feedback systems. OVERTVerify uses the OVERT algorithm and relational overapproximations to construct constraints that implicitly bound multivariate nonlinear functions. Combinatorial methods, including OVERTVerify, are prohibitively expensive computationally, and thus, many successful verification tools instead rely on abstraction propagation and refinement (Lopez et al., 2023).

In this paper, we introduce the OVERTPoly algorithm, which aims to address some of the limitations of combinatorial methods. We exploit a structured representation of the neural feedback system to create a combinatorial method with computational performance comparable to propagation-based methods. Our contributions include: introducing *polyhedral enclosures*, a novel combinatorial abstraction for multivariate nonlinear functions; providing an efficient method for encoding polyhedral enclosures as mixed integer linear programs (MILPs); defining a novel forward reachability algorithm with several optimizations; evaluating our algorithm and showing that it performs better than both OVERTVerify and CORA on a set of neural feedback system benchmarks. We discuss ablations and further optimizations in an extended version of this work Akinwande et al. (2025).

2. Background

2.1. Notation

We denote the set of non-negative real numbers as $\mathbb{R}_+$. If $\mathcal{X}$ is a set, we denote the *power set* of $\mathcal{X}$ (i.e., the set of all subsets of $\mathcal{X}$) as $2^{\mathcal{X}}$. We use $[i..j]$ to represent the set $\{z \in \mathbb{Z} \mid i \leq z \leq j\}$, $[i, j]$ to represent the set $\{r \in \mathbb{R} \mid i \leq r \leq j\}$, and $[n]$ to abbreviate $[1..n]$.

If S is any finite *sequence* $(s_1, \ldots, s_n)$, we write $|S|$ to denote n, the length of the sequence, and S_i to denote the i^{th} element of the sequence. We write $S_{[i..j]}$ for the sequence $(s_i, \ldots, s_j)$ and $S \circ S'$ for the sequence obtained by appending the sequence S' to the end of S. If a sequence is used where a set is expected, the meaning is the set of elements in the sequence (e.g., $s \in S$ means that s occurs in the sequence S). We use the same notation for both vectors and sequences and treat them as interchangeable.

We write $x \cdot y$ for the dot product of vectors x and y. We write $\mathbf{0}$ for the zero vector, $\mathbf{1}$ for the vector of all ones, and $\mathbf{e}_i$ for the i^{th} unit vector. The sizes of these vectors will be left implicit when it is clear from context. A *convex combination vector* $\boldsymbol{\theta}$ is a vector whose entries are non-negative and sum to 1, i.e., $\boldsymbol{\theta} \cdot \mathbf{1} = 1$ and $\boldsymbol{\theta}_i \geq 0$ for each $i \in [|\boldsymbol{\theta}|]$.

For a function $f : \mathcal{X} \to \mathbb{R}$ and a set $\mathcal{X}' \subseteq \mathcal{X}$, we define the *image* of $\mathcal{X}'$ under f as $f(\mathcal{X}') := \{f(x) \mid x \in \mathcal{X}'\}$. Similarly, if S is a sequence, then $f(S)$ is the sequence $(f(S_1), f(S_2), \dots)$. For $\mathcal{X}' \subseteq \mathcal{X}$, we define the *restriction* of f to $\mathcal{X}'$ as the function $f^{\mathcal{X}'} : \mathcal{X}' \to \mathbb{R}$ such that $f^{\mathcal{X}'}(x) = f(x)$ for every $x \in \mathcal{X}'$.

2.2. Convex Geometry

This work makes use of standard concepts from convex geometry. We summarize essential notions here and provide complete definitions in Appendix C.1. Let $P = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \dots, \boldsymbol{p}_k\}$ be a finite set of points in $\mathbb{R}^n$. The points in P are *affinely independent* iff the set $\{\boldsymbol{p}_1 - \boldsymbol{p}_0, \dots, \boldsymbol{p}_k - \boldsymbol{p}_0\}$ is linearly independent. We say that P is *full-dimensional* if it contains $n + 1$ affinely independent points. We rely on standard notions of simplices and Delaunay triangulations. We use **conv** to denote the convex hull operator, and **vert** to denote the operator that returns the set of vertices in a simplex. The *Delaunay triangulation* of a set of points P is a Delaunay triangulation Δ with $\textbf{vert}(\Delta) = P$.

Definition 1 (Grid) *A* grid of dimension n *is the Cartesian product of n finite subsets of $\mathbb{R}$, each containing at least two elements. Given a grid G, we define G_i as the projection of G onto dimension i, so that $G = G_1 \times \cdots \times G_n$. The* domain *of the grid is defined as $dom(G) = \mathbf{conv}(G)$. A* grid cell *of G is a subset of G whose convex hull is an n-dimensional hyperrectangle R, such that no points of G other than those forming the vertices of R are contained in R. For $\boldsymbol{x} \in G$, $Cells(G, \boldsymbol{x})$ denotes the set of all grid cells containing $\boldsymbol{x}$, i.e., $Cells(G, \boldsymbol{x}) = \{\mathcal{X} \mid \mathcal{X}$ is a grid cell of G and $\boldsymbol{x} \in \mathcal{X}\}$.*

3. Neural Feedback Systems

We define a discrete-time neural feedback system $\mathcal{D}$ as a tuple $\langle n, \mathbf{I}, \mathbf{F}, \mathbf{E}, \mathbf{u}, \delta, T, \mathbf{G}, \mathbf{A}\rangle$, where $n \in \mathbb{N}$ is the *dimension* of the system (i.e., every state of the system is an element of $\mathbb{R}^n$), $\mathbf{I} \subseteq \mathbb{R}^n$ is the set of *initial states*, $\mathbf{F}$ is a sequence $(f_1, \dots, f_n)$ of *state update functions* with $f_i : \mathbb{R}^n \to \mathbb{R}$, $\mathbf{E} \subseteq \mathbb{R}^n$ is a perturbation error set (i.e., a set from which an error term may be introduced when computing the next state), $\mathbf{u} : \mathbb{R}^n \to \mathbb{R}^n$ is the *control function*, $\delta \in \mathbb{R}_+$ is the *time step size*, $T \in \mathbb{N}$ is the *number of time steps*, $\mathbf{G} \subseteq \mathbb{R}^n$ is the set of *goal states*, and $\mathbf{A} : [0..T] \to 2^{\mathbb{R}^n}$ is a function from time steps to the set of *avoid states* (i.e., unsafe states) at that time step.

States evolve over a sequence of T discrete time steps, each of duration δ, and the *time horizon* is $\delta \cdot T$. If $\boldsymbol{x} \in \mathbb{R}^n$ is a state, the next-state function $next^{\mathcal{D}} : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ defines the set of possible next states (it is a set because of the nondeterminism introduced by the error term) as follows:

$$next^{\mathcal{D}}(\boldsymbol{x}) = \big\{\boldsymbol{x} + \big(\boldsymbol{F}(\boldsymbol{x}) + \mathbf{u}(\boldsymbol{x}) + \boldsymbol{\epsilon}\big) \cdot \delta \;\mid\; \epsilon \in \mathbf{E}\big\}. \tag{1}$$

We assume that each f_i is from the class of Lipschitz continuous multivariate functions composed of rational operations (i.e., $+, -, \times, \div$) on univariate elementary functions.[1] We note that the majority of NFS verification benchmarks (Lopez et al. (2023)) fall into this category. We call these functions *extended rational nonlinear functions*.[2] We also assume that the neural network controller $\mathbf{u}$ is a multilayer perceptron with n inputs, n outputs, and ReLU activations.

1. Following the convention of Muller and Muller (2006), we define elementary functions as the trigonometric functions, their inverses, the exponential functions, and logarithmic functions.
2. While we only consider a subset of nonlinear functions, the Kolmogorov-Arnold representation theorem (Kurková, 1991) suggests that our approach can be generalized to arbitrary nonlinear functions.

A trajectory $\tau^{\mathcal{D}}(\mathcal{X}_0)$, where $\mathcal{X}_0 \subseteq \mathbf{I}$, is the sequence of sets of states $(\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_T)$, where $\mathcal{X}_i = next^{\mathcal{D}}(\mathcal{X}_{i-1})$ for $i \in [1..T]$.

A system $\mathcal{D}$ is *safe* if it satisfies the following reach-avoid properties:

$$\forall\, \boldsymbol{x}_0 \in \mathbf{I}.\, \exists\, t \in [0..T].\, \tau^{\mathcal{D}}(\{\boldsymbol{x}_0\})_t \subseteq \mathbf{G}, \tag{2}$$

$$\forall\, t \in [0..T].\, \tau^{\mathcal{D}}(\mathbf{I})_t \cap \mathbf{A}(t) = \emptyset. \tag{3}$$

Property 2 is a *reach* property. It states that every trajectory starting from some state in the initial state set reaches the goal set (specified by $\mathbf{G}$) within the specified time horizon. On the other hand, Property 3 is an *avoid* property, requiring that the system avoids any unsafe states ($\mathbf{A}$) at each time within the given time horizon. An illustrative example can be found in the appendix.

4. Polyhedral Enclosures

In our verification approach, we use *polyhedral enclosures* to provide tight overapproximations of nonlinear functions. In this section, we provide a formal definition of a polyhedral enclosure and then discuss how to construct and compose them.

Definition 2 (Bounding Set) *A* bounding set *is a tuple $\mathcal{B} = \langle n, P, L, U \rangle$, where $n \in \mathbb{N}$, P is a finite, full-dimensional set of points in $\mathbb{R}^n$, and L and U are functions from P to $\mathbb{R}$, such that $L(\boldsymbol{p}) \leq U(\boldsymbol{p})$ for all $\boldsymbol{p} \in P$. The* domain *of $\mathcal{B}$ is defined as $dom(\mathcal{B}) = \mathbf{conv}(P)$.*

Definition 3 (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:*

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

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

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

Definition 4 (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 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:*

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

Definition 5 (Function Enclosure) *Let $\mathcal{B} = \langle n, P, L, U \rangle$ be a bounding set. A function $f : D \to \mathbb{R}$, where $dom(\mathcal{B}) \subseteq D \subseteq \mathbb{R}^n$, is* enclosed *by $\mathcal{B}$ if for every $\boldsymbol{x} \in dom(\mathcal{B})$ and every Delaunay triangulation Δ of P, $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}, \Delta)$.*

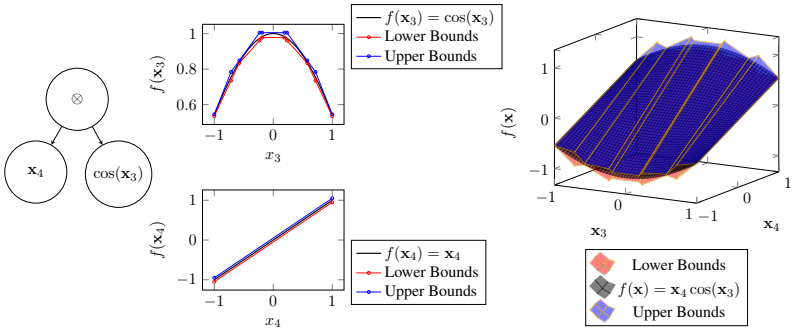

Figure 1: Overview of bounding algorithm for a sample problem. We first bound the univariate functions $\cos(\mathbf{x}_3)$ and $\mathbf{x}_4$, yielding piecewise-linear lower and upper bounds. We then compose these enclosures using multiplication to obtain the polyhedral enclosure shown on the right.

