POLAR: A Polynomial Arithmetic Framework for Verifying Neural-Network Controlled Systems

Chao Huang[1], Jiameng Fan[2], Xin Chen[3], Wenchao Li[2], and Qi Zhu[4]

[1] University of Liverpool, `chao.huang2@liverpool.ac.uk`
[2] Boston University, {`jmfan, wenchao`}`@bu.edu`
[3] University of Dayton, `xchen4@udayton.edu`
[4] Northwestern University, `qzhu@northwestern.edu`

Abstract. We present POLAR[5], a **pol**ynomial **ar**ithmetic-based framework for efficient bounded-time reachability analysis of neural-network controlled systems (NNCSs). Existing approaches that leverage the standard Taylor Model (TM) arithmetic for approximating the neural-network controller cannot deal with non-differentiable activation functions and suffer from rapid explosion of the remainder when propagating the TMs. POLAR overcomes these shortcomings by integrating TM arithmetic with **Bernstein polynomial interpolation** and **symbolic remainders**. The former enables TM propagation across non-differentiable activation functions and local refinement of TMs, and the latter reduces error accumulation in the TM remainder for linear mappings in the network. Experimental results show that POLAR significantly outperforms the current state-of-the-art tools in terms of both efficiency and tightness of the reachable set overapproximation.

1 Introduction

Neural networks have been increasingly used as the central decision makers in a variety of control tasks [23, 25, 18]. However, the use of neural-network controllers also gives rise to new challenges on verifying the correctness of the resulting closed-loop control systems especially in safety-critical settings. In this paper, we consider the reachability verification problem of neural-network controlled systems (NNCSs). The high-level architecture of a simple NNCS is shown in Figure 1 in which the neural network senses the system state, i.e. the value of $\vec{x}$, at discrete time steps, and computes the corresponding control values $\vec{u}$ for updating the system dynamics which is defined by an ordinary differential equation (ODE) over $\vec{x}$ and $\vec{u}$. The *bounded-time reachability analysis problem* of an NNCS is to compute an (overapproximated) reachable set that contains all the trajectories starting from an initial set for a finite number of control steps. The initial set can represent uncertainties in the starting state of the system or error (e.g. localization error) bounds in estimating the current system state

[5] The source code can be found in `https://github.com/ChaoHuang2018/POLAR_Tool`.

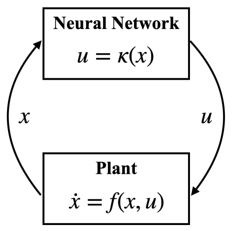


Fig. 1: A typical NNCS model.

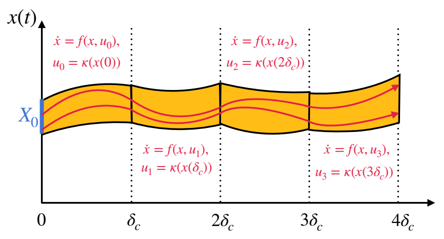


Fig. 2: Executions over 4 control steps.

during an execution of the system. Figure 2 shows an illustration of reachable sets for 4 steps, where the orange region represents the reachable set, and the two red, arrowed curves are two example trajectories starting from two different initial states in the initial set X_0 (blue).

Reachability analysis of general NNCSs is notoriously difficult due to nonlinearity in both the neural-network controller and the plant. The difficulty is further exacerbated by the coupling of the controller and the plant over multiple control steps. Since exact reachability of general nonlinear systems is undecidable [2], current approaches for reachability analysis of nonlinear dynamical systems largely focus on computing a tight overapproximation of the reachable sets [20, 26, 10, 6, 1]. Verisig [15] leverages properties of the sigmoid activation function and converts an NNCS with these activation functions to an equivalent hybrid system. Thus, existing tools for hybrid system reachability analysis can be directly applied to solve the NNCS reachability problem. However, this approach inherits the efficiency problem of hybrid system reachability analysis and does not scale beyond very small NNCSs. Another line of approach is to draw on techniques for computing the output ranges of neural networks [12, 17, 30–32, 28] by directly integrating them with reachability analysis tools designed for dynamical systems. NNV [29], for instance, combines star set analysis on the neural network with zonotope-based analysis of the nonlinear plant dynamics from CORA [1]. However, this type of approach has been shown to be ineffective for NNCS verification due to the lack of consideration on the interaction between the neural-network controller and the plant dynamics [8, 11, 13]. In particular, since the primary goal of these techniques is to bound the output range of the neural network instead of approximating its input-output function, they *cannot track state dependencies across the closed-loop system and across multiple time steps in reachability analysis.*

More recent advances in NNCS reachability analysis are based on the idea of *function overapproximation* of the neural network controller. A function overapproximation of a neural network κ has two components: an approximated function p and an error term I (e.g. an interval) that bounds the approximation error. Such function overapproximation that produces a *point-wise* approximation of κ with an interval error term (typically called a remainder) is also known as a *Taylor model* (TM). Function-overapproximation approaches can be broadly categorized into two classes: *direct end-to-end approximation* such as Sherlock [8], ReachNN [11] and ReachNN* [9], and *layer-by-layer propagation* such as Verisig 2.0 [14, 13]. The former computes a function overapproximation of the neural network end-to-end by sampling from the input space. The main drawback of this approach is that it does not scale beyond systems with more than a few input dimensions. The latter approach tries to exploit the neural network structure and uses *Taylor model arithmetic* to more efficiently obtain a function overapproximation of κ by propagating the TMs layer by layer through the network (details in Section 3). However, due to limitations of basic TM arithmetic, these approaches *cannot handle non-differentiable activation functions and suffer from rapid growth of the remainder* during propagation. For instance, explosion of the interval remainder would degrade a TM propagation to an interval analysis.

In this paper, we propose a principled **pol**ynomial **ar**ithmetic framework (POLAR) that enables precise layer-by-layer propagation of TMs for general feed-forward neural networks. Basic Taylor model arithmetic cannot handle ReLU that is nondifferentiable (cannot produce the polynomial), and also suffers from low approximation precision (large remainder). POLAR addresses the key challenges of applying basic TM arithmetic through a novel use of *univariate Bernstein polynomial interpolation* and *symbolic remainders*. Univariate Bernstein polynomial interpolation enables the handling of non-differentiable activation functions and local refinement of Taylor models (details in Section 3.1). Symbolic remainders can taper the growth of interval remainders by avoiding the so-called wrapping effect [16] in linear mappings. The paper has the following novel contributions: (I) A polynomial arithmetic framework using both Taylor and univariate Bernstein approximations for computing NNCS reachable sets to handle general NN controllers; (II) An adaptation of the symbolic remainder method for ODEs to the layer-by-layer propagation for neural networks; (III) A comprehensive experimental evaluation of our approach on challenging case studies that demonstrates significant improvements of POLAR against SOTA.

2 Preliminaries

A *Neural-Network Controlled System (NNCS)* is a continuous plant governed by a neural network controller. The plant dynamics is defined by an ODE of the form $\dot{\vec{x}} = f(\vec{x}, \vec{u})$ wherein the state variables and control inputs are denoted by the vectors $\vec{x}$ and $\vec{u}$ respectively. We assume that the function f is at least locally Lipschitz continuous such that its solution w.r.t. an initial state and constant control inputs is unique [22]. We denote the input-output mapping of the neural network controller as κ. The controller is triggered every δ_c time which is also called the *control stepsize*. A system *execution (trajectory)* is produced in the following way: starting from an initial state $\vec{x}(0)$, the controller senses the system state at the beginning of every control step $t{=}j\delta_c$ for $j{=}0, 1, \ldots$, and updates the control inputs to $\vec{v}_j{=}\kappa(\vec{x}(j\delta_c))$. The system's dynamics in that control step is governed by the ODE $\dot{\vec{x}}{=}f(\vec{x}, \vec{v}_j)$.

