POLAR: A POLYNOMIAL ARITHMETIC FRAMEWORK FOR VERIFYING NEURAL-NETWORK CONTROLLED SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose POLAR, a **pol**ynomial **ar**ithmetic framework that leverages polynomial overapproximations with interval remainders for bounded-time reachability analysis of neural-network controlled systems (NNCSs). Compared with existing arithmetic approaches that use standard Taylor models, our framework uses a novel approach to iteratively overapproximate the neuron output ranges layer-by-layer via a combination of Bernstein polynomial interpolation for continuous activation functions and Taylor model arithmetic for the other operations. This approach overcomes the main drawback in the standard Taylor model arithmetic, i.e. its inability to handle functions that cannot be well approximated by Taylor polynomials, and significantly improve the accuracy and efficiency of reachable states computation for NNCSs. To further tighten the overapproximation, our method keeps the Taylor model remainders symbolic under the linear mappings when propagating Taylor models across the neural-network controller. We show that POLAR can be seamlessly integrated with existing Taylor model flowpipe construction techniques, and our approach significantly outperforms the current state-of-the-art techniques on a suite of benchmarks.

1 INTRODUCTION

Neural networks have been increasingly used as the central decision makers in a variety of tasks Mnih et al. (2015); Lillicrap et al. (2016); Pan et al. (2018). 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 $\boldsymbol{x}$, at discrete time steps, and computes the corresponding control values $\boldsymbol{u}$ for updating the system dynamics which is defined by an ordinary differential equation (ODE) over $\boldsymbol{x}$ and $\boldsymbol{u}$. The bounded-time reachability analysis problem of an NNCS is to compute an (over-approximated) reachable set that contains all the trajectories starting from the initial set for a finite number of control steps. Figure 2 shows an illustration of reachable sets for 4 steps, where the orange regions are the reachable set, and two red arrowed curves are two exemplary trajectories starting from 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, further exacerbated by the coupling of the controller and the plant. Since exact reachability of general nonlinear systems is undecidable Alur & Dill (1994), current approaches for reachability analysis of nonlinear dynamical systems aim at computing a tight overapproximation of the reachable sets Dreossi et al. (2016); Lygeros et al. (1999); Yang et al. (2016); Prajna & Jadbabaie (2004); Huang et al. (2017a); Frehse et al. (2011); Chen et al. (2013); Althoff (2015). Earlier works on NNCS verification drew on techniques for computing the output ranges of neural networks Huang et al. (2017b); Katz et al. (2017); Dutta et al. (2018); Wang et al. (2018); Weng et al. (2018); Zhang et al. (2018); Singh et al. (2019) when integrating with the aforementioned reachability analysis. However, they have 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 Dutta et al. (2019); Huang et al. (2019). In particular, since their primary goal is to bound the output range of the neural network instead of approximating its input-output function,

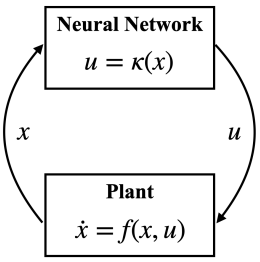


Figure 1: A typical NNCS model.

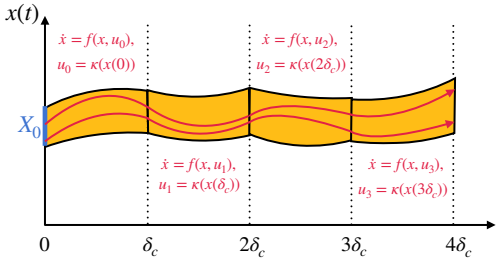


Figure 2: Executions over 4 control steps.

they *cannot be used to track dependency across the closed-loop system and across multiple time steps in reachability analysis*.

Direct end-to-end approximation Dutta et al. (2019); Huang et al. (2019); Fan et al. (2020) and *layer-by-layer propagation* Ivanov et al. (2019; 2021b;a) are the two main categories of approaches in the NNCS verification literature. The former directly computes a function approximation of the neural network and suffers from efficiency problems, i.e. they cannot handle systems with more than a few dimensions due to the need to sample in the input space. The latter approach tries to exploit the neural network structure and uses Taylor model (TM) arithmetic to more efficiently obtain a function approximation by propagating the TMs layer by layer. A Taylor model (p, I) consists of a polynomial p for point-wise approximation, and an interval remainder I to bound the approximation error. However, due to the limitations of basic TM arithmetic, these approaches *cannot handle non-differentiable activation functions and suffer from rapid growth of the interval remainders* during propagation which effectively degrades the analysis to an interval analysis.

In this paper, we tackle the challenge of dependency tracking by constructing a function overapproximation, specifically a Taylor model (TM) approximation, of the neural-network controller using a unified **pol**ynomial **ar**ithmetic framework (POLAR) that enables precise layer-by-layer propagation of TMs for general feed-forward neural networks. POLAR addresses the key challenges of applying basic TM arithmetic through a novel use of *univariate Bernstein polynomial interpolation* to handle non-differentiable activation functions and *symbolic remainders* to taper the growth of the interval remainders. For the former, we leverage an efficient sampling-based analysis to provide a sound overapproximation. The latter avoids the so-called wrapping effect Jaulin et al. (2001) that can lead to rapid growth of the interval remainders in linear mappings.

In summary, our work makes the following novel contributions.

- We propose a novel polynomial arithmetic framework for bounded-time reachability analysis of NNCSs, keeping track of state-wise dependency across the closed-loop system. Compared to existing propagation-based approaches, our framework has the advantage of being able to handle NN controllers with general and heterogeneous activation functions.
- We propose neuron-wise Bernstein polynomial interpolation and show that it can be seamlessly integrated with Taylor model approximations. In addition, we present the first application of symbolic remainders to tightening the overapproximation of neural network behaviors.
- We conduct a comprehensive comparison of our approach with state-of-the-art techniques, including an evaluation on the full set of benchmarks published in these works, showing the overwhelming advantage of our proposed approach.