4.1. Bounding Sets for Univariate Functions

Let $f : \mathbb{R} \to \mathbb{R}$ be a function that is twice differentiable on the interval $[a, b]$. We construct a bounding set enclosing f as follows.

We partition the interval into $m > 1$ subintervals of uniform convexity by identifying all points in $[a, b]$ where the second derivative of f vanishes:

$$z_0 := a,\ z_m := b, \text{ and } \frac{d^2}{dx^2} f(z_i) = 0 \text{ for } i = 1, \ldots, m-1.$$

Note that if $\frac{d^2}{dx^2} f(\boldsymbol{x}) = 0$ for *all* $x \in [a, b]$, then we set $m = 1$. For each subinterval $[z_i, z_{i+1}]$, we use the OVERT algorithm (Sidrane et al., 2022) to compute piecewise-linear upper and lower bounds for f on that interval. Relevant details about the OVERT algorithm are provided in the appendix. Stitching all of them together, we get piecewise-linear functions L and U such that $\forall\, x.\ a \leq x \leq b \implies L(x) \leq f(x) \leq U(x)$. Now, we define the point set P to be the set of all endpoints of line segments in the piecewise-linear functions L and U on the interval $[a, b]$ (including a and b themselves). A bounding set enclosing f is then: $\mathcal{B} := \langle 1, P, L^P, U^P \rangle$.

4.2. Composing Bounding Sets

We introduce a notion of *composition* for bounding sets. Our goal is to show that composition preserves enclosure. Before we can compose two bounding sets, we must ensure that they have identical point sets. This can be done, while preserving enclosure properties, via the processes of *lifting* and *interpolation*. Lifting extends a function to a higher dimensional domain in such a way that the restriction to the original domain is the original function. Interpolation extends a bounding set by adding vertices to its point set in a controlled way. Interpolation does not preserve enclosure properties in general, but it does for bounding sets whose point sets are grids. We thus limit our focus to such bounding sets below. Formal definitions and theorems about lifting and interpolation are provided in the appendix.

We are now ready to consider composing bounding sets. Let $\mathcal{B}^f = \langle n^f, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n^g, P, L^g, U^g \rangle$ be bounding sets, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. We further assume that we have already used lifting and interpolation to obtain bounding sets with

identical point sets. We wish to show that certain compositions of $\mathcal{B}^f$ and $\mathcal{B}^g$ enclose corresponding compositions of f and g. Proofs for all theorems are provided in the appendix.

Definition 6 (Linear Composition of Bounding Sets) *Let P be a grid, 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, and let $\bowtie \in \{+, -\}$. We define $\mathcal{B}^f \bowtie \mathcal{B}^g := \langle n, P, L^{fg}, U^{fg} \rangle$, where, for all $\boldsymbol{x} \in P$,*

$$L^{fg}(\boldsymbol{x}), U^{fg}(\boldsymbol{x}) = \begin{cases} L^f(\boldsymbol{x}) + L^g(\boldsymbol{x}), U^f(\boldsymbol{x}) + U^g(\boldsymbol{x}) & \text{if } \bowtie = +, \\ L^f(\boldsymbol{x}) - U^g(\boldsymbol{x}), U^f(\boldsymbol{x}) - L^g(\boldsymbol{x}) & \text{if } \bowtie = -, \end{cases}$$

Theorem 1 *Let P be a grid, $\mathcal{B}^f = \langle n, P, L^f, U^f \rangle$, and $\mathcal{B}^g = \langle n, P, L^g, U^g \rangle$, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. Then, for $\bowtie \in \{+, -\}$, $\mathcal{B}^f \bowtie \mathcal{B}^g$ encloses $f \bowtie g$.*

The situation is more complex for nonlinear operations. We must define a notion of composition that preserves the enclosure property while retaining the (piecewise) linear structure of bounding sets. We relax the bounds on each grid cell to obtain constant bounds, then perform the nonlinear operation using interval arithmetic, and then take the weakest bound at each grid cell boundary.

Definition 7 (Nonlinear Composition of Bounding Sets) *Let P be a grid, 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, and let $\bowtie \in \{\times, \div\}$. Furthermore, if $\bowtie = \div$, assume that $0 \notin [L^g(\boldsymbol{x}), U^g(\boldsymbol{x})]$ for every $\boldsymbol{x} \in P$. We define the nonlinear composition $\mathcal{B}^f \bowtie \mathcal{B}^g$ as $\langle n, P, L^{fg}, U^{fg} \rangle$, where L^{fg} and U^{fg} are defined as follows. First, for each grid cell $\mathcal{X}$ of P, define:*

$$L^f_{\mathcal{X}}, U^f_{\mathcal{X}} = \min(L^f(\mathcal{X})), \max(U^f(\mathcal{X}))$$
$$L^g_{\mathcal{X}}, U^g_{\mathcal{X}} = \min(L^g(\mathcal{X})), \max(U^g(\mathcal{X}))$$

Use interval arithmetic to define lower and upper bounds for the composition on each grid cell:

$$\mathcal{H}_{\bowtie}(\mathcal{X}) = \{h_1 \bowtie h_2 \,|\, h_1 \in \{L^f_{\mathcal{X}}, U^f_{\mathcal{X}}\}, h_2 \in \{L^g_{\mathcal{X}}, U^g_{\mathcal{X}}\}\}$$
$$L_{\bowtie}(\mathcal{X}) = \min(\mathcal{H}_{\bowtie}(\mathcal{X})),\ U_{\bowtie} = (\mathcal{X}) \max(\mathcal{H}_{\bowtie}(\mathcal{X})).$$

Finally, for $\boldsymbol{x} \in P$, define:

$$L^{fg}(\boldsymbol{x}) = \min(\{L_{\bowtie}(\mathcal{X}) \mid \mathcal{X} \in \mathit{Cells}(P, \boldsymbol{x})\}),$$
$$U^{fg}(\boldsymbol{x}) = \max(\{U_{\bowtie}(\mathcal{X}) \mid \mathcal{X} \in \mathit{Cells}(P, \boldsymbol{x})\}).$$

Theorem 2 *Let P be a grid, $\mathcal{B}^f = \langle n, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n, P, L^g, U^g \rangle$, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. Then, $\mathcal{B}^f \times \mathcal{B}^g$ encloses $f \times g$. Furthermore, if for every Delaunay triangulation Δ of P, $\mathcal{E}(\mathcal{B}^g, \Delta) \cap (z(\boldsymbol{x}) = 0) = \emptyset$ (i.e., the enclosure for g does not intersect the plane corresponding to the constant zero function), then $\mathcal{B}^f \div \mathcal{B}^g$ encloses $f \div g$.*

For product compositions with $n = 2$ (i.e. bilinear functions), we can obtain bounding sets that are more compact using McCormick envelopes (Hijazi, 2019). The McCormick envelope provides the smallest convex relaxation of bilinear functions by intersecting linear halfspace constraints. These envelopes can therefore be used to *refine* the product bounds introduced in Definition 7.

5. The OVERTPoly Algorithm

In this section, we present *OVERTPoly*, an algorithm for verifying nonlinear neural feedback systems using polyhedral enclosures. Given a discrete-time neural feedback system $\mathcal{D}$, we compute bounding sets enclosing each transition function $f_i \in \mathbf{F}$. Using these bounding sets, we compute an overapproximation $\hat{\tau}^{\mathcal{D}}(I)$ of the true system trajectory $\tau^{\mathcal{D}}(I)$ and verify safety by checking that $\hat{\tau}^{\mathcal{D}}(I)$ satisfies a specified reach-avoid property. In the following sections, we provide details about each step of the process.

5.1. Computing Bounding Sets for Nonlinear Dynamics

We compute bounding sets by recursively visiting the syntax tree of f. For constant functions, an exact bounding set can be constructed from the function itself. For univariate functions, we use the method based on OVERT described earlier. In the general case, functions that consist of an arithmetic operator applied to two functions with enclosing bounding sets can be bounded by applying the bounding set composition technique described above.

5.2. Generating an Efficient Optimization Model

Overapproximating $next^{\mathcal{D}}$ requires an efficient combinatorial representation for polyhedral enclosures. Let $\mathcal{B} = \langle n, P, L, U \rangle$ be a bounding set. We employ the aggregated convex combination method (Geißler et al., 2011). Let Δ be a Delaunay triangulation of P, with $\{S_1, \ldots, S_m\}$ denoting the n-simplices in Δ. We would like to represent an arbitrary $\boldsymbol{x} \in dom(P)$ as the convex combination of the vertices of some simplex S_i.

We define a binary vector $\boldsymbol{b}$ of size m to encode the *active* simplex in Δ. In other words, $\boldsymbol{x} \in S_i$ iff $b_i = 1$ for every $i \in [m]$. Let $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_d\}$. We define a convex combination variable $\boldsymbol{\lambda}$ that ranges over all points in P (i.e., $\boldsymbol{\lambda} \in \mathbb{R}^d$). We can then define an integer program $\mathcal{M}$ for $\mathcal{B}$.

$$\sum_{j=1}^{d} \boldsymbol{\lambda}_j = 1, \quad \boldsymbol{\lambda} \succcurlyeq \mathbf{0}, \quad \sum_{i=1}^{m} \boldsymbol{b}_i \leq 1, \quad \boldsymbol{b} \in \{0,1\}^m \tag{4a}$$

$$\boldsymbol{\lambda}_j \leq \sum_{\{i \mid \boldsymbol{p}_j \in \mathbf{vert}(S_i)\}} \boldsymbol{b}_i \quad \text{for } \; j \in [d], \tag{4b}$$

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

$$\overline{y} = \sum_{j=1}^{d} \boldsymbol{\lambda}_j \cdot U(\boldsymbol{p}_j), \quad \underline{y} = \sum_{j=1}^{d} \boldsymbol{\lambda}_j \cdot L(\boldsymbol{p}_j), \quad \underline{y} \leq y \leq \overline{y} \tag{4d}$$

Equation (4a) defines the auxiliary variables $\boldsymbol{\lambda}$ and $\boldsymbol{b}$ and enforces that only one binary variable can be active at a time. Equation (4b) defines the so-called SOS-2 constraint (Beale and Forrest, 1976), which requires that only points from simplices adjacent to the active simplex can have nonzero convex combination weights. Equation (4c) defines the state variable $\boldsymbol{x}$. To complete the program, Equation (4d) defines the polyhedral enclosure induced by $\mathcal{B}$ and Δ, requiring that the output variable y must be inside the enclosure.