Given an initial state set $X_0 \subset \mathbb{R}^n$, all executions from a state in this set can be formally defined by a *flowmap* function $\varphi_{\mathcal{N}} : X_0 \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, such that the system state at any time $t \geq 0$ from any initial state $\vec{x}_0 \in X_0$ is $\varphi_{\mathcal{N}}(\vec{x}_0, t)$. We call a state $\vec{x}' \in \mathbb{R}^n$ *reachable* if there exists $\vec{x}_0 \in X_0$ and $t \geq 0$ such that $\vec{x}' = \varphi_{\mathcal{N}}(\vec{x}_0, t)$. The *reachability problem* on NNCS is to decide whether a state is reachable in a given NNCS, and it is *undecidable* since NNCS is more expressive than two-counter machines for which the reachability problem is already undecidable [2]. Many formal verification problems can be reduced to the reachability problem. For example, the safety verification problem can be reduced to checking reachability to an unsafe state. In the paper, we focus on computing the reachable set for an NNCS over a bounded number K of control steps. Since flowmap $\varphi_{\mathcal{N}}$ often does not have a closed form due to the nonlinear ODEs, we seek to compute *state-wise overapproximations* for it over multiple time segments, that is, in each control step $[j\delta_c, (j+1)\delta_c]$ for $j = 0, \ldots, K-1$, the reachable set is overapproximated by a group of flowpipes $\mathcal{F}_1(\vec{x}_0, \tau), \ldots, \mathcal{F}_N(\vec{x}_0, \tau)$ over the N uniformly subdivided time segments of the time interval, such that $\mathcal{F}_i(\vec{x}_0, \tau)$ is a *state-wise overapproximation* of $\varphi_{\mathcal{N}}(\vec{x}_0, j\delta_c + (i-1)\delta + \tau)$ for $\tau \in [0, \delta_c/N]$, i.e., $\mathcal{F}_j(\vec{x}_0, \tau)$ contains the exact reachable state from any initial state $\vec{x}_0$ in the i-th time segment of the j-th control step. Here, τ is the local time variable which is independent in each flowpipe. A high-level flowpipe construction algorithm is presented as follows, in which $\hat{X}_0 = X_0$ and $\delta = \delta_c/N$ is called the *time step*.

```
1: for $j = 0$ to $K-1$ do
2:     Computing an overapproximation $\hat{U}_j$ for the control input range $\kappa(\hat{X}_j)$;
3:     Computing the flowpipes $\mathcal{F}_1(\vec{x}_0, \tau), \ldots, \mathcal{F}_N(\vec{x}_0, \tau)$ for the continuous dy-
       namics $\dot{\vec{x}} = f(\vec{x}, \vec{u}), \dot{\vec{u}} = 0$ from the initial set $\vec{x}(0) \in \hat{X}_j$, $\vec{u}(0) \in \hat{U}_j$;
4:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathcal{F}_1(\vec{x}_0, \tau), \ldots, \mathcal{F}_N(\vec{x}_0, \tau)\}$;
5:     $\hat{X}_{j+1} \leftarrow \mathcal{F}_N(\vec{z}, \delta)$;
6: end for
```

Notice that $\vec{x}(0)$ denotes the local initial set for the ODE used in the current control step, that is the system reachable set at the time $j\delta_c$, while the variables $\vec{x}_0$ in a flowpipe are the symbolic representation of an initial state in X_0. Intuitively, a flowpipe overapproximates not only the reachable set in a time step, but also the *dependency* from an initial state to its reachable state at a particular time. For settings where the plant dynamics of an NNCS is given as a difference equation in the form of $\vec{x}_{k+1} = f(\vec{x}_k, \vec{u}_k)$, we can obtain *discrete* flowpipes which are the reachable set overapproximations at discrete time points by repeatedly computing the state set at the next step using TM arithmetic.

Dependencies on the initial set. As we mentioned previously, the reachable state of an NNCS at a time $t > 0$ is *uniquely determined* by its initial state if there is no noise or disturbance in the system dynamics or on the state measurements. If we use X_j to denote the exact reachable set $\{\varphi_{\mathcal{N}}(\vec{x}_0, j\delta_c) \mid \vec{x}_0 \in X_0\}$ from a given initial set X_0, then the control input range is defined by the set $U_j = \{\kappa(\vec{x}_j) \mid \vec{x}_j = \varphi_{\mathcal{N}}(\vec{x}_0, j\delta_c) \text{ and } \vec{x}_0 \in X_0\}$. More intuitively, the set U_j is the image from the initial set X_0 under the mapping $\kappa(\varphi_{\mathcal{N}}(\cdot, j\delta_c))$. *The main challenge in computing NNCS reachable sets is to control the overapproximation, which requires accurately tracking the dependency of a reachable set on the initial set* *across multiple control steps.* In this paper, we present a polynomial arithmetic framework for tracking such dependencies using Taylor models.

Taylor model arithmetic. Taylor models are originally proposed to compute higher-order overapproximations for the ranges of continuous functions (see [4]). They can be viewed as a higher-order extension of intervals [24], which are sets of real numbers between lower and upper real bounds, e.g., the interval $[a, b]$ wherein $a \leq b$ represents the set of $\{x \,|\, a \leq x \leq b\}$. A *Taylor model (TM)* is a pair (p, I) wherein p is a polynomial of degree k over a finite group of variables $x_1, \ldots, x_n$ ranging in an interval domain $D \subset \mathbb{R}^n$, and I is the remainder interval. The range of a TM is the Minkowski sum of the range of its polynomial and the remainder interval. Thereby we sometimes intuitively denote a TM (p, I) by $p + I$ in the paper. TMs are closed under operations such as addition, multiplication, and integration (see [21]). Given functions f, g that are overapproximated by TMs (p_f, I_f) and (p_g, I_g), respectively, a TM for $f + g$ can be computed as $(p_f + p_g, I_f + I_g)$, and an order k TM for $f \cdot g$ can be computed as $(\, p_f \cdot p_g - r_k\, ,\, I_f \cdot B(p_g) + B(p_f) \cdot I_g + I_f \cdot I_g + B(r_k)\,)$, wherein $B(p)$ denotes an interval enclosure of the range of p, and the *truncated part* r_k consists of the terms in $p_f \cdot p_g$ of degrees $> k$. Similar to reals and intervals, TMs can also be organized as vectors and matrices to overapproximate the functions whose ranges are multidimensional. Notice that *a TM is a function overapproximation and not just a range overapproximation like intervals or polyhedra.*

3 Framework of POLAR

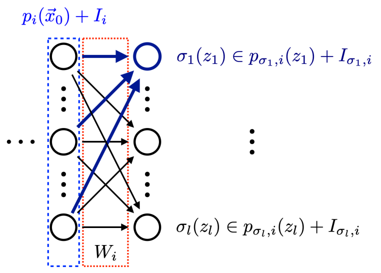
In this section, we describe POLAR's approach for computing a TM for the output range of a neural network (NN) when the input range is defined by a TM. POLAR uses the layer-by-layer propagation strategy, and features the following key novelties: (a) A method to compute univariate Bernstein Polynomial **(BP)** overapproximations for activation functions, and selectively uses Taylor or Bernstein polynomials to *limit the overestimation produced when overapproximating the output ranges of individual neurons.* (b) A technique to symbolically represent the intermediate linear transformations of TM interval remainders during the layer-by-layer propagation. The purpose of using Symbolic Remainders **(SR)** is to *reduce the accumulation of overestimation in composing a sequence of TMs.*

3.1 Main Framework

We begin by introducing POLAR's propagation framework that incorporates only (a), and then describe how to extend it by further integrating (b). Although using TMs to represent sets in layer-by-layer propagation is already used in [14, 13], the method only computes Taylor approximations for activation functions, and the TM output of one layer is propagated by the existing arithmetic for TM composition to the next layer. Such a method has the following shortcomings: (1) the activation functions have to be differentiable, (2) standard TM composition is often the source of overestimation even preconditioning and shrink wrapping are used. Here, we seek to improve the use of TMs in the above two aspects.

Algorithm 1 Layer-by-layer propagation using polynomial arithmetic and TMs

```
Input: Input TM $(p_1(\vec{x}_0), I_1)$ with $\vec{x}_0 \in X_0$, the $M+1$ matrices $W_1, \ldots, W_{M+1}$ of
    the weights on the incoming edges of the hidden and the output layers, the $M+1$
    vectors $B_1, \ldots, B_{M+1}$ of the neurons' bias in the hidden and the output layers, the
    $M+1$ activation functions $\sigma_1, \ldots, \sigma_{M+1}$ of hidden and output layers.
Output: a TM $(p_r(\vec{x}_0), I_r)$ that contains the set $\kappa((p_1(\vec{x}_0), I_1))$.
1: $(p_r, I_r) \leftarrow (p_1, I_1)$;
2: for $i = 1$ to $M+1$ do
3:     $(p_t, I_t) \leftarrow W_i \cdot (p_r, I_r) + B_i$;                       # Using TM arithmetic
4:     Computing a polynomial approximation $p_{\sigma,i}$ for $\sigma$ w.r.t. the domain $(p_t, I_t)$;
5:     Evaluating a conservative remainder $I_{\sigma,i}$ for $p_{\sigma,i}$ w.r.t. the domain $(p_t, I_t)$;
6:     $(p_r, I_r) \leftarrow p_{\sigma,i}(p_t + I_t) + I_{\sigma,i}$;                # Using TM arithmetic
7: end for
8: return $(p_r, I_r)$.
```



$p_i(\vec{x}_0) + I_i$
$\sigma_1(z_1) \in p_{\sigma_1,i}(z_1) + I_{\sigma_1,i}$
$\sigma_l(z_l) \in p_{\sigma_l,i}(z_l) + I_{\sigma_l,i}$
W_i

Fig. 3: Single layer propagation

Before presenting our layer-by-layer propagation method, we describe how a TM output is computed from a given TM input for a single layer. The idea is illustrated in Fig. 3. The circles in the right column denote the neurons in the current layer which is the i-th layer, and those in the left column denotes the neurons in the previous layer. The weights on the incoming edges to the current layer is organized as a matrix W_i, while we use B_i to denote the vector organization of the biases in the current layer. Given that the output range of the neurons in the previous layer is represented as a TM (vector) $(p_i(\vec{x}_0), I_i)$ wherein $\vec{x}_0$ are the variables ranging in the NNCS initial set. Then, the output TM $(p_{i+1}(\vec{x}_0), I_{i+1})$ of the current layer can be obtained as follows. First, we compute the polynomial approximations $p_{\sigma_1,i}, \ldots, p_{\sigma_l,i}$ for the activation functions $\sigma_1, \ldots, \sigma_l$ of the neurons in the current layer. Second, interval remainders $I_{\sigma_1,i}, \ldots, I_{\sigma_l,i}$ are evaluated for those polynomials to ensure that for each $j = 1, \ldots, l$, $(p_{\sigma_j,i}, I_{\sigma_j,i})$ is a TM of the activation function σ_j w.r.t. z_j ranging in the j-th dimension of the set $W_i(p_i(\vec{x}_0) + I_i)$. Third, $(p_{i+1}(\vec{x}_0, I_{i+1}))$ is computed as the TM composition $p_{\sigma,i}(W_i(p_i(\vec{x}_0) + I_i) + I_{\sigma,i}$ wherein $p_{\sigma,i}(\vec{z}) = (p_{\sigma_1,i}(z_1), \ldots, p_{\sigma_l,i}(z_k))^T$ and $I_{\sigma,i} = (I_{\sigma_1,i}, \ldots, I_{\sigma_l,i})^T$. Hence, when there are multiple layers, starting from the first layer, the output TM of a layer is treated as the input TM of the next layer, and the final output TM is computed by composing TMs layer-by-layer.