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 $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ wherein the state variables and control inputs are denoted by the vectors $\boldsymbol{x}$ and $\boldsymbol{u}$ 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 Meiss (2007). We denote the input-output mapping of the neural network controller as κ, and the controller is triggered every δ_c time which is also called the *control*

stepsize, then a system *execution (trajectory)* is produced in the following way. Starting from an initial state $\boldsymbol{x}(0)$, the controller senses the system state at the beginning of every control step $t{=}j\delta_c$ for $j{=}0, 1, \ldots$, and update the control inputs to $\boldsymbol{v}_j{=}\kappa(\boldsymbol{x}(j\delta_c))$. The system's dynamics in that control step is governed by the ODE $\dot{\boldsymbol{x}}{=}f(\boldsymbol{x}, \boldsymbol{v}_j)$. Given an initial state set $X_0 \subset \mathbb{R}^n$, all executions from a state in it 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 $\boldsymbol{x}_0 \in X_0$ is $\varphi_{\mathcal{N}}(\boldsymbol{x}_0, t)$. We call a state $\boldsymbol{x}' \in \mathbb{R}^n$ *reachable* if there exists $\boldsymbol{x}_0 \in X_0$ and $t \geq 0$ such that $\boldsymbol{x}' = \varphi_{\mathcal{N}}(\boldsymbol{x}_0, t)$. The *reachability problem* on NNCS is to decide whether a state is reachable in a given NNCS, and it is *undecidable* due to their higher expressiveness than the two-counter machines on which the reachability problem is undecidable Alur & Dill (1994). Many formal verification problems can be reduced to the reachability problem. For example, the safety verification problem can be reduced to the reachability problem 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 time segments, that is, in each control step $[(j-1)\delta_c, j\delta_c]$ for $j = 1, \ldots, K$, the reachable set is overapproximated by a group of flowpipes $\mathcal{F}_1(\boldsymbol{x}_0, \tau), \ldots, \mathcal{F}_N(\boldsymbol{x}_0, \tau)$ over the N uniformly subdivided time segments of the time interval, such that $\mathcal{F}_i(\boldsymbol{x}_0, \tau)$ is a *state-wise overapproximation* of $\varphi_{\mathcal{N}}(\boldsymbol{x}_0, (j-1)\delta_c + (i-1)\delta + \tau)$ for $\tau \in [0, \delta_c/N]$, i.e., $\mathcal{F}_j(\boldsymbol{x}_0, \tau)$ contains the exact reachable state from any initial state $\boldsymbol{x}_0$ 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 $\delta = \delta_c/N$ which is called the *flowpipe step or time step*.

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

Notice that $\boldsymbol{x}(0)$ denotes the local initial set for the ODE used in the current control step, while the variables $\boldsymbol{x}_0$ 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.

In the paper, we use Taylor models to represent the flowpipes. Taylor models are originally proposed to compute higher-order overapproximations for the ranges of continuous functions (see Berz & Makino (1998)). They can be viewed as a higher-order extension of intervals Moore et al. (2009) 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 often used as overapproximations for smooth functions. Given a smooth function $f(\boldsymbol{x})$ with $\boldsymbol{x} \in D$, its order k TM overapproximation, or simply TM, can be obtained as $(p_f(\boldsymbol{x}), I_f)$ such that p_f is the order k Taylor approximation of f w.r.t. a point in D, and I_f is a remainder interval which ensures that $\forall \boldsymbol{x} \in D.(f(\boldsymbol{x}) \in p_f(\boldsymbol{x}) + I_f)$. *Since p_f is also over the same variables $\boldsymbol{x}$, the overestimation in a TM can be estimated only on the width of I_f, i.e., if the remainder is zero, then p_f defines the same mapping as f. At this point, a TM serves more like an overapproximate function than just a range overapproximation such as intervals or polyhedra.*

TM arithmetic. TMs are closed under operations such as addition, multiplication, and integration (see Makino & Berz (2003)). 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. For a vector of TMs $((p_1, I_1), \ldots, (p_n, I_n))^T$ such that $p_1, \ldots, p_n$ are over the same variables, we collectively denote it by (p, I) such that p is the polynomial vector $(p_1, \ldots, p_n)^T$ and I is interval vector $(I_1, \ldots, I_n)^T$.

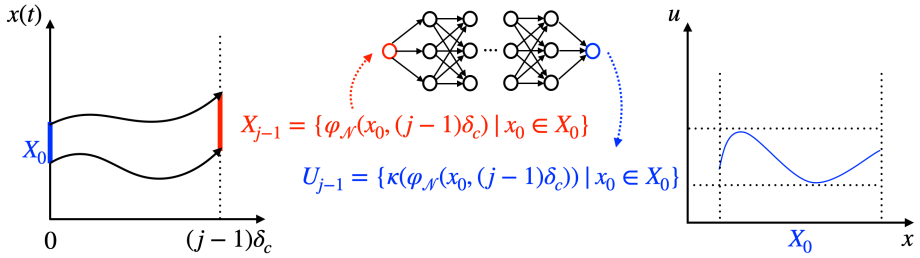


Figure 3: Exact dependency between the initial set and the control input range for a control step. X_{j-1} is the actual reachable set from X_0 at the time $(j-1)\delta_c$, its dependency to X_0 is represented by the flowmap $\varphi_{\mathcal{N}}$, while U_{j-1} is the output range of the NN controller w.r.t. the input range X_{j-1}, its dependency on X_0 is formed via the composition of κ and $\varphi_{\mathcal{N}}$.

As an example, given the TMs $(1-0.5x^2, [-0.1, 0.1])$ and $(x+0.1x^4, [-0.2, 0.2])$ over the domain $x \in [-1, 1]$. The order 4 TM for the sum is $(1+x-0.5x^2+0.1x^4, [-0.3, 0.3])$, and the order 4 TM for the product is $(x-x^3+x^4, [-0.38, 0.38])$.

Although the technique of Taylor model flowpipe construction can be used to generate state-wise overapproximate flowpipes for ODEs (see Berz & Makino (1998); Chen (2015)), it is still a challenge that how to compute and represent the set $\hat{U}_{j-1}$ by a TM which overapproximates the dependency from an initial state in X_0 to the actual control input range $U_{j-1} = \{\kappa(\varphi_{\mathcal{N}}(\boldsymbol{x}_0, (j-1)\delta_c)) \mid \boldsymbol{x}_0 \in X_0\}$ at the beginning of that control step. We illustrate this dependency in Figure 3. Our approach aims at solving this problem.

3 Polynomial Arithmetic for NNCS Verification

In this section, we show how to compute a TM representation for the set $\hat{U}$ to overapproximate the dependency from an initial state in X_0 to the actual control input range U in terms of the neural network controller κ.