We construct a separate mixed integer program for each transition function f_i in $\mathbf{F}$. This yields a set of programs $\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ with variables $\{y_1, \ldots, y_n\}$ representing the outputs of the transition functions. We also construct a program $\mathcal{M}_0$ to represent $\mathbf{u}$, the neural network controller. The construction uses standard neural network encoding techniques, as described in the appendix.

5.3. Computing forward reachable sets

To compute $\tau^{\mathcal{D}}(\mathcal{X}_0)_T$ (i.e. the system state at time step T starting from initial state $\mathcal{X}_0$), we must compute $\mathcal{X}_{t+1} = next^{\mathcal{D}}(\mathcal{X}_t)$ for every $t \in [0..T-1]$. This process is called *forward reachability analysis* (Bansal et al., 2017). Since we deal with nonlinear systems, an exact computation of $\mathcal{X}_{t+1}$ is intractable. Instead, we focus on computing an overapproximation $\hat{\mathcal{X}}_{t+1}$ such that $\mathcal{X}_{t+1} \subseteq \hat{\mathcal{X}}_{t+1}$. To do this, assume $\hat{\mathcal{X}}_t$ is an overapproximation of the set of states at time t. For each $i \in [n]$, we solve the following mixed integer linear program.

$$\min_{\boldsymbol{x} \in \hat{\mathcal{X}}_t} \boldsymbol{x}_i + (y_i + \mathbf{u}(\boldsymbol{x})_i + \epsilon) \cdot \delta \tag{5a}$$

$$\mathcal{M}_0, \ldots, \mathcal{M}_n, \qquad \epsilon \in \mathbf{E} \tag{5b}$$

For each $i \in [n]$, solving the problem in Equation (5) yields a lower bound l_i for $\boldsymbol{x}_i$ in the next time step. Similarly, replacing $\min$ with $\max$ and solving yields an upper bound u_i. We can then define $\hat{\mathcal{X}}_{t+1} = [l_1, u_1] \times \ldots \times [l_n, u_n]$. Constructing $\hat{\tau}^{\mathcal{D}}(I)$ sequentially by repeating this process for each $t \in [0..T-1]$ is called *concrete* reachability analysis (Sidrane et al., 2022).

This approach typically introduces more uncertainty each time we solve Equation (5). Therefore, sequential application of Equation (5) may lead to a coarse overapproximation of the reachable sets, a situation sometimes referred to as *excess conservatism*. To address this, we can instead represent two steps at once. For $i \in [0..n]$, let $\mathcal{M}'_i$ be the same as $\mathcal{M}_i$, except that each variable v appearing in $\mathcal{M}_i$ is replaced by v' in $\mathcal{M}'_i$. To solve the two-step problem, we then solve:

$$\min_{\boldsymbol{x} \in \hat{\mathcal{X}}_t} \boldsymbol{x}'_i + (y'_i + \mathbf{u}'(\boldsymbol{x}')_i + \epsilon') \cdot \delta \tag{6}$$

$$\mathcal{M}_0, \ldots, \mathcal{M}_n \tag{7}$$

$$\text{for each } i \in [n] \begin{cases} \boldsymbol{x}'_i = \boldsymbol{x}_i + (y_i + \mathbf{u}(\boldsymbol{x})_i + \epsilon_i), & \epsilon_i \in \mathbf{E} \end{cases} \tag{8}$$

$$\mathcal{M}'_0, \ldots, \mathcal{M}'_n \quad \epsilon' \in \mathbf{E} \tag{9}$$

This yields lower bounds for $\boldsymbol{x}_{t+2}$ (as before, replacing $\min$ by $\max$ produces upper bounds). This process can be repeated to generalize from a two-step analysis to a k-step analysis for arbitrary k. Constructing $\hat{\tau}^{\mathcal{D}}(I)$ this way is called *symbolic* reachability analysis (Sidrane et al., 2022).

The key challenge with symbolic reachability is scalability. This can be partly addressed by tracking dependencies with a dependency graph. The models $\mathcal{M}_0, \ldots, \mathcal{M}_n$ are the vertices, and there is an edge between $\mathcal{M}_i$ and $\mathcal{M}_j$ if the output of $\mathcal{M}_i$ depends on the previous output value of $\mathcal{M}_j$ or vice versa. We use dependency graphs to prune the set of models included in each integer linear program. We provide an example illustrating the construction and use of dependency graphs in the appendix.

6. Evaluation

	OVERTPoly		OVERTVerify		CORA	
	Time (s)	Volume	Time (s)	Volume	Time (s)	Volume
Single Pendulum	1.12	5.27E−2	7.33E−1	**5.24E−2**	**7.10E−1**	1.84
Adaptive Cruise Control (ACC)	12.28	1.34E−2	110.91	**1.32E−2**	**4.78**	6.84E+5
Translation Oscillation (TORA)	**561.58**	6.66E−1	983.76	**6.43E−1**	×	6.03E + 02
Unicycle Car Model	**3940.66**	1.56E−5	16348.38	**9.78E−6**	×	×

Table 1: Benchmark computation time (s) and set volumes. Computation times are listed for verified instances, and × indicates an unverified instance. All available set volumes are shown. Best performance is highlighted in bold.

We evaluate the performance of our tool by comparing it to a discrete-time implementation of CORA (Kochdumper and Althoff, 2023), a state-of-the-art propagation based tool, and OVERTVerify (Sidrane et al., 2022), a state-of-the-art combinatorial tool. We use the Polynomial Zonotope abstraction (Kochdumper and Althoff, 2023) in CORA. We note that the performance of the CORA algorithm depends on the choice of hyperparameters, and we report results for the lowest-precision settings needed to verify a given property. When CORA is unable to verify a property, we report results for the highest-precision settings.

6.1. Benchmarks

We test all algorithms on a subset of the benchmarks used in the annual ARCH-Competition (Lopez et al., 2023). Since we only support ReLU networks, we restrict ourselves to benchmarks that use ReLU networks. We describe pertinent details about these benchmarks in the appendix.

6.2. Results

Results are reported in Section 6. For each benchmark, we report the amount of time required to verify (or falsify) the specified property, as well as the volume of the reachable set at time T. Since the CORA algorithm does not support volume computation for polynomial zonotopes, we approximate the final set using grid-based sampling. Despite employing a combinatorial approach, our computation time is comparable to state-of-the-art propagation-based methods.

Single Pendulum: The Single Pendulum benchmark was used as a baseline for all tools. All tools completed the analysis in under two seconds, with CORA delivering the fastest results, and OVERTVerify achieving the highest precision.

ACC: OVERTPoly and CORA are able to verify the ACC benchmark in under 15 seconds, while OVERTVerify required around 110 seconds to verify the same property. CORA delivered the fastest results, while OVERTVerify delivered the most precise results. At the cost of a 2% increase in conservatism, OVERTPoly achieved a 9x improvement in computation time. The volume computed by CORA is large, but the tool is still able to verify the benchmark because the property does not depend on the magnitude of the state variables.

TORA: CORA is unable to verify the TORA benchmark. We tried all available hyperparameter combinations, but the set generated by CORA was too large to verify the property. OVERTPoly and OVERTVerify are both able to verify the property. OVERTPoly was able to verify the property almost twice as fast as OVERTVerify, at the cost of a 4% increase in conservatism.

Unicycle: CORA is similarly unable to verify the Unicycle benchmark (the tool crashes due to state explosion), while OVERTPoly and OVERTVerify require a combination of concrete and symbolic approaches to verify the property. OVERTPoly is able to verify the property around 4 times faster than OVERTVerify. OVERTPoly computes a set 59% larger than OVERTVerify's, making verification more challenging. This conservatism may be reduced by adjusting k, the number of symbolic steps used in the analysis.

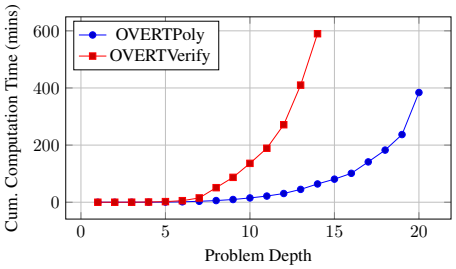

Figure 2: Symbolic reachability comparison between OVERTPoly and OVERTVerify using the Unicycle benchmark. Each tool was used to compute as many symbolic reachable steps as possible with a (per optimizer call) timeout of 3600 seconds.

Symbolic Reachability: Because symbolic reachability trades off precision for computational tractability, it is an ideal test of the scalability of combinatorial algorithms. Figure 2 highlights the scaling differences between OVERTPoly and OVERTVerify when running symbolic reachability at different depths. The figure shows that as the number of steps increases, OVERTPoly's scalability improves by up to several orders of magnitude.

7. Conclusion

In this work, we showed that using bounding sets and polyhedral enclosures provides a scalable abstraction for verifying reach-avoid properties of nonlinear neural feedback systems. Polyhedral enclosures (along with neural network controllers) are encoded as mixed integer linear programs, enabling forward reachability analysis of discrete time systems. These techniques are integrated in the OVERTPoly algorithm, which shows a significant improvement in both computation time and precision when compared to existing tools. These improvements come at the cost of computational complexity at higher dimensions. The improved scalability of precise verification tools enables the verification of more realistic neural feedback systems, which is a promising step toward safer autonomous transportation systems.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. (2146755). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Additional support was provided by NSF Grant No. 2211505 and by the Stanford Center for Automated Reasoning.

References

I Samuel Akinwande, Chelsea Sidrane, Mykel J Kochenderfer, and Clark Barrett. Verifying nonlinear neural feedback systems using polyhedral enclosures. *arXiv preprint arXiv:2503.22660*, 2025.

Matthias Althoff and Niklas Kochdumper. Cora 2016 manual. *TU Munich*, 85748, 2016.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

Evelyn ML Beale and John JH Forrest. Global optimization using special ordered sets. *Mathematical programming*, 10(1):52–69, 1976.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, pages 258–263. Springer, 2013.

Thai Duong, Abdullah Altawaitan, Jason Stanley, and Nikolay Atanasov. Port-hamiltonian neural ode networks on lie groups for robot dynamics learning and control. *arXiv preprint arXiv:2401.09520*, 2024.

Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 157–168, 2019.

Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles R Qi, Yin Zhou, et al. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9710–9719, 2021.

Michael Everett. Neural network verification in control. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6326–6340. IEEE, 2021.

Michael Everett, Golnaz Habibi, Chuangchuang Sun, and Jonathan P How. Reachability analysis of neural feedback loops. *IEEE Access*, 9:163938–163953, 2021.

Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 537–542. Springer, 2020.

Björn Geißler, Alexander Martin, Antonio Morsi, and Lars Schewe. Using piecewise linear functions for solving minlps. In *Mixed integer nonlinear programming*, pages 287–314. Springer, 2011.