We give the whole procedure by Algorithm 1. In our approach, the polynomial approximation $p_{\sigma,i}$ and its remainder interval $I_{\sigma,i}$ for the vector of activation functions σ in the i-th layer can be computed in the following two ways.

Taylor approximation. When the activation function is differentiable in the range defined by (p_t, I_t). The polynomial $p_{\sigma,i}$ can be computed as the order

k Taylor expansion of σ (in each of its dimension) at the center of (p_t, I_t), and the remainder is evaluated using interval arithmetic based on the Lagrange remainder form. More details are described elsewhere [21].

Bernstein interpolation. The use of Bernstein approximation only requires the activation function to be continuous in (p_t, I_t), and can be used not only in more general situations, but also to obtain better polynomial approximations than Taylor expansions (see [19]). Intuitively, an order k Taylor approximation can only guarantee to have the same value as the approximated function at the expansion point, however, an order k Bernstein interpolation has the same value as the approximated function at $k+1$ points. We give the details of our Bernstein overapproximation method as follows.

Bernstein approximation for $\sigma(\vec{z})$ w.r.t. $\vec{z} \in (p_t, I_t)$. Given (p_t, I_t) computed in Line 3, the j-th component of the polynomial vector $p_{\sigma,i}$ is the order k Bernstein interpolation of the activation function σ_j of the j-th neuron. It can be computed as $p_{\sigma_j,i}(z_j) = \sum_{s=0}^{k} \left(\sigma_j(\frac{\bar{Z}_j - \underline{Z}_j}{k} s + \underline{Z}_j) \binom{k}{s} \frac{(Z_j - \underline{Z}_j)^s (\bar{Z}_j - z_j)^{k-s}}{(\bar{Z}_j - \underline{Z}_j)^k} \right)$, such that $\bar{Z}_j$ and $\underline{Z}_j$ denote the upper and lower bounds respectively of the range in the j-th dimension of (p_t, I_t), and they can be obtained by interval evaluating TM.

Evaluating the remainder $I_{\sigma,i}$. The j-th component $I_{\sigma_j,i}$ of $I_{\sigma,i}$ is computed as a conservative remainder for the polynomial $p_{\sigma_j,i}$, and it can be obtain as a symmetric interval $[-\epsilon_j, \epsilon_j]$ such that

$$\epsilon_j = \max_{s=1,\cdots,m} \left(\left| p_{\sigma_j,i}(\frac{\overline{Z}_j - \underline{Z}_j}{m}(s - \frac{1}{2}) + \underline{Z}_j) - \sigma_j(\frac{\overline{Z}_j - \underline{Z}_j}{m}(s - \frac{1}{2}) + \underline{Z}_j) \right| + L_j \cdot \frac{\overline{Z}_j - \underline{Z}_j}{m} \right)$$

wherein L_j is a Lipschitz constant of σ_j with the domain (p_t, I_t), and m is the number of samples that are uniformly selected to estimate the remainder. The soundness of the error bound estimation above has been proven in [11] for multivariate Bernstein polynomials. Since univariate Bernstein polynomials, which we use in this paper, is a special case of multivariate Bernstein polynomials, our approach is also sound. A detailed proof is given in the appendix.

The following theorem states that a TM flowpipe computed by our approach is not only a range overapproximation of a reachable set segment, but also a function overapproximation for the dependency of a reachable state on its initial state. The proof is given in the appendix.

Theorem 1. *If $\mathcal{F}(\vec{x}_0, \tau)$ is the i-th TM flowpipe computed in the j-st control step, then for any initial state $\vec{x}_0 \in X_0$, the box $\mathcal{F}(\vec{x}_0, \tau)$ contains the actual reachable state $\varphi_{\mathcal{N}}(\vec{x}_0, (j-1)\delta_c + (i-1)\delta + \tau)$ for all $\tau \in [0, \delta]$.*

3.2 Selection of Polynomial Approximations

Since an activation function is univariate, both of its Taylor and Bernstein approximations have a size which is linear in the order k. Then we investigate the accuracy produced by both approximation forms. Since the main operation in the TM layer-by-layer propagation framework is the composition of TMs, we study the *preservation of accuracy* for both of the forms under the composition with a given TM. We first define the *Accuracy Preservation Problem.*

When a function $f(\vec{x})$ is overapproximated by a TM $(p(\vec{x}), I)$ w.r.t. a bounded domain D, the approximation quality, i.e., size of the overestimation, is directly reflected by the width of I, since $f(\vec{x}) = p(\vec{x})$ for all $\vec{x} \in D$ when I is zero by the TM definition. Given two order k TMs $(p_1(\vec{x}), I_1)$ and $(p_2(\vec{x}), I_2)$ which are overapproximations of the same function $f(\vec{x})$ w.r.t. a bounded domain $D \subset \mathbb{R}^n$, we use $(p_1(\vec{x}), I_1) \prec_k (p_2(\vec{x}), I_2)$ to denote that the width of I_1 is smaller than the width of I_2 in all dimensions, i.e., $(p_1(\vec{x}), I_1)$ is a more accurate overapproximation of $f(\vec{x})$ than $(p_2(\vec{x}), I_2)$.

Accuracy Preservation Problem. If both $(p_1(\vec{x}), I_1)$ and $(p_2(\vec{x}), I_2)$ are overapproximations of $f(\vec{x})$ with $\vec{x} \in D$, and $(p_1(\vec{x}), I_1) \prec_k (p_2(\vec{x}), I_2)$. Given another function $g(\vec{y})$ which is already overapproximated by a TM $(q(\vec{y}), J)$ whose range is contained in D. Then, *does* $p_1(q(\vec{y}) + J) + I_1 \prec_k p_2(q(\vec{y}) + J) + I_2$ *still hold using order* k *TM arithmetic?*

We give the following counterexample to show that the answer is **no**, i.e., although $(p_1(\vec{x}), I_1)$ is more accurate than $(p_2(\vec{x}), I_2)$, the composition $p_1(q(\vec{y}) + J) + I_1$ might not be a better order k overapproximation than $p_2(q(\vec{y}) + J) + I_2$ for the composite function $f \circ g$. Given $p_1 = 0.5 + 0.25x - 0.02083x^3$, $I_1 =$ [-7.93e-5, 1.92e-4], and $p_2 = 0.5 + 0.24855x - 0.004583x^3$, $I_2 =$ [-2.42e-4, 2.42e-4], which are both TM overapproximations for the sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ w.r.t. $x \in q(y) + J$ such that $q = 0.1y - 0.1y^2$, $J = [-0.1, 0.1]$, and $y \in [-1, 1]$. We have that $(p_1, I_1) \prec_3 (p_2, I_2)$, however after the compositions using order 3 TM arithmetic, the remainder of $p_1(q(y) + J) + I_1$ is $[-0.0466, 0.0477]$, while the remainder of $p_2(q(y) + J) + I_2$ is $[-0.0253, 0.0253]$, and we do not have $(p_1(q(y) + J) + I_1) \prec_3 (p_1(q(y) + J) + I_1)$.

Hence, we integrate an additional step in Algorithm 1 to replace line 4-6: in each iteration, both of Taylor and Bernstein overapproixmations are computed for each of the activation functions, and we choose the one that produces the smaller remainder interval I_r.

3.3 Symbolic Remainders in Layer-by-Layer Propagation

We describe the use of symbolic remainders (SR) in the layer-by-layer propagation of computing an NN output TM. The method was originally proposed in [7] for reducing the overestimation of TM flowpipes in the reachability computation for nonlinear ODEs, we adapt it particularly for reducing the error accumulation in the TM remainders during the layer-by-layer propagation. Unlike the BP technique whose purpose is to obtain tighter TMs for activation functions, the use of SR only aims at reducing the overestimation accumulation in the composition of a sequence of TMs each of which represents the input range of a layer.

Consider the TM composition for computing the output TM of a single layer in Fig. 3, the output TM $p_{\sigma,i}(W_i(p_i(\vec{x}_0) + I_i) + B_i) + I_{\sigma,i}$ equals to $Q_i W_i p_i(\vec{x}_0) + Q_i W_i I_i + Q_i B_i + p^R_{\sigma,i}(W_i(p_i(\vec{x}_0) + I_i) + B_i) + I_{\sigma,i}$ such that Q_i is the matrix of the linear coefficients in $p_{\sigma,i}$, and $p^R_{\sigma,i}$ consists of the terms in $p_{\sigma,i}$ of the degrees $\neq 1$.