Obviously, the key challenges of providing a TM overapproximation $(p_r(\boldsymbol{x}_0), I_r)$ in terms of κ lie on how to properly construct a polynomial p_r to well capture the state-wise dependency of κ, and how to effectively estimate a remainder I_r to provide a sound and tight error bound. While the basic TM arithmetic that works in a layer-by-layer manner for function approximation of a neural network shows a great potential, demonstrated in early works Ivanov et al. (2021b;a), it still suffers from two main challenges in terms of the construction of p_r and estimation of I_r respectively.

1. **Construction of p_r.** Taylor approximation requires the approximated function to be differentiable, thus excluding the use of common operations such as ReLU and pooling, and
2. **Estimation of I_r.** Interval remainders grows rapidly during layer-by-layer propagation of the TMs, effectively degrading it to an interval analysis.

Our approach, POLAR, focuses exactly on these two challenges by a novel use of *univariate Bernstein polynomial interpolation* (BP) to handle non-differentiable activation functions and *symbolic remainders* to taper the growth of the interval remainders.

Meanwhile, it is worthy noting that Bernstein polynomial interpolation can also help improve the remainder estimation, as it is always more accurate than Taylor expansion at the same order for a function. Thus, we use univariate Bernstein polynomial interpolation for *all the activation functions*, not limited to non-differentiable ReLU.

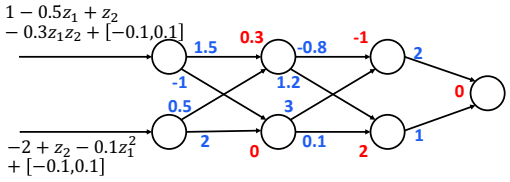


Figure 4: A running example: the numbers in blue are weights, and the numbers in red are biases.

Algorithm 1 Polynomial arithmetic for overapproximating neural network outputs

Input: Input TM $(p(\boldsymbol{x}_0), I)$ with $\boldsymbol{x}_0 \in X_0$, 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 $(p_r(\boldsymbol{x}_0), I_r)$ that contains the set $\kappa((p(\boldsymbol{x}_0), I))$.

1: $(p_r, I_r) \leftarrow (p, I)$;
2: **for** $i = 1$ to $M+1$ **do**
3: $\quad (p_t, I_t) \leftarrow W_i \cdot (p_r, I_r) + B_i$; # Using TM arithmetic
4: $\quad$ Computing an order k Bernstein polynomial p_σ for the activation function over the range defined by (p_t, I_t), i.e., $p_\sigma(\boldsymbol{y}) \approx \sigma_i(\boldsymbol{y})$ with $\boldsymbol{y} \in (p_t, I_t)$;
5: $\quad$ Evaluating a safe remainder I_σ for the polynomial p_σ such that $\forall \boldsymbol{y} \in (p_t, I_t).(\sigma_i(\boldsymbol{y}) \in p_\sigma(\boldsymbol{y}) + I_\sigma)$ holds;
6: $\quad (p_r, I_r) \leftarrow p_\sigma(p_t + I_t) + I_\sigma$; # Using TM arithmetic
7: **end for**
8: **return** (p_r, I_r).

We use the following running example to demonstrate the effectiveness of POLAR. We show the result of using basic TM arithmetic here, and in each of the following sections, we will demonstrate the improvements from using Bernstein polynomials and symbolic remainders respectively.

Example 1 (Running example with basic TM arithmetic) *We demonstrate the computation of a TM output for a small neural network shown in Figure 4. The input is a vector of two TMs over* $\boldsymbol{x}_0 = (z_1, z_2)^T$ *which denotes the initial state in a 2-D state space. The neural network has* 2 *inputs,* 1 *output and* 2 *hidden layers each of which has* 2 *neurons. The activation functions of all the hidden layers and output layer are sigmoid. In the figure, the TMs* $1 - 0.5z_1 + z_2 - 0.3z_1z_2 + [-0.1, 0.1]$ *and* $-2 + z_2 - 0.1z_1^2 + [-0.2, 0.2]$ *are the inputs, and* $z_1, z_2 \in [-1, 1]$. *We hope to obtain an order 2 TM for the output. Note that, with the basic TM arithmetic, we cannot obtain the output TM, as the remainders for the first hidden layer already explode to the order of* $1e23$ *due to the coarse Taylor approximations for the neural network.*

3.1 Polynomial Arithmetic Integrated with Bernstein Polynomial

Given a TM input for a neural network, our approach combines the use of TM arithmetic and Bernstein polynomial interpolation to produce a state-wise overapproximation set for the output of a neural network. The high-level steps are described in Algorithm 1, in which we assume that the neural network has M hidden layers, the neurons in the same layer have the same type of activation functions, and the output layer's neurons also have activation functions. We collectively use $\sigma(\boldsymbol{y})$ to denote applying the activation function σ to each element of $\boldsymbol{y}$.

In Algorithm 1, the input TM is propagated layer-wisely. While the linear transformation in terms of the layer weight and bias (line 3) and the TM composition (line 6) can be calculated by TM arithmetic, the key step is **how to point-wisely approximate activation function by Bernstein polynomial** (line 4) and **how to evaluate a sound remainder efficiently** (line 5).

Computing the Bernstein polynomial p_σ (line 4). Given the computed TM (p_t, I_t) in Line 3, p_σ is a vector of univariate Bernstein polynomials Lorentz (2013) each component of which is expressed in a variable y_j which denotes the range of the j-th component (dimension) of (p_t, I_t). Then the j-th component of p_σ is computed as $p_\sigma^j(y_j) = \sum_{s=0}^{k} \left(\sigma(\frac{\bar{Y}_j - \underline{Y}_j}{k} s + \underline{Y}_j) \binom{k}{s} \frac{(y_j - \underline{Y}_j)^s (\bar{Y}_j - y_j)^{k-s}}{(\bar{Y}_j - \underline{Y}_j)^k} \right)$ which is an approximation of the $\sigma(y_j)$ such that $\bar{Y}_j$ and $\underline{Y}_j$ denotes the upper and lower bounds respectively of the range in the j-th dimension in (p_t, I_t), and they can be obtained by interval evaluating the TM.

Evaluating a remainder I_σ for p_σ (line 5). The remainder I_σ is a vector of intervals, each component of which is a conservative remainder for the corresponding Bernstein polynomial in p_σ such that the approximation error for $\sigma((p_t, I_t))$ is contained. To do so, the j-th interval in I_σ is evaluated as

$[-\epsilon_j, \epsilon_j]$ such that

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

wherein m is the number of samples which are uniformly selected to estimate the remainder. The soundness of the error bound estimation above has been proven in Huang et al. (2019) 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. We also provide a detailed proof in the appendix.