Shashi Gowda, Yingbo Ma, Alessandro Cheli, Maja Gwóźzdź, Viral B. Shah, Alan Edelman, and Christopher Rackauckas. High-performance symbolic-numerics via multiple dispatch. *ACM*

Commun. Comput. Algebra, 55(3):92–96, jan 2022. ISSN 1932-2240. doi: 10.1145/3511528.3511535. URL https://doi.org/10.1145/3511528.3511535.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

Hassan Hijazi. Perspective envelopes for bilinear functions. In *AIP conference proceedings*, volume 2070, page 020017. AIP Publishing LLC, 2019.

Chao Huang, Jiameng Fan, Xin Chen, Wenchao Li, and Qi Zhu. Polar: A polynomial arithmetic framework for verifying neural-network controlled systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 414–430. Springer, 2022.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verifying the safety of autonomous systems with neural network controllers. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(1):1–26, 2020.

Jordan Jalving, Yankai Cao, and Victor M Zavala. Graph-based modeling and simulation of complex systems. *Computers & Chemical Engineering*, 125:134–154, 2019.

Jordan Jalving, Sungho Shin, and Victor M. Zavala. A graph-based modeling abstraction for optimization: concepts and implementation in Plasmo.jl. *Mathematical Programming Computation*, 14:699–747, 2022. doi: 10.1007/s12532-022-00223-3.

JuliaIntervals. Intervalrootfinding.jl. https://github.com/JuliaIntervals/IntervalRootFinding.jl, 2024. Accessed: 2024-11-18.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Niklas Kochdumper and Matthias Althoff. Constrained polynomial zonotopes. *Acta Informatica*, pages 1–38, 2023.

Niklas Kochdumper, Christian Schilling, Matthias Althoff, and Stanley Bak. Open-and closed-loop neural network verification using polynomial zonotopes. In *NASA Formal Methods Symposium*, pages 16–36. Springer, 2023.

Věra Kurková. Kolmogorov's theorem is relevant. *Neural Computation*, 3(4):617–622, 1991.

Changliu Liu, Tomer Arnon, Christopher Lazarus, Christopher Strong, Clark Barrett, and Mykel J. Kochenderfer. Algorithms for verifying deep neural networks, 2020. URL https://arxiv.org/abs/1903.06758.

Diego Manzanas Lopez, Matthias Althoff, Marcelo Forets, Taylor T Johnson, Tobias Ladner, and Christian Schilling. Arch-comp23 category report: Artificial intelligence and neural network control systems (ainncs) for continuous and hybrid systems plants. In *EPiC Series in Computing*, 2023.

Jean-Michel Muller and Jean-Michael Muller. *Elementary functions*. Springer, 2006.

Christian Schilling, Marcelo Forets, and Sebastián Guadalupe. Verification of neural-network control systems by integrating Taylor models and zonotopes. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 8169–8177, 2022. doi: 10.1609/aaai.v36i7.20790. URL https://ojs.aaai.org/index.php/AAAI/article/view/20790.

Chelsea Sidrane, Amir Maleki, Ahmed Irfan, and Mykel J Kochenderfer. Overt: An algorithm for safety verification of neural network control policies for nonlinear systems. *Journal of Machine Learning Research*, 23(117):1–45, 2022.

Jacob A Siefert, Trevor J Bird, Justin P Koeln, Neera Jain, and Herschel C Pangborn. Successor sets of discrete-time nonlinear systems using hybrid zonotopes. In *2023 American Control Conference (ACC)*, pages 1383–1389. IEEE, 2023.

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

Claire J Tomlin, Ian Mitchell, Alexandre M BAYEN, and Meeko Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

Joseph A Vincent and Mac Schwager. Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9029–9035. IEEE, 2021.

Yixuan Wang, Weichao Zhou, Jiameng Fan, Zhilu Wang, Jiajun Li, Xin Chen, Chao Huang, Wenchao Li, and Qi Zhu. Polar-express: Efficient and precise formal reachability analysis of neural-network controlled systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

Yuhao Zhang and Xiangru Xu. Reachability analysis and safety verification of neural feedback systems via hybrid zonotopes. In *2023 American Control Conference (ACC)*, pages 1915–1921. IEEE, 2023.

Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

Appendix A. Details about Technical Concepts Cited

We review technical tools and concepts used in this work, all of which are established in prior literature.

A.1. Bounding Convex Functions

To compute tight upper and lower bounds for nonlinear one-dimensional functions, we use a method introduced in the OVERT algorithm Sidrane et al. (2022). For a function $f(\boldsymbol{x})$ that is convex on the interval (a, b), an upper bound can be constructed by symbolically partitioning (a, b) into $m > 1$, subintervals with endpoints $(s_0 = a, s_1, \ldots, s_{m-1}, s_m = b)$ and defining secant lines from $(s_{i-1}, f(s_{i-1}))$ to $(s_i, f(s_i))$ for $i \in [m]$. The point locations s_i are then optimized to minimize the area between the secant lines and the function $f(\boldsymbol{x})$. A lower bound can be constructed similarly by partitioning (a, b) into subintervals with endpoints s_i, $i \in [0..m]$ and defining a sequence of line segments from (s_i, t_i) to (s_{i+1}, t_{i+1}), $i \in [0..m-1]$ in such a way that $t_0 = f(a), t_m = f(b)$, each segment from (s_i, t_i) (s_{i+1}, t_{i+1}), $i \in [m-2]$ is tangent to $f(\boldsymbol{x})$, and the area between the line segments and the function is once again minimized. Bounds for a univariate function $f(\boldsymbol{x})$ that is concave over an interval are computed analogously, using a series of secants for lower bounds and a series of tangents for upper bounds. The tightness of the bounds can be adjusted by modifying the parameter m. Bounds over an arbitrary interval can be computed by first decomposing the function into intervals of uniform convexity and then composing the bounds obtained for each region. Details of these methods are provided by Sidrane et al. (2022).

A.2. ReLU Functions as Mixed Integer Constraints

A *feed-forward neural network* (FNN) is a function constructed by composing linear and nonlinear operations. These operations are organized in *layers*, where each layer applies a linear transformation and (optionally) a nonlinear operation. The elements within a layer are referred to as *neurons*, and the nonlinear operations are known as *activation functions*. In this work, we focus on the Rectified Linear Unit (ReLU) activation function, defined as $\mathsf{ReLU}(x) = \max(x, 0)$. We refer to FNNs using only this activation function as ReLU-*activated feed-forward neural networks*.

ReLU activated FNNs specify piecewise-linear functions and can therefore be formulated as mixed-integer linear programs (MILPs). Following the technique used by the MIPVerify algorithm Tjeng et al. (2017), we encode ReLU constraints as follows. Let $l, u \in \mathbb{R}$ be real numbers intended to bound x, with $l \leq 0 \leq u$,[3] and let $\mathbf{1}_{(x \geq 0)}$ be an indicator variable. Then both the ReLU function $y = \max(x, 0)$ and the bounds $l \leq x \leq u$ can be represented by the constraints

$$y \leq x - l(1 - \mathbf{1}_{(x \geq 0)}), \quad y \geq x, \quad y \leq u \cdot \mathbf{1}_{(x \geq 0)}, \quad y \geq 0. \tag{10}$$

To encode a ReLU-activated FNN, the linear layers are expressed as standard linear constraints, while the nonlinear ReLU constraints use Equation (10).

A.3. Pruning Models Using Dependency Graphs

We include the ILP $\mathcal{M}_j$ in the optimization problem for a state variable $\boldsymbol{x}_i$ if $\mathcal{M}_i$ and $\mathcal{M}_j$ share an edge in the state dependency graph of the problem. However, when solving a symbolic reachability

3. Note that in the case that $l \leq u < 0$ or $0 < l \leq u$, a much simpler linear encoding is possible.

problem over k steps, the model at step k ($\mathcal{M}_i^k$) depends not only on its immediate neighbors in the graph, but also on its own history, specifically, on $\mathcal{M}_i^{k-1}, \mathcal{M}_i^{k-2}, \ldots, \mathcal{M}_i^0$. This results in the extended dependency structure illustrated in Figure 4. To manage these dependencies efficiently, we use the Plasmo framework (Jalving et al., 2022), which automatically prunes irrelevant ILPs based on the structure of the provided graph.

A.4. Tightening Nonlinear Composition

The nonlinear composition introduced in Definition 16 is sound but may be conservative in practice. In cases where the composition yields a bilinear term of the form $z = x \cdot y$, we can tighten the bounds using the McCormick envelope (Hijazi, 2019).

$$
\begin{aligned}
z &\geq x^{\ell} y + y^{\ell} x - x^{\ell} y^{\ell} \\
z &\geq x^{u} y + y^{u} x - x^{u} y^{u} \\
z &\leq x^{\ell} y + y^{u} x - x^{\ell} y^{u} \\
z &\leq x^{u} y + y^{\ell} x - x^{u} y^{\ell}
\end{aligned}
$$

where $x \in [x^{\ell}, x^{u}]$ and $y \in [y^{\ell}, y^{u}]$.

Appendix B. Illustrative Examples

In this section, we present examples that illustrate key concepts from the theory of polyhedral enclosures. We begin with a running example and then demonstrate how the ideas developed throughout the paper apply to it.

B.1. Illustrative Example: Unicycle Car Model

As a running example, we use a discrete-time version of the unicycle car model example from the 2023 ARCH competition (Lopez et al., 2023). The car is modeled with four variables, representing the x and y coordinates in a plane, the steering angle, and the velocity magnitude. Formally, we define $\mathcal{U} = \langle n^{\mathcal{U}}, \mathbf{I}^{\mathcal{U}}, \mathbf{F}^{\mathcal{U}}, \mathbf{E}^{\mathcal{U}}, \mathbf{u}^{\mathcal{U}}, \delta^{\mathcal{U}}, T^{\mathcal{U}}, \mathbf{G}^{\mathcal{U}}, \mathbf{A}^{\mathcal{U}} \rangle$, where $n^{\mathcal{U}} = 4$, $\mathbf{I}^{\mathcal{U}} = [9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51]$, $\mathbf{F}^{\mathcal{U}}(\boldsymbol{x}) = (\boldsymbol{x}_4 \cos(\boldsymbol{x}_3), \boldsymbol{x}_4 \sin(\boldsymbol{x}_3), 0, 0)$, $\mathbf{E} = \{0\} \times \{0\} \times \{0\} \times [-10^{-4}, 10^{-4}]$, $\mathbf{u}^{\mathcal{U}}$ is computed by a neural network with one hidden layer with 500 neurons and four outputs, the first two of which are set to the constant zero value (i.e., the controller only affects the velocity and steering), $\delta^{\mathcal{U}} = 0.2$, $T^{\mathcal{U}} = 50$, $\mathbf{G}^{\mathcal{U}} = [-0.6, 0.6] \times [-0.2, 0.2] \times [-0.06, 0.06] \times [-0.3, 0.3]$, and $\mathbf{A}^{\mathcal{U}}(t) = \emptyset$ for every $t \in [0..T^{\mathcal{U}}]$. At each step, the system updates its x and y coordinates based on the steering and velocity control outputs, where the velocity output is subject to a small amount of perturbation. It must reach a goal position near the origin with a steering angle near 0 and with a low velocity. Formally, we would like to verify that the reach property holds for $\mathcal{U}$. Figure 1 demonstrates the composition process applied to the transition function for the first state variable in this running example. We begin by bounding each univariate function independently, and then apply the composition algorithm to obtain multivariate bounds. Figure 3 depicts the state dependency graph generated by the transition functions of this example.