Algorithm 2 TM output computation using symbolic remainders, input and output are the same as those in Algorithm 1

```
1: Setting $\mathcal{Q}$ as an empty array which can keep $M+1$ matrices;
2: Setting $\mathcal{J}$ as an empty array which can keep $M+1$ multidimensional intervals;
3: $\mathbb{J} \leftarrow 0$;
4: for $i = 1$ to $M+1$ do
5:     Computing the composite function $p_{\sigma,i}$ and the remainder interval $I_{\sigma,i}$ using the
       BP technique;
6:     Evaluating $q_i(\vec{x}_0) + J_i$ based on $\mathbb{J}$ and $\mathcal{Q}[1]I_1$;      # $\mathcal{Q}[1]I_1 = I_1$ when $i = 1$
7:     $\mathbb{J} \leftarrow J_i$;
8:     $\Phi_i = Q_i W_i$;
9:     for $j = 1$ to $i-1$ do
10:        $\mathcal{Q}[j] \leftarrow \Phi_i \cdot \mathcal{Q}[j]$;
11:    end for
12:    Adding $\Phi_i$ to $\mathcal{Q}$ as the last element;
13:    for $j = 2$ to $i$ do
14:        $\mathbb{J} \leftarrow \mathbb{J} + \mathcal{Q}[j] \cdot \mathcal{J}[j-1]$;
15:    end for
16:    Adding $J_i$ to $\mathcal{J}$ as the last element;
17: end for
18: Computing an interval enclosure $I_r$ for $\mathbb{J} + \mathcal{Q}[1]I_1$;      # interval evaluation
19: return $q_{M+1}(\vec{x}_0) + I_r$.
```

Therefore, the remainder I_i in the second term can be kept symbolically such that we do not compute $Q_iW_iI_i$ out as an interval but keep its transformation matrix Q_iW_i to the subsequent layers. Given the image S of an interval under a linear mapping, we use $\underline{S}$ to denote that it is kept symbolically, i.e., we keep the interval along with the transformation matrix, and $\overline{S}$ to denote that the image is evaluated as an interval.

Then we present the use of SR in layer-by-layer propagation. Starting from the NN input TM $(p_1(\vec{x}_0), I_1)$, the output TM of the first layer is computed as

$$\underbrace{Q_1W_1p_1(\vec{x}_0) + Q_1B_1 + p^R_{\sigma,1}(W_1(p_1(\vec{x}_0) + I_1) + B_1) + I_{\sigma,1}}_{q_1(\vec{x}_0)+J_1} + \underline{Q_1W_1I_1}$$

which can be kept in the form of $q_1(\vec{x}_0) + J_1 + \underline{Q_1W_1I_1}$. Using it as the input TM of the second layer, we have the following TM

$$\begin{aligned} & p_{\sigma,2}(W_2(q_1(\vec{x}_0) + J_1 + \underline{Q_1W_1I_1}) + B_2) + I_{\sigma,2} \\ = & \underbrace{Q_2W_2q_1(\vec{x}_0) + Q_2B_2 + p^R_{\sigma,2}(W_2(q_1(\vec{x}_0) + J_1 + \overline{Q_1W_1I_1}) + B_2) + I_{\sigma,2}}_{q_2(\vec{x}_0)+J_2} \\ & + \underline{Q_2W_2J_1} + \underline{Q_2W_2Q_1W_1I_1} \end{aligned}$$

for the output range of the second layer. Therefore the output TM of the i-th layer can be obtained as $q_i(\vec{x}_0) + \mathbb{J}_i + \underline{Q_iW_i \cdots Q_1W_1I_1}$ such that $\mathbb{J}_i = J_i + \underline{Q_iW_iJ_{i-1}} + \underline{Q_iW_iQ_{i-1}W_{i-1}J_{i-2}} + \cdots + \underline{Q_iW_i \cdots Q_2W_2J_1}$.

We present the SR method by Algorithm 2 in which we use two lists: $\mathcal{Q}[j]$ for $Q_i W_i \cdot \cdots \cdot Q_j W_j$ and $\mathcal{J}[j]$ for $\mathbb{J}_j$ to keep the intervals and their linear transformations. The symbolic remainder representation is replaced by its interval enclosure I_r at the end of the algorithm.

Time and space complexity. Although Algorithm 2 produces TMs with tighter remainders than Algorithm 1 because of the symbolic interval representations under linear mappings, it requires (1) two extra arrays to keep the intermediate matrices and remainder intervals, (2) two extra inner loops which perform $i-1$ and $i-2$ iterations in the i-th outer iteration. The size of $Q_i W_i \cdot \cdots \cdot Q_j W_j$ is determined by the rows in Q_i and the columns in W_j, and hence the maximum number of neurons in a layer determines the maximum size of the matrices in $\mathcal{Q}$. Similarly, the maximum dimension of J_i is also bounded by the maximum number of neurons in a layer. Because of the two inner loops, time complexity of Algorithm 2 is quadratic in M, whereas Algorithm 1 is linear in M.

4 Experiments

In this section, we perform a comprehensive empirical study of POLAR against state-of-the-art (SOTA) techniques. We first demonstrate the performance of POLAR on two examples with high dimensional states and multiple inputs, which are far beyond the ability of current SOTA techniques (Section 4.1). A comprehensive comparison with SOTA over the full benchmarks in [11, 13] is then given (Section 4.2). Finally, we present additional ablation studies, scalability analysis, and the ability to handle discrete-time systems (Section 4.3).

All our experiments were run on a machine with 6-core 2.90 GHz Intel Core i5 and 8GB of RAM. POLAR is implemented with C++. We present the results for POLAR, Verisig 2.0 and Sherlock using a single core without parallelization. The results of ReachNN* were computed on the same machine with the aid of GPU acceleration on an Nvidia GeForce RTX 2060 GPU.

State-of-the-art tools. We compare with SOTA tools in the NNCS reachability analysis literature, including Sherlock [8] (only works for ReLU), Verisig 2.0 [13] (only works for sigmoid and tanh), NNV [29], and ReachNN*[9][6].

4.1 High Dimensional Case Studies: Attitude Control & QUAD.

We consider an attitude control of a rigid body with 6 states and 3 control inputs [27], and quadrotor (QUAD) with 12 states and 3 control inputs [3] to evaluate the performance of POLAR on difficult problems. The complexity of these two example lies in the combination of the numbers of the state variables and control inputs. For each example, we trained a sigmoid neural-network controller and compare POLAR with Verisig 2.0 and NNV. The detailed setting of these two examples can be found in the Appendix.

The result for the attitude control benchmark is shown in Figure 4, and the result for the QUAD benchmark is shown in Figure 5a. In the attitude control

[6] The results of ReachNN* are based on GPU acceleration.

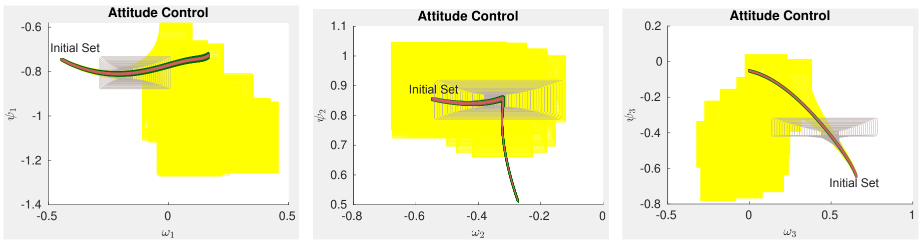
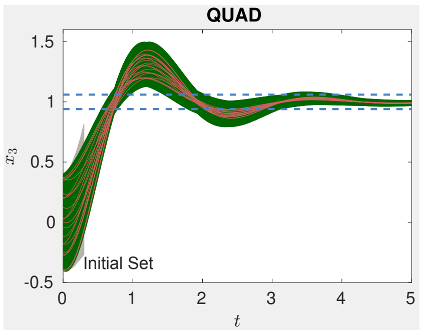
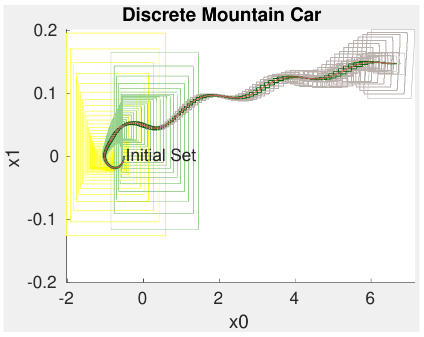


Fig. 4: Comparison between reachable sets of the 6-dimensional attitude control benchmark produced by POLAR (dark green), Verisig 2.0 (gray) and NNV (yellow). The red curves are simulated trajectories.