Advantages of Bernstein polynomials. We briefly discuss the advantages of using BP in the approximation of activation functions. (i) Taylor approximation requires the approximated function to be differentiable, however BP approximation only requires the function to be continuous. This fact makes Taylor approximation not applicable to ReLU functions. (ii) BP approximations are essentially polynomial interpolations which are always more accurate approximation forms than Taylor expansion at the same order for the a function (see Phillips (2003)). Thus, we apply univariate BP for general activation functions, rather than just for ReLU (iii) Unlike using multivariate Bernstein polynomials in Huang et al. (2019), our framework only computes univariate Bernstein polynomials such that the complexity is always linear in the approximation order k, and the remainder is much easier to be evaluated. In contrast, the size of a multivariate Bernstein polynomial grows exponentially in k.

Example 2 (Running example with Bernstein polynomial) *We can obtain the output TM of Example 1 with BP (see the supplementary document), where the remainder is* $[-0.132, 0.132]$.

It is important that the polynomial arithmetic algorithm 1 containing the univariate BP can provide a sound state-wise approximation of a neural network, that leads to the soundness of overall flowpipe construction. Specifically, if the input TM (p, I) is a state-wise overapproximation for the reachable state at the beginning of i-th control step, i.e., $p(\boldsymbol{x}_0) + I$ contains the reachable state at the time $t = (i-1)\delta_c$ from $\boldsymbol{x}_0$ for all $\boldsymbol{x}_0 \in X_0$, then the returned TM $p_r(\boldsymbol{x}_0) + I_r$ by Algorithm 1 contains the exact i-th control input in the execution starting from $\boldsymbol{x}_0$ for all $\boldsymbol{x}_0 \in X_0$, i.e., the output TM (p_r, I_r) is a state-wise overapproximation of the control input. Hence, we have the important property given in Theorem 1, and then the set $\mathcal{R}$ computed by flowpipe construction (Section 2) is an overapproximation of the system reachable set in K control steps. A detailed proof can be found in the appendix.

Theorem 1 *If $(p(\boldsymbol{x}_0, t), I)$ is the i-th flowpipe computed in the j-th control step, then for any initial state $\boldsymbol{x}_0 \in X_0$, the box $(p(\boldsymbol{x}_0, t), I)$ is guaranteed to contain the reachable state $\varphi_{\mathcal{N}}(\boldsymbol{x}_0, (j-1)\delta_c + (i-1)\delta + t)$, i.e., the TM flowpipes computed by our approach are state-wise overapproximations for the reachable sets.*

3.2 Symbolic Remainders in Neural Network Analysis

The layer-by-layer propagation across the neural network in Algorithm 1 requires a sequence of consecutive compositions of TMs (line 3 and line 6 for each layer), then the TM remainders may potentially be greatly enlarged by these operations, since they are evaluated by pure interval arithmetic. For example, we want to obtain a TM for the polynomial $p(\boldsymbol{y}) = c + A\boldsymbol{y} + p_h(\boldsymbol{y})$ with $\boldsymbol{y} = (q(\boldsymbol{x}_0), I)$, such that $c + A\boldsymbol{y}$ is the constant and linear part of p, and p_h is the higher-order part. The result can be obtained by evaluating $c + A \cdot q(\boldsymbol{x}_0) + A \cdot I + p_h(q(\boldsymbol{x}_0) + I)$ by TM arithmetic such that the term $A \cdot I$ will be wrapped by its interval evaluation which introduces extra overestimation. During the consecutive compositions of TMs, that overestimation will accumulate, and such phenomenon is called *wrapping effect* Jaulin et al. (2001). To further reduce the overestimation in our TMs, we propose to adapt the symbolic remainder technique, which was previously proposed in Chen & Sankaranarayanan (2016) for ODEs, to avoid the wrapping effect in the layer-by-layer propagation for NNs under linear mappings. The key idea is that we do not compute the interval of $A \cdot I$ out when propagating the TMs to the next layer, but keeps its linear transformation matrix A. When the set is involved in a later linear mapping $B \cdot A \cdot I$, we only composing the matrices and keep the set as $(B \cdot A) \cdot I$. Such a technique requires us to use extra lists to keep the linear transformation matrices. The detailed method is given below.

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

1: $(p_r, I_r) \leftarrow (p, I)$;
2: Setting $\mathcal{Q}$ as an empty array which can keep $M+1$ matrices;
3: Setting $\mathcal{J}$ as an empty array which can keep $M+1$ multidimensional intervals;
4: **for** $i = 1$ to $M+1$ **do**
5: Computing the composite function q_i and the remainder interval I_{σ_i} using Bernstein overapproximation and TM arithmetic (similar to Line 3 to 5 in Algorithm 1);
6: Constructing Q_i using the coefficients of the linear terms in q_i;
7: Computing q_i^R from q_i by eliminating the linear terms;
8: $(\phi_i, J_i) \leftarrow q_i^R(p_r + I_r) + I_{\sigma_i}$; # Using TM arithmetic
9: $\mathbb{J} \leftarrow J_i$;
10: **for** $j = 1$ to $i-1$ **do**
11: $\mathcal{Q}[j] \leftarrow Q_i \cdot \mathcal{Q}[j]$;
12: **end for**
13: Adding Q_i to $\mathcal{Q}$ as the last element;
14: **for** $j = 2$ to i **do**
15: $\mathbb{J} \leftarrow \mathbb{J} + \mathcal{Q}[j] \cdot \mathcal{J}[j-1]$;
16: **end for**
17: Adding J_i to $\mathcal{J}$ as the last element;
18: Computing (p_r, I_r) according to (1); # Using TM arithmetic
19: **end for**
20: **return** $\mathcal{R}$.