Example 1 *(Bounding Set) Let* $\boldsymbol{p}_1 = (-5, -5)$, $\boldsymbol{p}_2 = (-5, 5)$, $\boldsymbol{p}_3 = (5, -5)$, $\boldsymbol{p}_4 = (5, 5)$ *be points in* $\mathbb{R}^2$ *and let* $P_1 = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4\}$. *Then, let* $\mathcal{B}^1$ *be the bounding set* $\langle 2, P_1, L_1, U_1 \rangle$, *with* $U_1(\boldsymbol{p}) = 5$ *and* $L_1(\boldsymbol{p}) = -5$ *for every* $\boldsymbol{p} \in P_1$. *The domain of* $\mathcal{B}^1$, $dom(\mathcal{B}^1)$, *is a square with sides of length* 10 *centered at the origin.*

Example 2 *(Polyhedral Enclosure) Consider again the bounding set* $\mathcal{B}_1$ *from Example 1. Let* Δ_1 *be the triangulation of* P_1 *whose 2-simplices are the triangles* T_1, *with vertices* $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, *and* T_2, *with vertices* $\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4$. *Then,* $\mathcal{P}(\mathcal{B}^1_{T_1})$ *and* $\mathcal{P}(\mathcal{B}^1_{T_2})$ *are the two prisms obtained by slicing the polyhedron formed by* $\mathcal{B}^1$ *in half along the plane* $x_1 + x_2 = 0$. *And* $\mathcal{E}(\mathcal{B}^1, \Delta_1)$ *is their union, which, in this case, is again just the polyhedron formed by* $\mathcal{B}^1$.

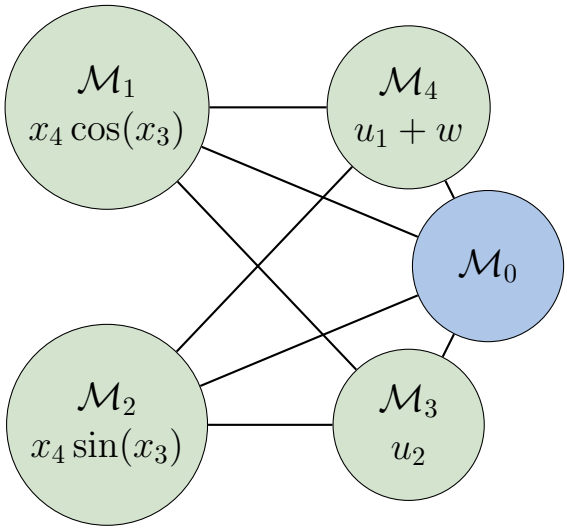

Figure 3: Dependency graph for Unicycle example. The green vertices represent state variables, while the blue vertex represents the neural network controller. Note that each variable is connected to the controller because the neural network depends on the full state.

Example 3 *(Function Enclosure) Let* $f(x_1, x_2) = x_2 \cos(x_1)$. *The maximum value of* f *on* $dom(\mathcal{B}^1)$ *is* 5 *and the minimum value is -5.* $\mathcal{E}(\mathcal{B}^1, \Delta)$ *is the origin-centered cube with side length* 10 *for every Delaunay point set triangulation* Δ *of* P_1. *Thus,* $\mathcal{B}^1$ *encloses* f.

Example 4 *(Lifted Function) Let* f *be the function from Example 3, and let* $S = \{3, 4\}$. *Then, if* $\boldsymbol{y} = (y_1, y_2, y_3, y_4)$, $f{\uparrow}^{S,4}(y) = y_4 \cos(y_3)$.

Example 5 *(Lifted Bounding Sets) Let* $\mathcal{B}^1$ *be the bounding set from Example 1, let* $S = \{1, 2\}$, *let* $\boldsymbol{p}^u = (0, 0, 5)$, *and let* $\boldsymbol{p}^l = (0, 0, -5)$. *Then,* $\mathcal{B}{\uparrow}^{S,3}_{\boldsymbol{p}^l, \boldsymbol{p}^u}$ *is a bounding set whose domain is the cube with sides of length 10 centered at the origin, and the polyhedron formed by the lifted bounding set is a hypercube in* $\mathbb{R}^4$ *also centered at the origin.*

Example 6 *(Interpolated Function) Let* $\mathcal{B}^1$ *be the bounding set from Example 1, and let* Δ *be the triangulation of* P_1 *whose 2-simplices are the triangles* T_1, *with vertices* $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, *and* T_2, *with vertices* $\boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4$. *Let* $\boldsymbol{q} = (0, 0)$, *and let* $P'_1 = P_1 + \boldsymbol{q} = P_1 \cup \{(-5, 0), (0, 0), (5, 0), (0, -5), (0, 5)\}$.

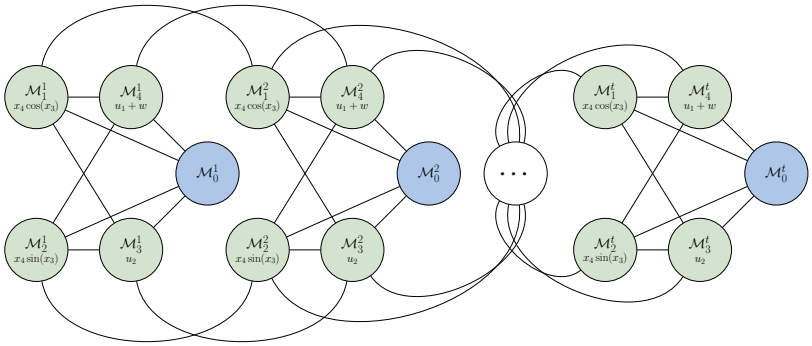

Figure 4: Symbolic dependency graph for Unicycle example. We encode temporal dependencies using edges between state dependency graphs

Then, let $U_1' = U_1 +_\Delta \boldsymbol{q}$. *Let* $\boldsymbol{x} = (-5, 0)$. *To compute* $U_1'(\boldsymbol{x})$, *we first note that* $\mathcal{S}_\Delta(\boldsymbol{x}) = T_1$, $\mathbf{vert}(T_1) = \{\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3\}$, *and* $\theta(\boldsymbol{x}) = (0.5, 0.5, 0)$. *Thus,* $U_1'(\boldsymbol{x}) = \theta(\boldsymbol{x}) \cdot U_1(\mathbf{vert}(T_1)) = (0.5, 0.5, 0) \cdot (5, 5, 5) = 5$.

Example 7 *(Interpolated Bounding Set) Let* $\mathcal{B}^1$ *be the bounding set from Example 1,* Δ *the triangulation from Example 6, and* $\boldsymbol{q} = (0, 0)$. *Then* $\mathcal{B}^1 +_\Delta \boldsymbol{q} = \langle 2, P_1', L_1 +_\Delta \boldsymbol{q}, U_1' \rangle$, *where* P_1' *and* U_1' *are as in Example 6, and* $L_1 +_\Delta \boldsymbol{q}$ *is computed similarly.*.

Appendix C. Formal Definitions

C.1. Polyhedra

Let P be a set of $k+1$ points: $P = \{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_k\}$, with each $\boldsymbol{p}_i \in \mathbb{R}^n$. The *convex hull* of P is

$$\mathbf{conv}(P) = \{\boldsymbol{\theta}_0 \boldsymbol{p}_0 + \ldots + \boldsymbol{\theta}_k \boldsymbol{p}_k \mid \boldsymbol{\theta} \cdot \mathbf{1} = 1, \boldsymbol{\theta} \succcurlyeq \mathbf{0}\}. \tag{11}$$

The points in P are *affinely independent* iff the set $\{\boldsymbol{p}_1 - \boldsymbol{p}_0, \ldots, \boldsymbol{p}_k - \boldsymbol{p}_0\}$ is linearly independent. The *polyhedron formed by* a set of points P is just $\mathbf{conv}(P)$. A subset of $\mathbb{R}^n$ is a polyhedron if it is the convex hull of a finite set of points in $\mathbb{R}^n$.[4]

Definition 8 (k-Simplex) *A polyhedron* $\mathcal{S}$ *is a* k*-simplex if it is the convex hull of* $k+1$ *affinely independent points. A polyhedron is a simplex if it is a* k*-simplex for some* k, *and* k *is called its* dimension.

If $\mathcal{S}$ is a k-simplex, let $\mathbf{vert}(\mathcal{S})$, the *vertices* of $\mathcal{S}$, denote the (unique) set of $k+1$ points P such that $\mathcal{S} = \mathbf{conv}(P)$. A *face* of $\mathcal{S}$ is the convex hull of any non-empty subset of $\mathbf{vert}(\mathcal{S})$.

Definition 9 (Simplicial k-Complex) *A* simplicial complex Δ *is a set of simplices such that:*

4. Sometimes the term "polyhedron" is reserved for three-dimensional objects, with the generalization to arbitrary dimensions called a "polytope." We use "polyhedron" also for the general case (Boyd and Vandenberghe, 2004; Ziegler, 2012).

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

Δ is a *pure simplicial k-complex* if the largest dimension of any simplex in Δ is k (also called the *dimension* of Δ) and if every simplex in Δ of dimension less than k is a face of some simplex in Δ of dimension k.

Definition 10 (Full-Dimensional) *Let P be a finite set of points in $\mathbb{R}^n$. We say that P is* full-dimensional *if it contains $n+1$ affinely independent points.*

Definition 11 (Point Set Triangulation) *If P is a finite, full-dimensional set of points in $\mathbb{R}^n$, then a pure simplicial n-complex Δ is a* triangulation of P *if $P = \bigcup_{\mathcal{S}\in\Delta} \mathbf{vert}(\mathcal{S})$ and $\mathbf{conv}(P) = \bigcup_{\mathcal{S}\in\Delta} \mathcal{S}$.*

Let C be a closed n-ball. We use C^O to denote the corresponding open n-ball and C^S to denote the hypersphere that forms the surface of C. The vertices—with respect to a set P of points—of C are defined as $V_P(C) = C \cap P$. For a polyhedron $\mathcal{P}$, the *circumsphere* of $\mathcal{P}$ (when it exists) is a hypersphere that touches all of the vertices of $\mathcal{P}$. Note that circumspheres always exist for simplices and for hyperrectangles.