(a) QUAD



(b) Mountain Car

Fig. 5: (a) Results of QUAD. POLAR for 50 steps (dark green sets), Verisig 2.0 for 3 steps (grey sets), and simulation traces for 50 steps (red curves). It took POLAR 1533 seconds to compute the flowpipes for 50 steps. On the other hand, it took Verisig 2.0 more than 5 hours to compute the flowpipes for the first 3 steps, and at the 4th step, the remainders of the TM computed by Verisig 2.0 for the outputs of the neural-network controller already exploded to 10^{15}. NNV crashed with out-of-memory errors when computing the 1st step. (b) Results of Mountain Car. POLAR for 150 steps (dark green sets), Verisig 2.0 for 150 steps (grey sets), ReachNN* for 90 steps (light green sets), NNV for 65 steps, and simulation traces for 150 steps (red curves).

benchmark, POLAR computed the TM flowpipes for 30 control steps in 201 seconds. From Figure 4, We can observe that the flowpipes computed by POLAR are tight w.r.t. the simulated traces. As a comparison, although Verisig 2.0 [13] can handle this system in theory, its remainder exploded very quickly and the tool crashed after only a few steps. NNV computed flowpipes for 25 steps by doing extensive splittings on the state space and crashed with out-of-memory errors. In the QUAD benchmark, POLAR computed the TM flowpipes for 50

Table 1: V: number of state variables, σ: activation functions, M: number of hidden layers, n: number of neurons in each hidden layer. For each approach (POLAR, ReachNN*, Sherlock, Verisig 2.0), we give the runtime in seconds if it successfully verifies the property. 'Unknown': the property could not be verified. '–': the approach cannot be applied due to the type of σ.

#	V	NN Controller			**POLAR**	ReachNN*	Sherlock	Verisig 2.0
		σ	M	n		[9]	[8]	[13]
1	2	ReLU	2	20	**12**	26	42	–
		sigmoid	2	20	**17**	75	–	47
		tanh	2	20	**20**	Unknown	–	46
		ReLU+tanh	2	20	**13**	71	–	–
2	2	ReLU	2	20	**2**	5	3	–
		sigmoid	2	20	9	13	–	**7**
		tanh	2	20	**3**	73	–	Unknown
		ReLU+tanh	2	20	**2**	Unknown	–	–
3	2	ReLU	2	20	**16**	94	143	–
		sigmoid	2	20	**36**	146	–	44
		tanh	2	20	**26**	137	–	38
		ReLU+sigmoid	2	20	**15**	150	–	–
4	3	ReLU	2	20	**2**	8	21	–
		sigmoid	2	20	**3**	22	–	11
		tanh	2	20	**3**	21	–	10
		ReLU+tanh	2	20	**2**	12	–	–
5	3	ReLU	3	100	**13**	103	15	–
		sigmoid	3	100	76	**27**	–	190
		tanh	3	100	**76**	Unknown	–	179
		ReLU+tanh	3	100	**10**	Unknown	–	–
6	4	ReLU	3	20	**16**	1130	35	–
		sigmoid	3	20	**21**	13350	–	83
		tanh	3	20	**19**	2416	–	70
		ReLU+tanh	3	20	**15**	1413	–	–
ACC	6	tanh	3	20	**343**	Unknown	–	3344
QMPC	6	tanh	2	20	**61**	–[1]	–	652
Attitude Control	6	sigmoid	3	64	**201**	–[1]	–	Unknown
QUAD	12	sigmoid	3	64	**1533**	–[1]	–	Unknown

[1] This example has multi-dimensional control inputs. ReachNN* only supports NN controllers that produce single-dimensional control inputs.

control steps in 1533 seconds, while Verisig 2.0 and NNV took hours to compute flowpipes just for the first few steps.

4.2 Comparison over A Full Set of Benchmarks

We compare POLAR with the SOTA tools mentioned previously, including Sherlock, Verisig 2.0, NNV, and ReachNN* over the full benchmarks in [11, 13]. We refer to [11, 13] for more details of these benchmarks. The results are presented in Table 1 where NNV is not included since we were not able to successfully use it to prove any of the benchmarks likely because it is designed for linear systems. Similar results for NNV are also observed in [13]. We can see that POLAR successfully verifies all the cases and the runtime is **on average 8x and up to 94x faster**[7] compared with the tool with the second best efficiency. The "Unknown"

[7] These are lower bounds on the improvements since other tools terminated early for certain settings due to explosion of their computed flowpipes.

verification results either indicate the overapproximation of reachable set were too large for verifying the safety property or the tool terminated early due to an explosion of the overapproximation. POLAR achieves the best performance among all the tools (visualizations and detailed comparisons of the reachable sets can be found in the Appendix).

4.3 Discussion

POLAR demonstrates substantial performance improvement over existing tools. In this section, we seek to further explore the capability of POLAR. We conduct several experiments for the QUAD benchmark to better understand the limitation and scalability of POLAR. We also include a mountain car example to show that POLAR is able to handle discrete-time systems.

Ablation Studies. To explore the impact of the two proposed techniques, namely Bernstein polynomial interpolation (BP) and symbolic remainder (SR) on the overall performance, we conduct a series of experiments on the QUAD benchmark with different configurations. Table 2 shows the performance of POLAR with and without the proposed techniques SR and BP in the NN propagation: 1) TM: only TM arithmetic is used; 2) TM+SR: SR is used with TM arithmetic; 3) BP is used with TM arithmetic; and 4) Both BP and SR are used with TM arithmetic. Based on the results, we can observe that SR significantly improves the accuracy of the reachable set overapproximation. Finally, the combination of basic TM with BP and SP not only achieves the best accuracy, but also is the most efficient. While the additional BP and SR operations can incur runtime overhead compared with basic TM, they help to produce a tighter overestimation and thus reduce the state space being explored during reachability analysis. As a result, the overall performance including runtime is better.

The following further observations can be obtained from Table 2. (i) Both of the independent use of BP and SR techniques significantly improves the performance of reachable set overapproximations. (ii) When the BP technique is used, Bernstein approximation is often not used on activation functions, but the few times for which they are used significantly improve the accuracy. The reason of having this phenomenon is that Taylor and Bernstein approximations are similarly accurate in approximating activation functions with small domain. However, the Lagrange form-based remainder evaluation in Taylor polynomials performs better than the sample-based remainder evaluation in Bernstein polynomials in those cases. It can also be seen that for each X_0, the use of Bernstein approximation becomes more frequent when the TMs has larger remainders. (iii) When both BP and SR techniques are used, the approach produces the tightest TMs compared with the other columns in the table even though the use Bernstein approximation is less often. The reason is that the remainders of the TMs are already well-limited and most of the activation functions handled in the reachability computation are with a "small" TM domain.

Scalability Analysis. Table 1 shows that POLAR can handle much larger NNCSs compared with the current SOTA. To better understand the scalability of POLAR, we further conduct scalability analysis on the size of the NN controller

Table 2: Ablation Studies for POLAR on the QUAD benchmark. We compare the width of TM remainder on x_3 at the 50th step under different settings. For settings with BP, we also list the percentage of times where BP is used among 9600 neurons. If a setting cannot compute flowpipes for all 50 steps, it is marked as *Unknown*. X_0 is the radius of the initial set. k is the order of the TM.

X_0	k	TM		TM+SR		TM+BP			TM+BP+SR		
		Width	Time (s)	Width	Time (s)	Width	Time (s)	BP %	Width	Time (s)	BP %
0.05	2	7.5e-04	229	1.3e-04	233	6.8e-04	228	5.79%	1.2e-04	231	1.34%
	3	5.2e-04	273	6.5e-05	251	5.0e-04	274	3.62%	6.5e-05	251	0%
	4	4.9.e-04	332	6.2e-05	270	4.7e-04	336	3.57%	6.2e-05	270	0%
0.1	2	*Unknown*	–	2.3e-03	319	1.0e-02	325	9.68%	1.1e-03	289	4.80%
	3	1.8e-03	352	2.2e-04	287	1.7e-03	349	6.85%	2.2e-04	287	0%
	4	1.6e-03	431	1.9e-04	304	1.5e-03	427	6.70%	1.9e-04	304	0%
0.2	2	*Unknown*	–	*Unknown*	–	*Unknown*	–	–	*Unknown*	–	–
	3	9.0e-03	721	1.9e-03	412	7.8e-03	670	4.03%	1.6e-03	394	0.77%
	4	5.0e-03	761	9.2e-04	403	4.7e-03	728	4.38%	8.1e-04	396	0.07%
0.4	2	*Unknown*	–	*Unknown*	–	*Unknown*	–	–	*Unknown*	–	–
	3	*Unknown*	–	*Unknown*	–	*Unknown*	–	–	*Unknown*	–	–
	4	*Unknown*	–	*Unknown*	–	*Unknown*	–	–	3.7e-02	1533	3.25%

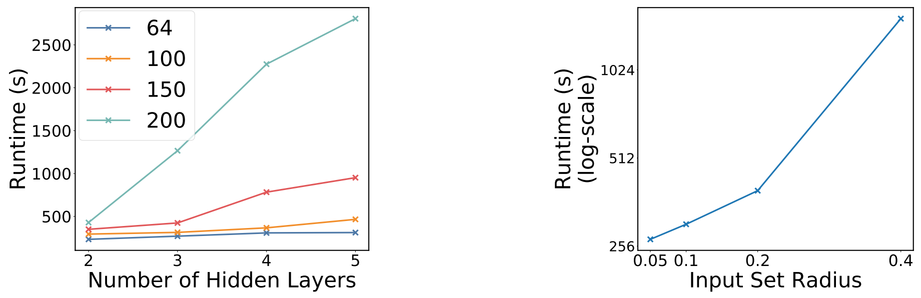


Fig. 6: Scalability analysis for POLAR on the QUAD benchmark. We present the runtime of QUAD for 50 steps reachability analysis. Under all settings, POLAR can verify that the system reaches the target set at the 50th step. Left figure: Runtime on different neural network architectures with the input set radius as 0.05. We study neural-network controllers with different number of layers (2, 3, 4, 5) and neurons (64, 100, 150, 200). Right figure: Runtime on the different input set radius of the QUAD benchmark. We use the same network in Figure 5 which has 3 hidden layers with 64 neurons in each layer.

and the width of the initial set using the QUAD benchmark. The experiment results in Figure 6 for the neural networks with different widths and depths show that POLAR scales well on the number of layers and the number of neurons in each layer in the NN controller. On the other hand, the time cost grows rapidly when the width of the initial set becomes larger. Such a phenomenon already exists in the literature for reachability analysis of ODE systems [5]. The reason for this is that when the initial set is larger, it is more difficult to track the state dependencies and requires keeping more terms in a TM flowpipe.