Specifically, we denote the resulting TM (p_r, I_r) and the Bernstein overapproximation (p_σ, I_σ) in the i-th iteration (layer) in Algorithm 1 by $(p_{r,i}, I_{r,i})$ and $(p_{\sigma_i}, I_{\sigma_i})$ respectively. Then $(p_{r,i}(\boldsymbol{x}_0), I_{r,i})$ for $i = 1, \ldots, M+1$ is computed as $p_{r,i}(\boldsymbol{x}_0) + I_{r,i} = p_{\sigma_i}(W_i \cdot (p_{r,i-1}(\boldsymbol{x}_0) + I_{r,i-1}) + I_{\sigma_i}$ with $(p_{r,0}(\boldsymbol{x}_0), I_{r,0}) = (p(\boldsymbol{x}_0), I)$ which is the input TM. If we use $q_i(\boldsymbol{y})$ to denote the function $p_{\sigma_i}(W_i \cdot \boldsymbol{y})$, the above expression can be simplified to $p_{r,i}(\boldsymbol{x}_0) + I_{r,i} = q_i(p_{r,i-1}(\boldsymbol{x}_0) + I_{r,i-1}) + I_{\sigma_i}$. By decomposing the linear and the remaining part in q_i, the function can be expressed as $q_i(\boldsymbol{y}) = Q_i\boldsymbol{y} + q_i^R(\boldsymbol{y})$ such that Q_i is the constant matrix consists of the coefficients of the linear terms in q_i, and q_i^R is the nonlinear and the constant part in q_i. Hence, the output TM can be further expressed as

$$p_{r,i}(\boldsymbol{x}_0) + I_{r,i} = Q_i(p_{r,i-1}(\boldsymbol{x}_0) + I_{r,i-1}) + \underbrace{q_i^R(p_{r,i-1}(\boldsymbol{x}_0) + I_{r,i-1}) + I_{\sigma_i}}_{\phi_i(\boldsymbol{x}_0) + J_i}$$

and then by unfolding the recurrence relation, we have

$$\begin{aligned} p_{r,i}(\boldsymbol{x}_0) + I_{r,i} &= Q_i(p_{r,i-1}(\boldsymbol{x}_0) + I_{r,i-1}) + \phi_i(\boldsymbol{x}_0) + J_i \\ &= Q_i(Q_{i-1}(p_{r,i-2}(\boldsymbol{x}_0) + I_{r,i-2}) + \phi_{i-1}(\boldsymbol{x}_0) + J_{i-1}) + \phi_i(\boldsymbol{x}_0) + J_i \\ &= Q_i \cdot \cdots \cdot Q_1 \cdot p(\boldsymbol{x}_0) + Q_i \cdot \cdots \cdot Q_1 \cdot I + \Phi_i(\boldsymbol{x}_0) + \mathbb{J}_i \end{aligned} \tag{1}$$

wherein $\Phi_i = \phi_i(\boldsymbol{x}_0) + Q_i \cdot \phi_{i-1}(\boldsymbol{x}_0) + \cdots + Q_i \cdot \cdots \cdot Q_2 \cdot \phi_1(\boldsymbol{x}_0)$, and $\mathbb{J}_i = J_i + Q_i \cdot J_{i-1} + \cdots + Q_i \cdot \cdots \cdot Q_2 \cdot J_1$. Hence, the term $Q_i \cdot \cdots \cdot Q_1 \cdot I$ can be evaluated without wrapping effect, and if we keep an array for J_j, the computation of $\mathbb{J}_i$ is also free from wrapping effect. Algorithm 2 shows the improvement of Algorithm 1 in accuracy using the symbolic remainder method such that we additionally use two arrays: $\mathcal{Q}[j]$ is $Q_i \cdot \cdots \cdot Q_j$ and $\mathcal{J}[j]$ is J_i for $1 \leq j \leq i$.

Example 3 (Running example with symbolic remainder) *The remainder of the output TM of Example 1 with our symbolic remainder technique is further reduced to* $[-0.069, 0.069]$*, which is more than* 40% *improvement compared to the result without symbolic remainder, i.e.* $[-0.132, 0.132]$.

Time and space complexity. Although Algorithm 2 always produces a TM with a smaller remainder than Algorithm 1 because of the symbolic treatment of the remainder intervals I and J_i under linear mappings, it requires (1) two extra arrays to keep the matrices $Q_M, Q_M \cdot Q_{M-1}, \ldots, Q_M \cdot \cdots \cdot Q_1$, and the remainder $J_{M+1}, \ldots, J_1$, (2) two extra inner loops which perform $i-1$ and $i-2$ iterations in the i-th outer iteration. The size of $Q_i \cdot \cdots \cdot Q_j$ is determined by the rows in W_i and the columns in W_j, and hence the maximum number of neurons in a layer determines the maximum size of the

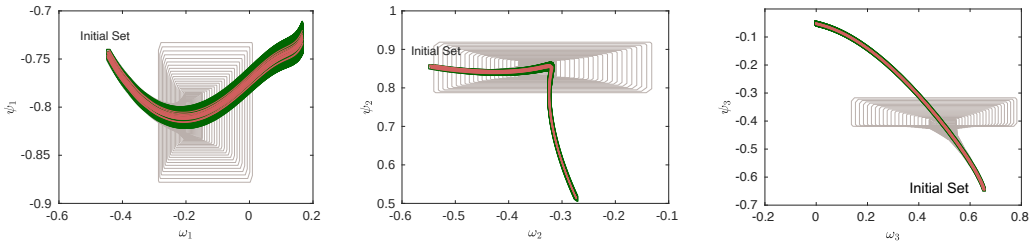


Figure 5: Comparison between reachable sets of the 6-dimensional attitude control benchmark produced by POLAR (green) and Verisig 2.0 (gray). The red curves are simulated trajectories.

matrices in 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 first present an illustrating example of attitude control with 6 state variables and 3 control inputs. We then present a comprehensive comparison to the state-of-the-art tools over the full benchmarks in the related work Huang et al. (2019). Finally, we remark on the observed limitations of our approach. All our experiments were run on a machine with 6-core 2.90 GHz Intel Core i5 and 8GB of RAM.

Illustrating example: attitude control. We consider the attitude control of a rigid body with six states and three inputs as a physically illustrating example Prajna et al. (2004). The complexity of this example lies in the combination of the numbers of the state variables and control inputs. 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 $\boldsymbol{x}{=}(\omega, \psi)$ 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 $\boldsymbol{u}{\in}\mathbb{R}^3$ is updated every 0.1 seconds by a neural network with 3 hidden layers each of which has 64 neurons. The activation of the hidden layers are sigmoid and identity respectively. The initial state set is: $\omega_1{\in}[-0.45, -0.44], \omega_2{\in}[-0.55, -0.54], \omega_3{\in}[0.65, 0.66], \psi_1{\in}[-0.75, -0.74], \psi_2 \in [0.85, 0.86], \psi_3 \in [-0.65, -0.64]$. POLAR computed the TM flowpipes for 30 control steps in **201** seconds. Figure 5 shows the plot of the octagonal enclosures of the flowpipes. We can observe that the flowpipes computed by POLAR are tight w.r.t. the simulated traces. As a comparison, although Verisig 2.0 Ivanov et al. (2021a) can handle this system in theory, its remainder explodes very quickly and the tool crashes after only a few steps.