Now, let P be a finite, full-dimensional set of points in $\mathbb{R}^n$, let Δ be a triangulation of P, and let $\mathcal{S}$ be an n-simplex in Δ. We define $C(\mathcal{S})$ to be the n-ball whose boundary is the circumsphere of $\mathcal{S}$. $\mathcal{S}$ satisfies the *Delaunay condition* and is called a *Delaunay simplex of P* if $V_P(C(\mathcal{S})^O) = \emptyset$ (i.e., the only points from P contained in $C(\mathcal{S})$ are on its surface). Δ is a *Delaunay triangulation* if every n-simplex in Δ satisfies the Delaunay condition. A set of points in $\mathbb{R}^n$ has a Delaunay triangulation of dimension n iff it is full-dimensional.

For a point $\boldsymbol{x} \in \mathbf{conv}(P)$, let $\mathcal{S}_\Delta(\boldsymbol{x})$ be the n-simplex in Δ containing $\boldsymbol{x}$ (if $\boldsymbol{x}$ is in more than one n-simplex, which can only occur when it is on a face, we assume $\mathcal{S}_\Delta(\boldsymbol{x})$ chooses one of the simplices in a deterministic way). We define $\theta_\Delta(\boldsymbol{x})$ to be the convex combination vector such that $\boldsymbol{x} = \theta_\Delta(\boldsymbol{x}) \cdot \mathbf{vert}(\mathcal{S}_\Delta(\boldsymbol{x}))$.

Appendix D. Composition Details

Let $\mathcal{B}^f = \langle n^f, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n^g, P, L^g, U^g \rangle$ be bounding sets, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. We want to define a method to compose $\mathcal{B}^f$ and $\mathcal{B}^g$ in a manner that preserves enclosure.

The first step is to lift f and g to some dimension n. We have to decide how we want to embed $\mathbb{R}^{n^f}$ and $\mathbb{R}^{n^g}$ into $\mathbb{R}^n$. We can choose any n such that $\max(n^f, n^g) \leq n \leq n^f + n^g$. The embedding is done via sets S^f and S^g. $S^f \subseteq [n]$ lists the positions taken by the arguments to f in a full list of n variables, and the same is true for S^g.

Definition 12 (Lifted Function) *Let $f : \mathbb{R}^k \to \mathbb{R}$, let $n > k$, and let $S = \{j_1, \ldots, j_k\}$ be a subset of $[n]$ with $j_i < j_{i+1}$ for each $i \in [k-1]$. We define f* lifted to n by S, *$f{\uparrow}^{S,n}$ as follows. For each $\boldsymbol{y} \in \mathbb{R}^n$, $(f{\uparrow}^{S,n})(\boldsymbol{y}) = f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^k$ and for each $i \in [k]$, $x_i = y_{j_i}$.*

Definition 13 (Lifted Bounding Set) *Let $\mathcal{B} := \langle k, P, L, U \rangle$ be a bounding set, let $n > k$, let $S = \{j_1, \ldots, j_k\}$ be a subset of $[n]$ with $j_i < j_{i+1}$ for each $i \in [k-1]$, and let $\boldsymbol{p}^l, \boldsymbol{p}^u \in \mathbb{R}^n$, with*

$\boldsymbol{p}_i^l < \boldsymbol{p}_i^u$ *for* $i \in [n] \setminus S$. *We define* $\mathcal{B}$ lifted to n by S from $\boldsymbol{p}^l$ to $\boldsymbol{p}^u$, $\mathcal{B}{\uparrow}^{S,n}_{\boldsymbol{p}^l,\boldsymbol{p}^u}$ *as follows.* $\mathcal{B}{\uparrow}^{S,n}_{\boldsymbol{p}^l,\boldsymbol{p}^u} = \langle n, P', L{\uparrow}^{S,n}, U{\uparrow}^{S,n}\rangle$, *where* $P' = \{\boldsymbol{p}' \in \mathbb{R}^n \mid \exists \boldsymbol{p} \in P.\, \boldsymbol{p}'_i = \boldsymbol{p}_i$ *if* $i \in S$ *and* $\boldsymbol{p}'_i \in \{\boldsymbol{p}_i^l, \boldsymbol{p}_i^u\}$ *otherwise* .

Note that the purpose of $\boldsymbol{p}^l$ and $\boldsymbol{p}^u$ is to provide lower and upper bounds for the new dimensions added when lifting the points in P, to ensure that P' is full-dimensional.

The theorem below states that lifting preserves function enclosure. Due to limited space, proofs for this and the following theorems are provided in the appendix.

Theorem 3 *If a bounding set* $\mathcal{B} = \langle k, P, L, U\rangle$ *encloses a function* f*, then every lifted bounding set encloses the corresponding lifted function. Formally, for every* $n > k$, $S \subset [n]$ *with* $|S| = k$, *and* $\boldsymbol{p}^l, \boldsymbol{p}^u \in \mathbb{R}^n$, *with* $\boldsymbol{p}_i^l < \boldsymbol{p}_i^u$ *for* $i \in [n] \setminus S$, $\mathcal{B}{\uparrow}^{S,n}_{\boldsymbol{p}^l,\boldsymbol{p}^u}$ *encloses* $f{\uparrow}^{S,n}$.

Then, $f' = f{\uparrow}^{S^f,n}$ and $g' = g{\uparrow}^{S^g,n}$ are lifted versions of f and g, respectively, that both map from $\mathbb{R}^n$ to $\mathbb{R}$. To lift the bounding sets, we define $\boldsymbol{p}^l$, $\boldsymbol{p}^u$ to be vectors of size n such that $\boldsymbol{p}_i^l = \min(P_i^f \cup P_i^g)$ and $\boldsymbol{p}_i^u = \max(P_i^f \cup P_i^g)$, for $i \in [n]$. Now, let $\mathcal{B}^{f'} = \langle n, P^{f'}, L^{f'}, U^{f'}\rangle = \mathcal{B}^f{\uparrow}^{S^f,n}_{\boldsymbol{p}^l,\boldsymbol{p}^u}$ and $\mathcal{B}^{g'} = \langle n, P^{g'}, L^{g'}, U^{g'}\rangle = \mathcal{B}^g{\uparrow}^{S^g,n}_{\boldsymbol{p}^l,\boldsymbol{p}^u}$. We know that $\mathcal{B}^{f'}$ and $\mathcal{B}^{g'}$ enclose f' and g', respectively, by Theorem 3. At this point, we need $dom(\mathcal{B}^{f'}) = dom(\mathcal{B}^{g'})$. Notice that because of the way we chose $\boldsymbol{p}^l$ and $\boldsymbol{p}^u$ ‘, this will be the case unless there is some $j_{if} \in S^f$ and $j_{ig} \in S^g$ such that $j_{if} = j_{ig}$ and $\min(P^f_{if}) \neq \min(P^g_{ig})$ or $\max(P^f_{if}) \neq \max(P^g_{ig})$. To ensure this doesn't happen, we assume that each dimension has fixed lower and upper bounds. In particular, for neural feedback systems, we assume that the initial state set $\mathbf{I}$ bounds each individual state variable. Next, we need both bounding sets to use the same point set. We achieve this by defining $\mathcal{B}^{f''}$ as follows. Let $\{\boldsymbol{p_1}, \ldots, \boldsymbol{p_m}\}$ be an enumeration of the points in $P^{g'} \setminus P^{f'}$. Define $\mathcal{B}_0^{f''} = \mathcal{B}^{f'}$, and define $\mathcal{B}_i^{f''} = \mathcal{B}_{i-1}^{f''} +_{\Delta_i} \boldsymbol{p_i}$, for $i \in [m]$, where Δ_i is an arbitrary Delaunay triangulation of the point set of $\mathcal{B}_{i-1}^{f''}$. We then set $\mathcal{B}^{f''} = \mathcal{B}_m^{f''}$. We define $\mathcal{B}^{g''}$ similarly as the result of inserting into the point set of $\mathcal{B}^{g'}$ any missing points from the point set of $\mathcal{B}^{f''}$. The resulting bounding sets, $\mathcal{B}^{f''}$ and $\mathcal{B}^{g''}$ are defined over the same point set and still enclose f' and g', respectively, by Theorem 4.

Definition 14 (Grid Expansion) *Let* G *be an* n*-dimensional grid, and let* $\boldsymbol{q} \in dom(G)$. *Let* G'_i *be the set* $G_i \cup \{q_i\}$. *Then* G *expanded by* $\boldsymbol{q}$, *written* $G + \boldsymbol{q}$, *is the grid* $G'_1 \times \cdots \times G'_n$.

Definition 15 (Interpolated Function) *Let* $P \subset \mathbb{R}^n$ *be a grid,* $f : P \to \mathbb{R}$ *a function, and* Δ *a Delaunay triangulation of* P. *Let* $\boldsymbol{q} \in dom(P)$, *and let* $P' = P + \boldsymbol{q}$.

Then, f interpolated by $\boldsymbol{q}$ using Δ, *written* $f +_\Delta \boldsymbol{q}$ *is the function* $f' : P' \to \mathbb{R}$ *defined as:*

$$(f +_\Delta \boldsymbol{q})(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) \text{ if } \boldsymbol{x} \in P, \text{ and} \\ \theta_\Delta(\boldsymbol{x}) \cdot f(\mathbf{vert}(\mathcal{S}_\Delta(\boldsymbol{x}))) \text{ otherwise.} \end{cases} \tag{12}$$

Definition 16 (Interpolated Bounding Set) *Let* $\mathcal{B} = \langle n, P, L, U\rangle$ *be a bounding set, with* P *a grid, let* Δ *be a Delaunay triangulation of* P, *and let* $\boldsymbol{q} \in dom(P)$. *Then* $\mathcal{B}$ interpolated by $\boldsymbol{q}$ using Δ, *written* $\mathcal{B} +_\Delta \boldsymbol{q}$, *is defined as:*

$$\mathcal{B} +_\Delta \boldsymbol{q} = \langle n, P + \boldsymbol{q}, L +_\Delta \boldsymbol{q}, U +_\Delta \boldsymbol{q}\rangle. \tag{13}$$

This kind of interpolation also preserves enclosure.

Theorem 4 *Let $\mathcal{B} = \langle n, P, L, U \rangle$ be a bounding set, with P a grid. If $\mathcal{B}$ encloses a function f, then for every Delaunay triangulation Δ of P and every point $\boldsymbol{q} \in dom(\mathcal{B})$, $\mathcal{B} +_{\Delta} \boldsymbol{q}$ also encloses f.*

Appendix E. Implementation Details

We exploit the meta-programming features of the Julia programming language for expression parsing and symbolic analysis. We use the Symbolics.jl (Gowda et al., 2022) package for analytical derivatives, IntervalRootfinding.jl (JuliaIntervals, 2024) for sound root finding, and OVERT.jl (Sidrane et al., 2022) for univariate bound computation. We combine methods from OVERTVerify.jl and NeuralVerification.jl (Liu et al., 2020) for our local implementation of MIPVerify. We use the Plasmo framework (Jalving et al., 2019, 2022) to solve MILPs on graphs, with Gurobi as a backend solver (Gurobi Optimization, LLC, 2024).