Discrete-time NNCS. Finally, we use Mountain car, a common benchmark in Reinforcement Learning literature, to show that POLAR also works on discrete-time systems. The detailed setting can be found in the Appendix B. The comparison with Verisig 2.0, ReachNN* and NNV is shown in Figure 5b. POLAR also outperforms these tools substantially for this example.

5 Conclusion

In this paper, we propose POLAR, a polynomial arithmetic framework, which integrates TM flowpipe construction, Bernstein overapproximation, and symbolic remainder method to efficiently compute reachable set overapproximations for NNCS. Empirical comparison over a suite of benchmarks shows that POLAR performs significantly better than SOTAs in terms of both computation efficiency and tightness of reachable set estimation.

References

1. Althoff, M.: An introduction to CORA 2015. In: International Workshop on Applied veRification for Continuous and Hybrid Systems (ARCH). EPiC Series in Computing, vol. 34, pp. 120–151 (2015)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
3. Beard, R.: Quadrotor dynamics and control rev 0.1 (2008)
4. Berz, M., Makino, K.: Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. Reliable computing **4**, 361–369 (1998)
5. Chen, X.: Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models. Ph.D. thesis, RWTH Aachen University (2015)
6. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Proc. of CAV'13. LNCS, vol. 8044, pp. 258–263 (2013)
7. Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: Proc. of RTSS'16. pp. 13–24 (2016)
8. Dutta, S., Chen, X., Sankaranarayanan, S.: Reachability analysis for neural feedback systems using regressive polynomial rule inference. In: Proc. of HSCC'19. pp. 157–168. ACM (2019)
9. Fan, J., Huang, C., Chen, X., Li, W., Zhu, Q.: ReachNN*: A tool for reachability analysis of neural-network controlled systems. In: Proceedings of International Symposium on Automated Technology for Verification and Analysis (ATVA). LNCS, vol. 12302, pp. 537–542. Springer (2020)
10. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Proceedings of International Conference on Computer Aided Verification (CAV). Lecture Notes in Computer Science, vol. 6806, pp. 379–395 (2011)
11. Huang, C., Fan, J., Li, W., Chen, X., Zhu, Q.: ReachNN: Reachability analysis of neural-network controlled systems. ACM Trans. Embed. Comput. Syst. **18**(5s), 106:1–106:22 (2019)
12. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Proc. of CAV'17. LNCS, vol. 10426, pp. 3–29. Springer (2017)

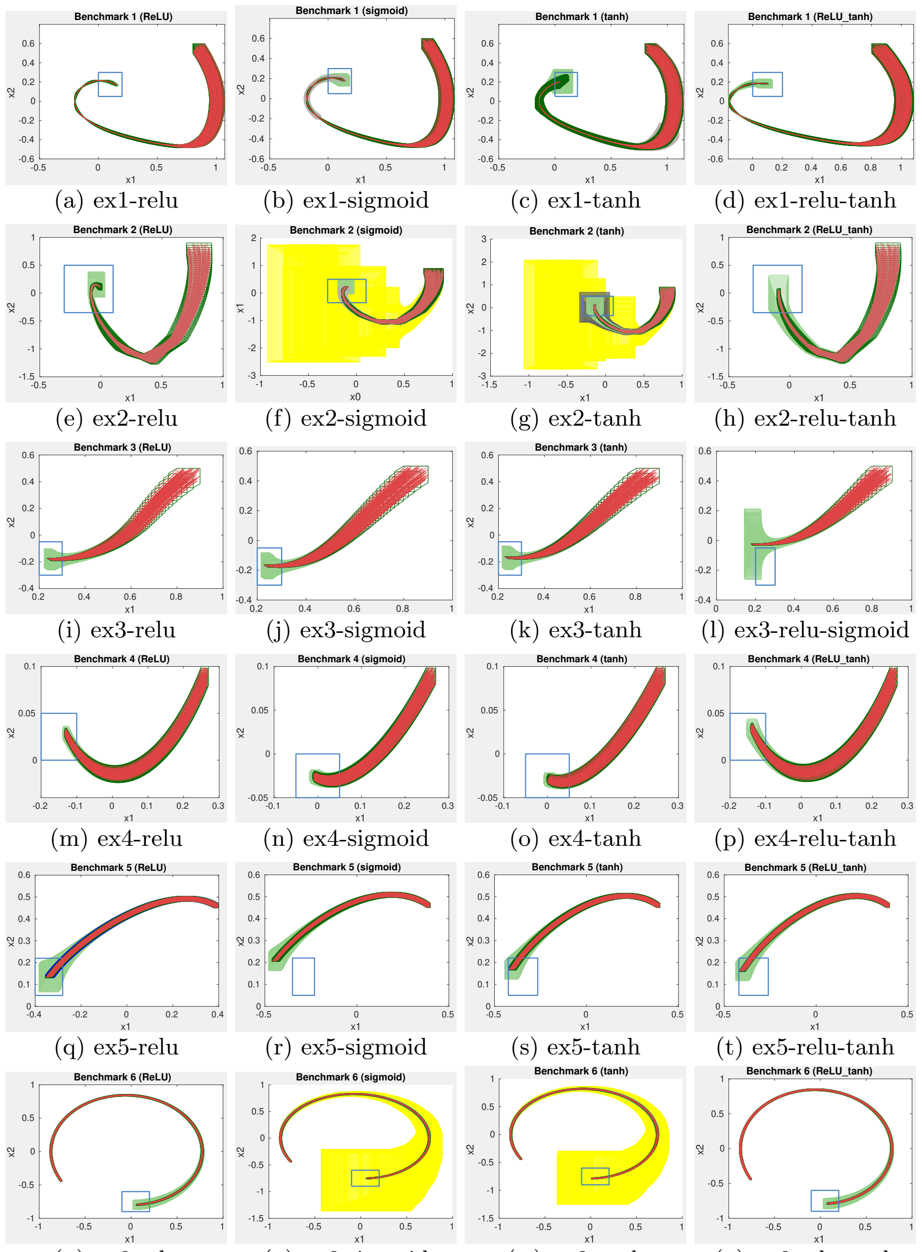
13. Ivanov, R., Carpenter, T., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In: Proc. of CAV'21. LNCS, vol. 12759, pp. 249–262. Springer (2021)
14. Ivanov, R., Carpenter, T.J., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verifying the safety of autonomous systems with neural network controllers. ACM Trans. Embed. Comput. Syst. **20**(1), 7:1–7:26 (2021)
15. Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., Lee, I.: Verisig: verifying safety properties of hybrid systems with neural network controllers. In: Proc. of HSCC'18. pp. 169–178. ACM (2019)
16. Jaulin, L., Kieffer, M., Didrit, O., Walter, É.: Interval analysis. In: Applied Interval Analysis. Springer (2001)
17. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Proc. of CAV'17. LNCS, vol. 10426, pp. 97–117. Springer (2017)
18. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research **17**(1), 1334–1373 (2016)
19. Lorentz, G.G.: Bernstein Polynomials. American Mathematical Society (2013)
20. Lygeros, J., Tomlin, C.J., Sastry, S.: Controllers for reachability specifications for hybrid systems. Automatica **35**(3), 349–370 (1999)
21. Makino, K., Berz, M.: Taylor models and other validated functional inclusion methods. International Journal of Pure and Applied Mathematics **4**(4), 379–456 (2003)
22. Meiss, J.D.: Differential Dynamical Systems. SIAM publishers (2007)
23. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
24. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM (2009)
25. Pan, Y., Cheng, C., Saigol, K., Lee, K., Yan, X., Theodorou, E.A., Boots, B.: Agile autonomous driving using end-to-end deep imitation learning. In: Proc. of RSS'18 (2018)
26. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: HSCC. pp. 477–492. Springer (2004)
27. Prajna, S., Parrilo, P.A., Rantzer, A.: Nonlinear control synthesis by convex optimization. IEEE Transactions on Automatic Control **49**(2), 310–314 (2004)
28. Singh, G., Ganvir, R., Püschel, M., Vechev, M.T.: Beyond the single neuron convex barrier for neural network certification. In: Proc. of NeurIPS'19. pp. 15072–15083 (2019)
29. Tran, H., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Proc. of CAV'20. LNCS, vol. 12224, pp. 3–17. Springer (2020)
30. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: Proc. of USENIX Security (USENIX). pp. 1599–1614 (2018)
31. Weng, T.W., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning (ICML) (2018)
32. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 4944–4953 (2018)

A Additional Experimental Results

Here, we present additional plots of reachable sets computed by different techniques for the benchmarks in Section 4 of the main paper.