Comparison to state-of-the-arts. We compare POLAR to the state-of-the-art tools on the full benchmarks in Huang et al. (2019), including Sherlock Dutta et al. (2019) (only for ReLU) , Verisig 2.0 Ivanov et al. (2021a) (only for sigmoid and tanh), NNV Tran et al. (2020), and ReachNN*Fan et al. (2020)[1]. We refer to Huang et al. (2019); Ivanov et al. (2021a) for the details of the 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 Ivanov et al. (2021a). We can see that POLAR successfully verifies almost all cases within seconds, which is on average 23x and up to 353x faster compared to the tool with the second best efficiency, and achieves the best performance among all the tools (detailed

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

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* Fan et al. (2020)	Sherlock Dutta et al. (2019)	Verisig 2.0 Ivanov et al. (2021a)
		σ	M	n				
1	2	ReLU	3	20	**22**	26	42	–
		sigmoid	3	20	**20**	75	–	47
		tanh	3	20	**18**	Unknown	–	46
		ReLU+tanh	3	20	**11**	71	–	–
2	2	ReLU	3	20	**2**	5	3	–
		sigmoid	3	20	Unknown	13	–	**7**
		tanh	3	20	**3**	73	–	Unknown
		ReLU+tanh	3	20	**2**	Unknown	–	–
3	2	ReLU	3	20	**13**	94	143	–
		sigmoid	3	20	**24**	146	–	44
		tanh	3	20	**22**	137	–	38
		ReLU+sigmoid	3	20	**14**	150	–	–
4	3	ReLU	3	20	**1**	8	21	–
		sigmoid	3	20	**3**	22	–	11
		tanh	3	20	**3**	21	–	10
		ReLU+tanh	3	20	**2**	12	–	–
5	3	ReLU	4	100	**7**	103	15	–
		sigmoid	4	100	**15**	27	–	190
		tanh	4	100	**16**	Unknown	–	179
		ReLU+tanh	4	100	**6**	Unknown	–	–
6	4	ReLU	4	20	**4**	1130	35	–
		sigmoid	4	20	**6**	13350	–	83
		tanh	4	20	**6**	2416	–	70
		ReLU+tanh	4	20	**4**	1413	–	–

comparisons on the tightness of reachable sets can be found in the Appendix). In addition, POLAR scales better with the size of the neural network controller compared to ReachNN* and Verisig 2.0. This demonstrates the potential of handling larger scale systems such as the attitude control example. Note that the only exception is $\#2$ with sigmoid NN controller. We discuss this in detail below.

Limitations. POLAR did not prove the property on test # 2 with sigmoid activation but Verisig 2.0 and ReachNN* did. The main reason is the combined use of Bernstein and Taylor approximations. Although a lower-order Taylor approximation can be derived from a higher-order one by truncating the higher-order terms, it is not the case for a Bernstein approximation since the term coefficients often need to be recomputed if the order is changed. Thus, simply using truncation to simplify a TM may produce a large remainder. Secondly, TM-based techniques can be sensitive to the choice of TM settings (stepsizes, orders, etc.). In our experiments, we use the same TM settings for all the tools.

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 show that POLAR performs significantly better than state-of-the-art techniques in terms of both computation efficiency and tightness of reachable set estimation. Future work includes automatic tuning of the hyper-parameters in POLAR to further improve performance and accessibility.

Reproducibility Statement. The implementation code can be found in our attached attached zip package, and we will also release it on GitHub once published. In addition, we also provide proofs of two important propositions, the TM outputs of Example 2 and Example 3, and the additional experimental results that include the visualization of the reachable set computed by different approaches in the Appendix.

REFERENCES

Matthias Althoff. An introduction to CORA 2015. In *International Workshop on Applied veRification for Continuous and Hybrid Systems (ARCH)*, volume 34 of *EPiC Series in Computing*, pp. 120–151, 2015. 1

Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2): 183–235, 1994. 1, 3

Martin Berz and Kyoko Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable computing*, 4:361–369, 1998. 3, 4

Xin Chen. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. PhD thesis, RWTH Aachen University, 2015. 4

Xin Chen and Sriram Sankaranarayanan. Decomposed reachability analysis for nonlinear systems. In *Proc. of RTSS'16*, pp. 13–24, 2016. 6

Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pp. 258–263, 2013. 1

Tommaso Dreossi, Thao Dang, and Carla Piazza. Parallelotope bundles for polynomial reachability. In *Proceedings of ACM International Conference on Hybrid Systems: Computation and Control (HSCC)*, pp. 297–306. ACM, 2016. 1

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *Proc. of NFM'18*, volume 10811 of *LNCS*, pp. 121–138. Springer, 2018. 1

Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proc. of HSCC'19*, pp. 157–168. ACM, 2019. 1, 2, 8, 9, 13, 14

Jiameng Fan, Chao Huang, Xin Chen, Wenchao Li, and Qi Zhu. ReachNN*: A tool for reachability analysis of neural-network controlled systems. In *Proceedings of International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 12302 of *LNCS*, pp. 537–542. Springer, 2020. 2, 8, 9, 13, 14

Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pp. 379–395, 2011. 1

Chao Huang, Xin Chen, Wang Lin, Zhengfeng Yang, and Xuandong Li. Probabilistic safety verification of stochastic hybrid systems using barrier certificates. *ACM Trans. Embed. Comput. Syst.*, 16 (5s):186, 2017a. 1

Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. ReachNN: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst.*, 18(5s):106:1–106:22, 2019. 1, 2, 6, 8, 13, 14

Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Proc. of CAV'17*, volume 10426 of *LNCS*, pp. 3–29. Springer, 2017b. 1

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proc. of HSCC'18*, pp. 169–178. ACM, 2019. 2

Radoslav Ivanov, Taylor Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig 2.0: Verification of neural network controllers using taylor model preconditioning. In *Proc. of CAV'21*, volume 12759 of *LNCS*, pp. 249–262. Springer, 2021a. 2, 4, 8, 9

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 Trans. Embed. Comput. Syst.*, 20(1):7:1–7:26, 2021b. 2, 4, 13, 14

Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. Interval analysis. In *Applied Interval Analysis*. Springer, 2001. 2, 6