Appendix F. Experimental Details

F.1. Experimental Setup

Computation times were obtained by running all tools on an AMD Ryzen 9 7950x processor. For the pendulum, ACC, and TORA benchmarks, we used concrete reachability. For the Unicycle benchmark, which is more challenging, we used five iterations of symbolic reachability with $k = 10$ (called *hybrid-symbolic reachability* in (Sidrane et al. (2022) for both OVERTPoly and OVERTVerify. For OVERTPoly, we also refined the nonlinear composition described above with a more precise method specialized for low dimensions based on McCormick envelopes (Hijazi (2019).

Appendix G. Proofs of Technical Claims

In this section, we provide detailed proofs for the lemmas and theorems introduced in the paper. Each claim is restated for clarity, followed by its corresponding proof.

Theorem 5 *If a bounding set $\mathcal{B} = \langle k, P, L, U \rangle$ encloses a function f, then every lifted bounding set encloses the corresponding lifted function. Formally, for every $n > k$, $S \subset [n]$ with $|S| = k$, and $\boldsymbol{p}^{\,l}, \boldsymbol{p}^{\,u} \in \mathbb{R}^n$, with $\boldsymbol{p}_i^l < \boldsymbol{p}_i^u$ for $i \in [n] \setminus S$, $\mathcal{B}{\uparrow}_{\boldsymbol{p}^{\,l}, \boldsymbol{p}^{\,u}}^{S,n}$ encloses $f{\uparrow}^{S,n}$.*

G.0.1. PROOF OF THEOREM 1

Proof Let $\mathcal{B}{\uparrow}_{\boldsymbol{p}^{\,l}, \boldsymbol{p}^{\,u}}^{S,n} = \langle n, P', L{\uparrow}^{S,n}, U{\uparrow}^{S,n} \rangle$, and let $\boldsymbol{y}$ be a point in $dom(\mathcal{B}{\uparrow}_{\boldsymbol{p}^{\,l}, \boldsymbol{p}^{\,u}}^{S,n})$.

We need to prove that for any Δ, $L_y \leq f{\uparrow}^{S,n}(\boldsymbol{y}) \leq U_y$. We consider the validity of the lower bound, as the reasoning for the upper bound is symmetric.

Suppose a Δ exists where $L_y \not\leq f{\uparrow}^{S,n}(\boldsymbol{y})$. Define $\boldsymbol{x}$ to be the restriction of $\boldsymbol{y}$ to $dom(\mathcal{B})$. A similar restriction of the previous inequality yields $L_{\boldsymbol{x}} \not\leq f(\boldsymbol{x})$. However, the theorem states that $\mathcal{B}$ encloses f, which is true iff $L_{\boldsymbol{x}} \leq f(\boldsymbol{x})$. This presents a contradiction. ■

Lemma 1 *Let G be an n-dimensional grid, and let Δ be a Delaunay triangulation of G. Then, for every n-simplex $\mathcal{S} \in \Delta$, $C(\mathcal{S})$ is also the circumsphere of a grid cell of G.*

G.0.2. PROOF OF LEMMA 1

Proof This is a straightforward consequence of the duality between Delaunay triangulations and Voronoi Diagrams. Notice that in a grid, the Voronoi cells are hyperrectangles whose vertices are the centers of the grid cells. Thus, every circumsphere of an n-simplex in the Delaunay triangulation must have its center at the center of a grid cell. Since the circumsphere cannot be larger than the circumsphere of the grid cell by the Delaunay condition and cannot be smaller as it would then touch no points in the grid, it must be exactly the circumsphere of the grid cell. ■

Theorem 6 *Let $\mathcal{B} = \langle n, P, L, U\rangle$ be a bounding set, with P a grid. If $\mathcal{B}$ encloses a function f, then for every Delaunay triangulation Δ of P and every point $\boldsymbol{q} \in dom(\mathcal{B})$, $\mathcal{B} +_{\Delta} \boldsymbol{q}$ also encloses f.*

G.1. Proof of Theorem 2

Proof Let $\boldsymbol{q}$ be an arbitrary point in $dom(P)$, let $\mathcal{B}' = \mathcal{B} +_{\Delta} \boldsymbol{q} = \mathcal{B}' = \langle n, P', L', U'\rangle$, and let Δ' be a Delaunay triangulation of P'. To show that $\mathcal{B}'$ encloses f, we must show that $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}', \Delta')$ for every $\boldsymbol{x} \in dom(\mathcal{B}')$. let $\mathcal{S} = \mathcal{S}_{\Delta'}(\boldsymbol{x})$, $V_s = \mathbf{vert}(\mathcal{S})$, and $\theta = \theta_{\Delta'}(\boldsymbol{x})$.

Let G' be the grid cell of P' whose circumsphere coincides with that of $\mathcal{S}$ (see Lemma 1). Let $L_{\boldsymbol{x}} = \theta \cdot L'(V_s)$, and $U_{\boldsymbol{x}} = \theta \cdot U'(V_s)$. We consider two cases.

First, if G' is also a grid cell of P, then the simplex $\mathcal{S}$ must also occur in a Delaunay triangulation of P. Recall that $\mathcal{B}_{\mathcal{S}}$ is the restriction of the bounding set $\mathcal{B}$ to the simplex $\mathcal{S}$, obtained by restricting P to only the points in V_s. The restriction of $\mathcal{B}'$ to $\mathcal{S}$ is going to be just the same as the restriction of $\mathcal{B}$ to $\mathcal{S}$, that is, $\mathcal{B}_{\mathcal{S}} = \mathcal{B}'_{\mathcal{S}}$. So, by substitution, $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}_{\mathcal{S}})$ iff $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}'_{\mathcal{S}})$. But we know that $\mathcal{B}$ encloses f, so we must have $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{B}_{\mathcal{S}}$. Therefore, $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{B}'_{\mathcal{S}}$, and so $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}', \Delta')$.

The more interesting case is when G' is not a grid cell of P. Since P' is a refined version of P, we know that $G' \subseteq \mathbf{conv}(G)$ for some grid cell G of P. For notational convenience, let $I_{\boldsymbol{q}}$ be the set of points inserted into P to obtain P'.

We wish to show that $L_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \leq U_{\boldsymbol{x}}$. We include only the proof for the upper bound since the proof for the lower bound is symmetric. Let $\mathcal{P}_G = \mathbf{conv}(U(G))$ be the polyhedron enclosed by the points in $U(G)$.

Recall that the point $U_{\boldsymbol{x}}$ is defined as $U_{\boldsymbol{x}} = \theta \cdot U'(V_s)$. We aim to show that it is also a convex combination of $U(G)$, and thus in $\mathcal{P}_G$. First, note that for each $\boldsymbol{v} \in V_s \cap G$, we have that $U'(\boldsymbol{v}) = U(\boldsymbol{v}) \in U(G)$.

On the other hand, consider $\boldsymbol{v} \in V_s \setminus G$, meaning $\boldsymbol{v} \in I_{\boldsymbol{q}}$. In this case, we need to look at the simplex containing $\boldsymbol{v}$ in Δ, the triangulation used to interpolate U to get U'. Let $\hat{\mathcal{S}} = \mathcal{S}_{\Delta}(\boldsymbol{v})$, and let $V = \mathbf{vert}(\hat{\mathcal{S}})$ be its vertices. We know, by definition of interpolation, that $U'(\boldsymbol{v}) = \theta_{\Delta}(\boldsymbol{v}) \cdot U(V)$. Now, if $\boldsymbol{v}$ is in the interior of $\mathbf{conv}(G)$, then we must have $V \subseteq G$. To see why, recall that (by Lemma 1) the circumsphere of $\hat{\mathcal{S}}$ must also be the circumsphere of some grid cell of P. If this grid cell is something other than G, then $\boldsymbol{v}$ would not be expressible as a convex combination of the points in the grid cell, of which V is a subset, meaning that $\hat{\mathcal{S}}$ must have the same circumsphere as G, and so $V \subseteq G$, and thus $U(V) \subseteq U(G)$

Finally, suppose that $\boldsymbol{v}$ is on a face of $\mathbf{conv}(G)$. In that case, we know that $\theta_{\Delta}(\boldsymbol{v})$ is non-zero only for dimensions corresponding to vertices of that face. Thus, the equation $U'(\boldsymbol{v}) = \theta_{\Delta}(\boldsymbol{v}) \cdot U(V)$ can be rewritten as $U'(\boldsymbol{v}) = \theta' \cdot U(V_F)$ for some convex combination vector θ' and for $V_F \subseteq G$.

Thus, for each $\boldsymbol{v} \in V_s$, we have that $U'(\boldsymbol{v})$ is a convex combination of points in $U(G)$. Thus, since $U_{\boldsymbol{x}}$ is a convex combination of points in $U'(V_s)$, we have that $U_{\boldsymbol{x}}$ is a convex combination of points in $U(G)$, meaning $U(\boldsymbol{x}) \in \mathcal{P}_G$.

We now claim that $f(\boldsymbol{x}) \leq U^G_{\boldsymbol{x}}$, where $U^G_{\boldsymbol{x}}$ is the point on the lower face of $\mathcal{P}_G$ at $\boldsymbol{x}$. To see this, note that the face must contain $U(\mathcal{S}')$ for some n-simplex $\mathcal{S}'$ containing $\boldsymbol{x}$ and whose vertices are a subset of G. $\mathcal{S}'$ satisfies the Delaunay condition, since its circumsphere is the same as that of G. It must therefore also be part of some Delaunay triangulation of P, and thus, since $\mathcal{B}$ encloses f, we have that $f(\boldsymbol{x}) \in \mathcal{B}_{\mathcal{S}'}$, and so $f(\boldsymbol{x}) \leq U^G_{\boldsymbol{x}} \leq U_{\boldsymbol{x}}$. ■

Theorem 7 *Let $\mathcal{B}^f = \langle n, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n, P, L^g, U^g \rangle$, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. Then, for $\bowtie \in \{+, -\}$, $\mathcal{B}^f \bowtie \mathcal{B}^g$ encloses $f \bowtie g$.*

G.1.1. PROOF OF THEOREM 3

Proof Let $\mathcal{B}^{fg} = \mathcal{B}^f \bowtie \mathcal{B}^g$, let $x \in dom(\mathcal{B}^{fg})$, and let Δ be a Delaunay triangulation of P. Let $\mathcal{S} = \mathcal{S}_\Delta(\boldsymbol{x})$. We must show that $(\boldsymbol{x}, f(\boldsymbol{x}) \bowtie g(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}^{fg}_S, \Delta)$.