For each benchmark, the goal is to check whether the system will reach a given target set. For each tool and in each test, if the computed reachable set overapproximation for the last control step lies entirely in the target set, we consider the tool to have successfully verified the reachability property. If the overapproximation of the reachable set does not intersect with the target set, the tool would have successfully disproved the reachability property. Otherwise, we consider the verification result to be unknown.

Results of Benchmark 1-6, the ACC benchmark, and the QMPC benchmark are shown in Figure 7, 8, 9 respectively, while the results of the Attitude control benchmark and the QUAD benchmark are shown previously in Figure 4 and Figure 5a. The red trajectories are sample system executions and should be contained entirely by the flowpipes computed by each tool. The dark green sets are the flowpipes computed by POLAR. The light green sets are the flowpipes computed by ReachNN* [11, 9]. The blue sets are the flowpipes computed by Sherlock [8]. The grey sets are the flowpipes computed by Verisig 2.0 [14]. In some benchmarks, the reachable sets computed by Verisig 2.0 are almost overlapping with the reachable sets computed by POLAR. However, POLAR takes much less time to compute the reachable sets compared to Verisig 2.0 as shown in Table 1 of the main paper. We also show results from NNV [29] in yellow for some of the benchmarks. For the rest, NNV used up all of the system memory (8GB) and could not finish the computation. Our observations are consistent with those in [14] where NNV is not able to verify any of these benchmarks. The blue box represents the target set in each test. POLAR produces the tightest reachable set estimation and successfully proves or disproves the reachability property for all the examples.



(a) ex1-relu (b) ex1-sigmoid (c) ex1-tanh (d) ex1-relu-tanh

(e) ex2-relu (f) ex2-sigmoid (g) ex2-tanh (h) ex2-relu-tanh

(i) ex3-relu (j) ex3-sigmoid (k) ex3-tanh (l) ex3-relu-sigmoid

(m) ex4-relu (n) ex4-sigmoid (o) ex4-tanh (p) ex4-relu-tanh

(q) ex5-relu (r) ex5-sigmoid (s) ex5-tanh (t) ex5-relu-tanh

(u) ex6-relu (v) ex6-sigmoid (w) ex6-tanh (x) ex6-relu-tanh

Fig. 7: Results of Benchmarks. We can see that except for ex2-sigmoid, POLAR produces the tightest reachable set estimation (dark green sets) and successfully proves or disproves the reachability property for all the examples. This is in comparison with other state-of-the-art tools including ReachNN* [11, 9] (light green sets), Sherlock [8] (blue sets), Verisig 2.0 [14] (grey sets), and NNV [29] (yellow sets). Except for (f), (g), (v) and (w), NNV used up all of the system memory and could not finish the computation.

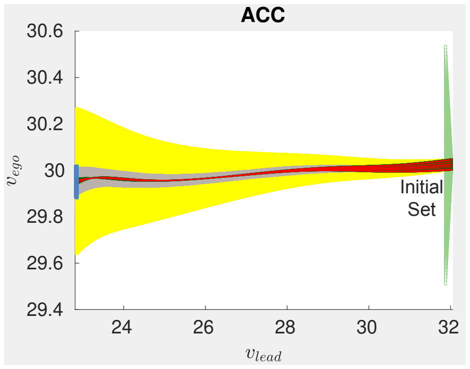


Fig. 8: Results of Adaptive Cruise Control (ACC). POLAR for 50 steps (dark green sets), Verisig 2.0 for 50 steps (grey sets), ReachNN* for 3 steps (light green sets), NNV for 50 steps (yellow sets), and simulation traces for 50 steps (red curves).

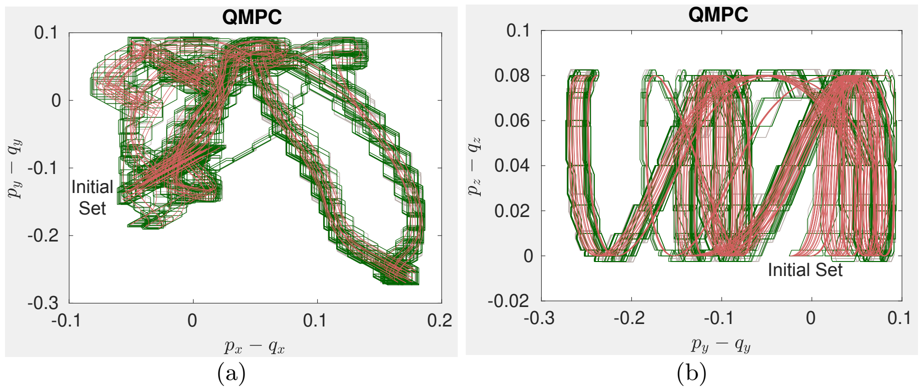


Fig. 9: Results of QMPC. POLAR for 30 steps (dark green sets), Verisig 2.0 for 30 steps (grey sets), and simulation traces for 30 steps (red curves).

B Discrete-time Mountain Car Benchmark

Example 1 (MC). In this benchmark, an under-powered car targets to drive up a steep hill. Since the car does not have enough power to Widthelerate up the hill, it needs to drive up the opposite hill first to gain enough momentum. The car has the following discrete-time dynamics:

$$\begin{cases} x_0[t+1] = x_0[t] + x_1[t], \\ x_1[t+1] = x_1[t] + 0.0015 \cdot u[t] - 0.0025 \cdot \cos(3 \cdot x_0[t]). \end{cases}$$

For this benchmark, the initial set is $x_0 \in [-0.53, -0.5]$ and $x_1 = 0$. The target is $x_0 \geq 0.2$ and $x_1 \geq 0$ where the car reaches the top of the hill and is moving forward. The total control steps N is 150.

C Benchmarks with High Dimensional States and Multiple Outputs

Example 2 (Attitude Control). We consider the attitude control of a rigid body with six states and three inputs as a physically illustrating example [27]. The system dynamics is

$$\begin{cases} \dot{\omega}_1 = 0.25(u_0 + \omega_2\omega_3), \qquad \dot{\omega}_2 = 0.5(u_1 - 3\omega_1\omega_3), \qquad \dot{\omega}_3 = u_2 + 2\omega_1\omega_2, \\ \dot{\psi}_1{=}0.5\left(\omega_2(\psi_1^2{+}\psi_2^2{+}\psi_3^2{-}\psi_3){+}\omega_3(\psi_1^2{+}\psi_2^2{+}\psi_2{+}\psi_3^2){+}\omega_1(\psi_1^2{+}\psi_2^2{+}\psi_3^2{+}1)\right), \\ \dot{\psi}_2{=}0.5\left(\omega_1(\psi_1^2{+}\psi_2^2{+}\psi_3^2{+}\psi_3){+}\omega_3(\psi_1^2{-}\psi_1{+}\psi_2^2{+}\psi_3^2){+}\omega_2(\psi_1^2{+}\psi_2^2{+}\psi_3^2{+}1)\right), \\ \dot{\psi}_3{=}0.5\left(\omega_1(\psi_1^2{+}\psi_2^2{-}\psi_2{+}\psi_3^2){+}\omega_2(\psi_1^2{+}\psi_1{+}\psi_2^2{+}\psi_3^2){+}\omega_3(\psi_1^2{+}\psi_2^2{+}\psi_3^2{+}1)\right). \end{cases}$$

wherein the state $\vec{x}{=}(\omega, \psi)$ consists of the angular velocity vector in a body-fixed frame $\omega{\in}\mathbb{R}^3$, and the Rodrigues parameter vector $\psi{\in}\mathbb{R}^3$.

The control torque $u{\in}\mathbb{R}^3$ is updated every 0.1 second by a neural network with 3 hidden layers, each of which has 64 neurons. The activations of the hidden layers are sigmoid and identity, respectively. We train the neural-network controller using supervised learning methods to learn from a known nonlinear controller [27]. The initial state set is:

$$\omega_1 \in [-0.45, -0.44], \omega_2 \in [-0.55, -0.54], \omega_3 \in [0.65, 0.66],$$
$$\psi_1 \in [-0.75, -0.74], \psi_2 \in [0.85, 0.86], \psi_3 \in [-0.65, -0.64].$$

Example 3 (QUAD). We study a neural-network controlled quadrotor (QUAD) with 12 states [3]. For the states, we have the inertial (north) position x_1, the inertial (east) position x_2, the altitude x_3, the longitudinal velocity x_4, the lateral velocity x_5, the vertical velocity x_6, the roll angle x_7, the pitch angle x_8, the yaw angle x_9, the roll rate x_{10}, the pitch rate x_{11}, and the yaw rate x_{12}. The control torque $u \in \mathbb{R}^3$ is updated every 0.1 second by a neural network with 3 hidden layers, each of which has 64 neurons. The activations of the hidden layers and the output layer are sigmoid and identity, respectively.