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Proc. of CAV'17*, volume 10426 of *LNCS*, pp. 97–117. Springer, 2017. 1

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR'16 (Poster)*, 2016. 1

George G. Lorentz. *Bernstein Polynomials*. American Mathematical Society, 2013. 5

John Lygeros, Claire J. Tomlin, and Shankar Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999. 1

Kyoko Makino and Martin Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 4(4):379–456, 2003. 3

James D. Meiss. *Differential Dynamical Systems*. SIAM publishers, 2007. 2

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 1

R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. 3

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Proc. of RSS'18*, 2018. 1

George M. Phillips. *Interpolation and Approximation by Polynomials*. Springer Science & Business Media, 2003. 6

Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pp. 477–492. Springer, 2004. 1

Stephen Prajna, Pablo A Parrilo, and Anders Rantzer. Nonlinear control synthesis by convex optimization. *IEEE Transactions on Automatic Control*, 49(2):310–314, 2004. 8

Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin T. Vechev. Beyond the single neuron convex barrier for neural network certification. In *Proc. of NeurIPS'19*, pp. 15072–15083, 2019. 1

Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *Proc. of CAV'20*, volume 12224 of *LNCS*, pp. 3–17. Springer, 2020. 8, 13, 14

Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *Proc. of USENIX Security (USENIX)*, pp. 1599–1614, 2018. 1

Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning (ICML)*, 2018. 1

Zhengfeng Yang, Chao Huang, Xin Chen, Wang Lin, and Zhiming Liu. A linear programming relaxation based approach for generating barrier certificates of hybrid systems. In *Formal Methods*, pp. 721–738. Springer, 2016. 1

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. 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. 1

A TM OUTPUTS OF THE RUNNING EXAMPLE

The result with Bernstein polynomial:

$$(7.920880969687617e-01)+(-7.738563820601800e-03*z_2)+$$
$$(4.240508129753556e-03*z_1)+(1.541011940991888e-03*z_2^2)+$$
$$(2.115950977442756e-03*z_1*z_2)+(3.129465795158815e-04*z_1^2)+$$
$$[-1.316162684352782e-01, 1.316162684352782e-01]$$

The result with symbolic remainder:

$$(7.879035543566243e-01)+(-7.425376458676299e-03*z_2)+$$
$$(4.133105242196053e-03*z_1)+(-7.124243361287374e-05*z_2^2)+$$
$$(7.930977707887526e-05*z_1*z_2)+(-2.207266239135132e-05*z_1^2)+$$
$$[-6.906656051103792e-02, 6.906656051103792e-02]$$

B THEOREM PROOF

B.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 $[\frac{\overline{Y}_j-\underline{Y}_j}{m}(i-1)+\underline{Y}_j, \frac{\overline{Y}_j-\underline{Y}_j}{m}(i)+\underline{Y}_j]$, and let $c=\frac{\overline{Y}_j-\underline{Y}_j}{m}(i-\frac{1}{2})+\underline{Y}_j$ 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 $\frac{\overline{Y}_j-\underline{Y}_j}{m}$, i.e., $L_j\cdot\frac{\overline{Y}_j-\underline{Y}_j}{m}$. The detailed deduction is given below.

$$\begin{aligned}
&|p_\sigma^j(x)-\sigma_j(x)|\\
=&|p_\sigma^j(x)-p_\sigma^j(c)+p_\sigma^j(c)-\sigma_j(c)+\sigma_j(c)-\sigma_j(x)|\\
\le&|p_\sigma^j(x)-p_\sigma^j(c)|+|p_\sigma^j(c)-\sigma_j(c)|+|\sigma_j(c)-\sigma_j(x)| && \text{Triangle inequality}\\
\le&|p_\sigma^j(x)-p_\sigma^j(c)|+|p_\sigma^j(c)-\sigma_j(c)|+L_j\cdot\frac{\overline{Y}_j-\underline{Y}_j}{2m} && \text{Def. of Lipschitz continuity for } \sigma_j\\
\le&L_j\cdot\frac{\overline{Y}_j-\underline{Y}_j}{2m}+|p_\sigma^j(c)-\sigma_j(c)|+L_j\cdot\frac{\overline{Y}_j-\underline{Y}_j}{2m} && \text{Def. of Lipschitz continuity for } p_\sigma^j \text{ [a]}\\
=&|p_\sigma^j(c)-\sigma_j(c)|+L_j\frac{\overline{Y}_j-\underline{Y}_j}{m}
\end{aligned}$$

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

B.2 PROOF OF THEOREM 1

Proof. First, any of our Bernstein overapproximation $p_\sigma+I_\sigma$ is an input-wise overapproximation for the (component-wise) activation function σ in a layer, i.e., for any input $\boldsymbol{y}$ in the range that I_σ is evaluated, we have that $\sigma(\boldsymbol{y})\in p_\sigma(\boldsymbol{y})+I_\sigma$. Therefore, the Taylor model (TM) output (p_r, I_r) of Algorithm 2 and 3 is a state-wise overapproximation w.r.t. the TM variable $\boldsymbol{z}$, which is the variable representing the initial state and ranging in the system initial set X_0.

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 $j=0$, the TM flowpipes are computed for the reachable set in the first control step and the evolution is under the pure continuous dynamics $\dot{\boldsymbol{x}}=f(\boldsymbol{x},\boldsymbol{u}_0)$, $\dot{\boldsymbol{u}}=0$ with $\boldsymbol{x}(0)\in X_0$ and

$\boldsymbol{u}(0) = \kappa(\boldsymbol{x}(0))$. The range of $\boldsymbol{u}(0)$, that is $\{\kappa(\boldsymbol{x}(0)) \mid \boldsymbol{x}(0) \in X_0\}$, is overapproximated by a TM $(p_r(\boldsymbol{x}_0), I_r)$ with $\boldsymbol{z} \in \mathcal{X}_0$ using Algorithm 2 or 3. Hence, by performing TM flowpipe construction for the ODE $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$, $\dot{\boldsymbol{u}} = 0$ with the initial set $\boldsymbol{x}(0) = \boldsymbol{x}_0$, $\boldsymbol{u}(0) \in p_r(\boldsymbol{x}_0) + I_r$ such that $\boldsymbol{z}$ range in X_0, we have that for any $i = 1, \ldots, N$, the i-th TM flowpipe $(p_i(\boldsymbol{x}_0, t), I_i)$ contains the exact reachable state at the time $(i-1)\delta + t$. Here, t is the flowpipe local time variable ranging in $[0, \delta]$.