Let $V = \mathbf{vert}(\mathcal{S})$ and $\boldsymbol{\theta} = \theta_\Delta(\boldsymbol{x})$. Let $L^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^{fg}(V)$ and $U^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^{fg}(V)$. Showing $(\boldsymbol{x}, f(\boldsymbol{x}) \bowtie g(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}^{fg}_S, \Delta)$ reduces to showing $L^{fg}_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \bowtie g(\boldsymbol{x}) \leq U^{fg}_{\boldsymbol{x}}$.

Let $L^f_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^f(V)$, $U^f_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^f(V)$, $L^g_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^g(V)$, and $U^g_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^g(V)$. We know that $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}^f_S)$ and $(\boldsymbol{x}, g(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}^g_S)$, so we have $L^f_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \leq U^f_{\boldsymbol{x}}$ and $L^g_{\boldsymbol{x}} \leq g(\boldsymbol{x}) \leq U^g_{\boldsymbol{x}}$.

Now, if $\bowtie = +$, we have $L^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^{fg}(V) = \boldsymbol{\theta} \cdot (L^f(V) + L^g(V)) = \boldsymbol{\theta} \cdot L^f(V) + \boldsymbol{\theta} \cdot L^g(V) = L^f_{\boldsymbol{x}} + L^g_{\boldsymbol{x}} \leq f(\boldsymbol{x}) + g(\boldsymbol{x})$. We can use a similar argument to show that $f(\boldsymbol{x}) + g(\boldsymbol{x}) \leq U^{fg}_{\boldsymbol{x}}$.

If $\bowtie = -$, we have $L^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^{fg}(V) = \boldsymbol{\theta} \cdot (L^f(V) - U^g(V)) = \boldsymbol{\theta} \cdot L^f(V) - \boldsymbol{\theta} \cdot U^g(V) = L^f_{\boldsymbol{x}} - U^g_{\boldsymbol{x}} \leq f(\boldsymbol{x}) - g(\boldsymbol{x})$. We can use a similar argument to show that $f(\boldsymbol{x}) - g(x) \leq U^{fg}_{\boldsymbol{x}}$. ■

Theorem 8 *Let $\mathcal{B}^f = \langle n, P, L^f, U^f \rangle$ and $\mathcal{B}^g = \langle n, P, L^g, U^g \rangle$, and suppose that $\mathcal{B}^f$ encloses f and $\mathcal{B}^g$ encloses g. Then, $\mathcal{B}^f \times \mathcal{B}^g$ encloses $f \times g$. Furthermore, if for every Delaunay triangulation Δ of P, $\mathcal{E}(\mathcal{B}^g, \Delta) \cap (z(\boldsymbol{x}) = 0) = \emptyset$ (i.e., the enclosure for g does not intersect the plane corresponding to the constant zero function), then $\mathcal{B}^f \div \mathcal{B}^g$ encloses $f \div g$.*

G.1.2. PROOF OF THEOREM 4

Proof For $\bowtie \in \{\times, \div\}$, let $\mathcal{B}^{fg} = \mathcal{B}^f \bowtie \mathcal{B}^g$. Also, let $x \in dom(\mathcal{B}^{fg})$, and let Δ be a Delaunay triangulation of P. Let $\mathcal{S} = \mathcal{S}_\Delta(\boldsymbol{x})$. We must show that $(\boldsymbol{x}, f(\boldsymbol{x}) \bowtie g(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}^{fg}_S, \Delta)$.

Let $V = \mathbf{vert}(\mathcal{S})$ and $\boldsymbol{\theta} = \theta_\Delta(\boldsymbol{x})$. Let $L^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^{fg}(V)$ and $U^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^{fg}(V)$. Showing $(\boldsymbol{x}, f(\boldsymbol{x}) \bowtie g(\boldsymbol{x})) \in \mathcal{E}(\mathcal{B}^{fg}_S, \Delta)$ reduces to showing $L^{fg}_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \bowtie g(\boldsymbol{x}) \leq U^{fg}_{\boldsymbol{x}}$.

Let $L^f_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^f(V)$, $U^f_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^f(V)$, $L^g_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^g(V)$, and $U^g_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot U^g(V)$. We know that $(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}^f_S)$ and $(\boldsymbol{x}, g(\boldsymbol{x})) \in \mathcal{P}(\mathcal{B}^g_S)$, so we have $L^f_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \leq U^f_{\boldsymbol{x}}$ and $L^g_{\boldsymbol{x}} \leq g(\boldsymbol{x}) \leq U^g_{\boldsymbol{x}}$.

Now, we know by Lemma 1 that $V \subseteq \mathcal{X}$ for some grid cell $\mathcal{X}$. Thus, since $L^f_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^f(V)$, and $L^f(\boldsymbol{v}) \geq L^f_{\mathcal{X}}$ for each $\boldsymbol{v} \in V$, it follows that $L^f_{\boldsymbol{x}} \geq L^f_{\mathcal{X}}$, and so $L^f_{\mathcal{X}} \leq f(\boldsymbol{x})$. A similar argument shows that $f(\boldsymbol{x}) \leq U^f_{\mathcal{X}}$. And the same analysis for g yields $L^g_{\mathcal{X}} \leq g(\boldsymbol{x}) \leq U^g_{\mathcal{X}}$. It then follows from interval arithmetic that $L_{\bowtie}(\mathcal{X}) \leq f(\boldsymbol{x}) \bowtie g(\boldsymbol{x}) \leq U_{\bowtie}(\mathcal{X})$.

Finally, since $L^{fg}_{\boldsymbol{x}} = \boldsymbol{\theta} \cdot L^{fg}(V)$ and $L^{fg}(\boldsymbol{v}) \leq L_{\bowtie}(\mathcal{X})$ for each $\boldsymbol{v} \in V$, it follows that $L^{fg}_{\boldsymbol{x}} \leq L_{\bowtie}(\mathcal{X})$. By similar reasoning, $U_{\bowtie}(\mathcal{X}) \leq U^{fg}_{\boldsymbol{x}}$. It follows that $L^{fg}_{\boldsymbol{x}} \leq f(\boldsymbol{x}) \bowtie g(\boldsymbol{x}) \leq U^{fg}_{\boldsymbol{x}}$.

■

Appendix H. Benchmark Descriptions

In this section, we describe the ARCH Competition benchmarks used to evaluate our tool. If S is a set, we use S^c to define the complement of S.

H.0.1. SINGLE PENDULUM

We would like to verify that the angle of an inverted pendulum remains within a specified range during a specified time interval. Formally, we define the neural feedback system $\mathcal{P}$, where $n^{\mathcal{P}} = 2$, $I^{\mathcal{P}} = [1.0, 1.2] \times [0.0, 0.2]$, $F^{\mathcal{P}}(\boldsymbol{x}) = (\boldsymbol{x}_2, c_1\boldsymbol{x}_1 - c_2\boldsymbol{x}_2)$,$E^{\mathcal{P}} = \{0\} \times \{0\}$, $u^{\mathcal{P}}$ is computed by a neural network with two hidden layers (each with 25 neurons), and two outputs, the first of which is set to the constant zero value, $\delta^{\mathcal{P}} = 0.05$, $T^{\mathcal{P}} = 20$, and $G^{\mathcal{P}} = \emptyset$. Let R be the set defined by $[0, 1] \times [-\infty, \infty]$, then $A^{\mathcal{P}}$ is the sequence of sets defined by

$$A^{\mathcal{P}}(t) = \begin{cases} \emptyset, & \text{if } t < 10 \\ R^c & \text{if } 10 \leq t \leq 20 \end{cases}$$

We would like to verify that the avoid property holds for $\mathcal{P}$.

H.0.2. ACC

In this benchmark, we would like to verify that a neural network controlled vehicle tracks a set velocity while maintaining a safe distance from a second vehicle. Formally, we define the neural feedback system $\mathcal{A}$, where $n^{\mathcal{A}} = 6$, $I^{\mathcal{A}} = [90, 110] \times [32, 32.2] \times \{0\} \times [10, 11] \times [30, 30.2] \times \{0\}$, $F^{\mathcal{A}}(\boldsymbol{x}) = (\boldsymbol{x}_2, \boldsymbol{x}_3, -2\boldsymbol{x}_3 - c_1\boldsymbol{x}_2^2, \boldsymbol{x}_5, \boldsymbol{x}_6, -2\boldsymbol{x}_6 - c_1\boldsymbol{x}_5^2)$,$E^{\mathcal{A}} = \{0\} \times \{0\} \times \{0\} \times \{0\}$, $u^{\mathcal{A}}$ is computed by a neural network with five hidden layers (each with 20 neurons), and six outputs, of which outputs 1,2,4, and 5 are set to the constant zero value, and output 3 is set to the constant -4 value, $\delta^{\mathcal{A}} = 0.1$, $T^{\mathcal{A}} = 50$, and $G^{\mathcal{A}} = \emptyset$. Let R be the set of states satisfying $\boldsymbol{x}_1 - \boldsymbol{x}_4 \geq 10 + 1.4 \cdot \boldsymbol{x}_5$ then $A^{\mathcal{A}}$ is the sequence of sets defined by

$$A^{\mathcal{A}}(t) = \begin{cases} R^c, & \text{if } 0 \leq t \leq 50 \\ \emptyset & \text{if } t > 50 \end{cases}$$

We would like to verify that the avoid property holds for $\mathcal{A}$.

H.0.3. TORA

We would like to verify that the states of an actuated cart remain within a safe region during a specified time interval. Formally, we define the neural feedback system $\mathcal{T}$, where $n^{\mathcal{T}} = 4$, $I^{\mathcal{T}} = [0.6, 0.7] \times [-0.7, -0.6] \times [-0.4, -0.3] \times [0.5, 0.6]$, $F^{\mathcal{T}}(\boldsymbol{x}) = (\boldsymbol{x}_2, -\boldsymbol{x}_1 + 0.1\sin(\boldsymbol{x}_3), \boldsymbol{x}_4, 0)$,$E^{\mathcal{T}} = \{0\} \times \{0\} \times \{0\} \times \{0\}$, $u^{\mathcal{T}}$ is computed by a neural network with three hidden layers (each with 100 neurons), and four outputs, the first three of which is set to the constant zero value, $\delta^{\mathcal{T}} = 0.1$,

$T^{\mathcal{T}} = 20$, and $G^{\mathcal{T}} = \emptyset$. Let R be the set defined by $[-2, 2] \times [-2, 2] \times [-2, 2] \times [-2, 2]$, then $A^{\mathcal{P}}$ is the sequence of sets defined by

$$A^{\mathcal{P}}(t) = \begin{cases} R^c, & \text{if } 0 \leq t \leq 20 \\ \emptyset & \text{if } t > 20 \end{cases}$$

We would like to verify that the avoid property holds for $\mathcal{T}$.

H.0.4. UNICYCLE CAR MODEL

See running example.