$$
\begin{cases}
\dot{x}_1 = \cos(x_8)\cos(x_9)x_4 + \left(\sin(x_7)\sin(x_8)\cos(x_9) - \cos(x_7)\sin(x_9)\right)x_5 \\
\qquad + \left(\cos(x_7)\sin(x_8)\cos(x_9) + \sin(x_7)\sin(x_9)\right)x_6 \\
\dot{x}_2 = \cos(x_8)\sin(x_9)x_4 + \left(\sin(x_7)\sin(x_8)\sin(x_9) + \cos(x_7)\cos(x_9)\right)x_5 \\
\qquad + \left(\cos(x_7)\sin(x_8)\sin(x_9) - \sin(x_7)\cos(x_9)\right)x_6 \\
\dot{x}_3 = \sin(x_8)x_4 - \sin(x_7)\cos(x_8)x_5 - \cos(x_7)\cos(x_8)x_6 \\
\dot{x}_4 = x_{12}x_5 - x_{11}x_6 - g\sin(x_8) \\
\dot{x}_5 = x_{10}x_6 - x_{12}x_4 + g\cos(x_8)\sin(x_7) \\
\dot{x}_6 = x_{11}x_4 - x_{10}x_5 + g\cos(x_8)\cos(x_7) - g - u_1/m \\
\dot{x}_7 = x_{10} + \sin(x_7)\tan(x_8)x_{11} + \cos(x_7)\tan(x_8)x_{12} \\
\dot{x}_8 = \cos(x_7)x_{11} - \sin(x_7)x_{12} \\
\dot{x}_9 = \dfrac{\sin(x_7)}{\cos(x_8)}x_{11} - \sin(x_7)x_{12} \\
\dot{x}_{10} = \dfrac{J_y - J_z}{J_x}x_{11}x_{12} + \dfrac{1}{J_x}u_2 \\
\dot{x}_{11} = \dfrac{J_z - J_x}{J_y}x_{10}x_{12} + \dfrac{1}{J_y}u_3 \\
\dot{x}_{12} = \dfrac{J_x - J_y}{J_z}x_{10}x_{11} + \dfrac{1}{J_z}\tau_\psi
\end{cases}
$$

The initial set is:

$$
\begin{aligned}
&x_1{\in}[-0.4, 0.4], x_2{\in}[-0.4, 0.4], x_3{\in}[-0.4, 0.4], x_4{\in}[-0.4, 0.4], \\
&x_5{\in}[-0.4, 0.4], x_6{\in}[-0.4, 0.4], x_7{=}0, x_8{=}0, x_9{=}0, x_{10}{=}0, x_{11}{=}0, x_{12}{=}0
\end{aligned}
$$

The control goal is to stabilize the attitude x_3 to a goal region $[0.94, 1.06]$.

Example 4 (Discrete-Time Mountain Car (MC)). We consider a common benchmark in Reinforcement Learning problems, namely Mountain Car. In this benchmark, an under-powered car targets to drive up a steep hill. Since the car does not have enough power to accelerate up the hill, it needs to drive up the opposite hill first to gain enough momentum. The car has the following discrete-time dynamics:

$$
\begin{aligned}
x_0[t+1] &= x_0[t] + x_1[t], \\
x_1[t+1] &= x_1[t] + 0.0015 \cdot u[t] - 0.0025 \cdot \cos(3 \cdot x_0[t]).
\end{aligned}
$$

For this benchmark, the initial set is $x_0 \in [-0.53, -0.5]$ and $x_1 = 0$. The target is $x_0 \geq 0.2$ and $x_1 \geq 0$ where the car reaches the top of the hill and is moving forward. The total control steps N is 150.

D Theorem Proof

D.1 Proof of Soundness of Sampling-based Error Analysis

Proof. The input range of an activation function σ_j is subdivided into m line segments. Consider the i-th segment $[\frac{\overline{Z}_j-\underline{Z}_j}{m}(i-1)+\underline{Z}_j, \frac{\overline{Z}_j-\underline{Z}_j}{m}(i)+\underline{Z}_j]$, and let $c=\frac{\overline{Z}_j-\underline{Z}_j}{m}(i-\frac{1}{2})+\underline{Z}_j$ be the center of the segment. The difference between the Bernstein polynomial p_σ^j and the activation function at the center of the i-th segment is computed as $\left|p_\sigma^j(c)-\sigma_j(c)\right|$. Then, the value of ϵ_j can be bounded by this difference at the center, as well as the product between the Lipschitz constant of the activation function with respect to this segment L_j and the size of the segment $\frac{\overline{Z}_j-\underline{Z}_j}{m}$, i.e., $L_j\cdot\frac{\overline{Z}_j-\underline{Z}_j}{m}$. The detailed deduction is given below.

$$
\begin{aligned}
&|p_{\sigma_j,i}(x)-\sigma_j(x)| \\
=&|p_{\sigma_j,i}(x)-p_{\sigma+j,i}(c)+p_{\sigma_j,i}(c)-\sigma_j(c)+\sigma_j(c)-\sigma_j(x)| \\
\leq&|p_{\sigma_j,i}(x)-p_{\sigma_j,i}(c)|+|p_{\sigma_j,i}(c)-\sigma_j(c)|+|\sigma_j(c)-\sigma_j(x)| && \text{Triangle inequality} \\
\leq&|p_{\sigma_j,i}(x)-p_{\sigma_j,i}(c)|+|p_{\sigma_j,i}(c)-\sigma_j(c)|+L_j\cdot\frac{\overline{Z}_j-\underline{Z}_j}{2m} && \text{Lipschitz continuity for } \sigma_j \\
\leq&L_j\cdot\frac{\overline{Z}_j-\underline{Z}_j}{2m}+|p_{\sigma_j,i}(c)-\sigma_j(c)|+L_j\cdot\frac{\overline{Z}_j-\underline{Z}_j}{2m} && \text{Lipschitz continuity for } p_{\sigma_j,i} \\
=&|p_{\sigma_j,i}(c)-\sigma_j(c)|+L_j\frac{\overline{Z}_j-\underline{Z}_j}{m}
\end{aligned}
$$

Note that Bernstein polynomial $p_{\sigma_j,i}$ has the same Lipschitz constant with σ_j. Thus we also use L_j to bound $|p_{\sigma_j,i}(x)-p_{\sigma_j,i}(c)|$ in the deduction. The error bound over the whole range $[\underline{Z}_j,\overline{Z}_j]$ should be the largest error bound among all the segments. □.

D.2 Proof of Theorem 1

Proof. First, due to the overapproximation property of our methods, any of our Bernstein overapproximation $p_{\sigma,I}+I_{\sigma,i}$ satisfied that for $\vec{z}$ in the domain on which $I_{\sigma,i}$ is evaluated, we have that $\sigma(\vec{z})\in p_{\sigma,i}(\vec{z})+I_{\sigma,i}$. Therefore, by the overapproximation property of TM arithmetic, the returned $(p_r(\vec{x}_0),I_r)$ of Algorithm 1 or 2 is a state-wise overapproximation of the control input range w.r.t. the TM variable $\vec{x}_0\in X_0$ wherein X_0 is the NNCS initial set.

We prove Theorem 1 by an induction on the number of control steps j. Assume that $N=\delta_c/\delta$ is the number of flowpipes computed in each control step.

Base Case. When $j=1$, the TM flowpipes are computed for the reachable set in the first control step and the evolution is under the pure continuous dynamics $\dot{\vec{x}}=f(\vec{x},\vec{u}_0)$, $\dot{\vec{u}}=0$ with $\vec{x}(0)\in X_0$ and $\vec{u}(0)=\kappa(\vec{x}(0))$. The image of the mapping $\kappa(\vec{x}_0)$ from $\vec{x}_0\in X_0$ is overapproximated by a TM $(p_r(\vec{x}_0),I_r)$ with $\vec{x}_0\in X_0$. Hence, by performing TM flowpipe construction for the ODE $\dot{\vec{x}}=f(\vec{x},\vec{u})$, $\dot{\vec{u}}=0$ with the initial set $\vec{u}(0)\in p_r(\vec{x}_0)+I_r$, $\vec{x}(0)=\vec{x}_0$, we have that for

any $i = 1, \ldots, N$, the i-th TM flowpipe $\mathcal{F}_i(\vec{x}_0, \tau)$ contains the exact reachable state at the time $(i-1)\delta + \tau$ for $\tau \in [0, \delta]$.
Induction. When $j > 1$, we assume that the local initial set $\hat{X}_{j-1} = (p_0(\vec{x}_0), I_0)$ is a state-wise overapproximation of the reachable set at the time $j\delta_c$ from any $\vec{x}_0 \in X_0$. Then the TM $(p_r(\vec{x}_0), I_r)$ is a state-wise overapproximation for the control input set $\kappa(\hat{X}_{j-1})$, i.e., the NN controller's output produced based on the $j\delta_c$-time state in the execution from an initial state $\vec{x}_0 \in X_0$ is contained in the box $p_r(\vec{x}_0) + I_r$ for any $\vec{x}_0 \in X_0$. Hence, for any $i = 1, \ldots, N$, the i-th flowpipe $\mathcal{F}_i(\vec{x}_0, \tau)$ computed for the ODE $\dot{\vec{x}} = f(\vec{x}, \vec{u})$, $\dot{\vec{u}} = 0$ with the initial set $\vec{u}(0) \in p_r(\vec{x}_0) + I_r$, $\vec{x}(0) \in \hat{X}_{j-1}$ contains the actual reachable state $\varphi_{\mathcal{N}}(\vec{x}_0, (j-1)\delta_c + (i-1)\delta + \tau)$ for any $\tau \in [0, \delta]$. $\square$