Induction. When $j > 0$, we assume that the local initial set $X_j = (p_0(\boldsymbol{x}_0), I_0)$ is a state-wise overapproximation of the reachable set at the time $j\delta_c$ from any $\boldsymbol{x}_0 \in X_0$. Then by Algorithm 2 or 3, the obtained TM $(p_r(\boldsymbol{x}_0), I_r)$ is a state-wise overapproximation for the control input set $\kappa(X_j)$, i.e., the control output generated by the neural network controller based on the $j\delta_c$-time state in the execution from any initial state $\boldsymbol{x}_0 \in X_0$ is contained in the box $p_r(\boldsymbol{x}_0) + I_r$. Hence, for any $i = 1, \ldots, N$, the i-th flowpipe $(p_i(\boldsymbol{x}_0, t), I_i)$ computed for the ODE $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$, $\dot{\boldsymbol{u}} = 0$ with the initial set $\boldsymbol{x}(0) \in \mathcal{X}_j$, $\boldsymbol{u}(0) \in p_r(\boldsymbol{x}_0) + I_r$ is a state-wise overapproximation for the reachable set in the time interval of $[j\delta_c + (i-1)\delta, j\delta_c + i\delta]$. More precisely, the reachable state at the time $j\delta_c + (i-1)\delta + t$ for any $t \in [0, \delta]$ is contained in the box $p_i(\boldsymbol{x}_0, t) + I_i$. □

C Additional Experimental Results

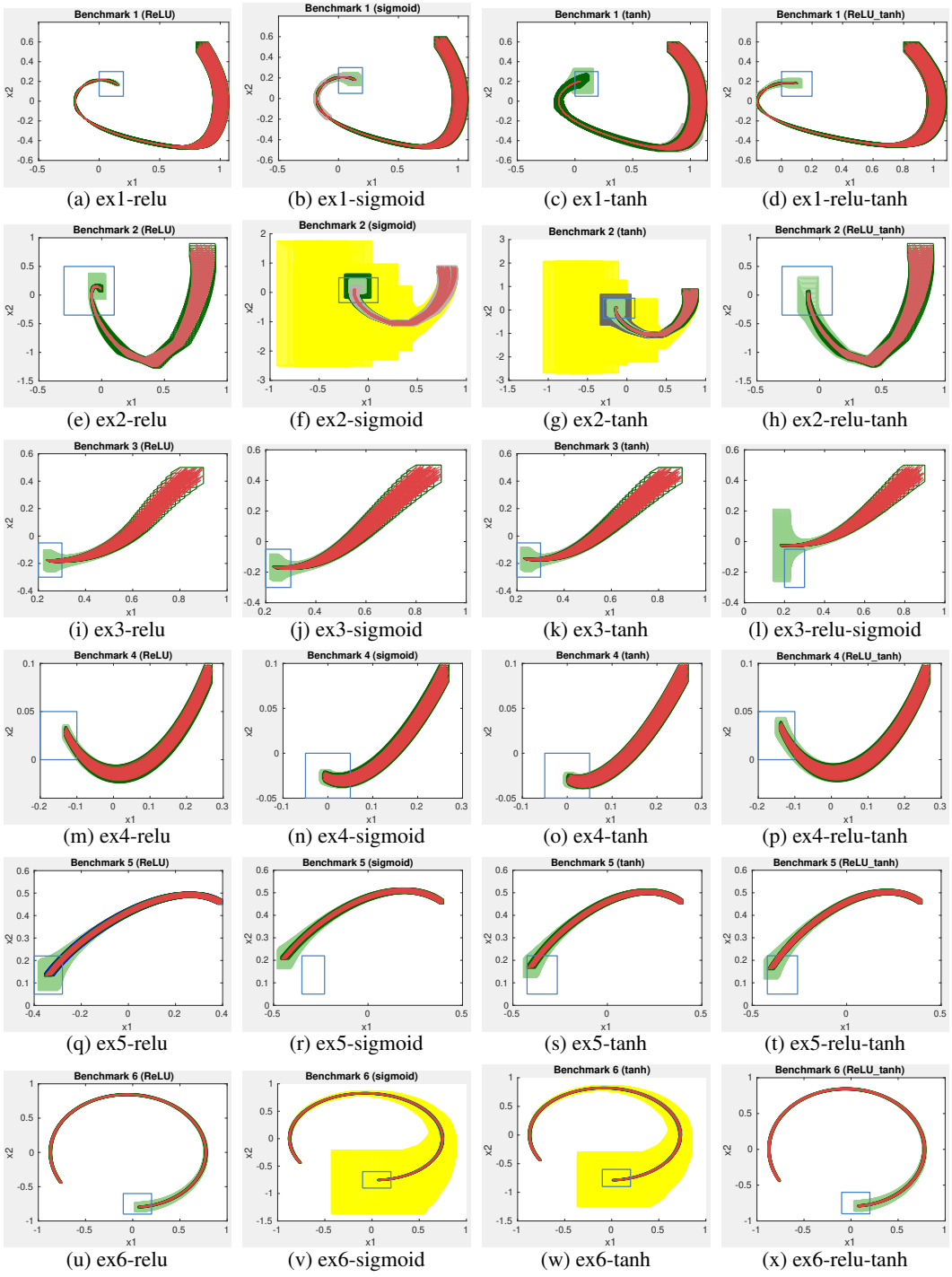
Here, we present additional plots of reachable sets computed by different techniques for the benchmarks in Section 4 of the main paper.

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.

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* Huang et al. (2019); Fan et al. (2020). The blue sets are the flowpipes computed by Sherlock Dutta et al. (2019). The grey sets are the flowpipes computed by Verisig 2.0 Ivanov et al. (2021b). 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 Tran et al. (2020) 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 Ivanov et al. (2021b) where NNV is not able to verify any of these benchmarks. The blue box represents the target set in each test. Except for ex2-sigmoid, POLAR produces the tightest reachable set estimation and successfully proves or disproves the reachability property for all the examples.

D Implementation

The source code can be found in the attached zip. Please follow the instructions described in README.md to run the examples shown in the paper.



(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

Figure 6: 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* Huang et al. (2019); Fan et al. (2020) (light green sets), Sherlock Dutta et al. (2019) (blue sets), Verisig 2.0 Ivanov et al. (2021b) (grey sets), and NNV Tran et al. (2020) (yellow sets).