# PolyARBerNN: A Neural Network Guided Solver and Optimizer for Bounded Polynomial Inequalities

WAEL FATNASSI and YASSER SHOUKRY, University of California, Irvine, USA

Constraints solvers play a significant role in the analysis, synthesis, and formal verification of complex cyber-physical systems. In this article, we study the problem of designing a scalable constraints solver for an important class of constraints named polynomial constraint inequalities (also known as nonlinear real arithmetic theory). In this article, we introduce a solver named PolyARBerNN that uses convex polynomials as abstractions for highly nonlinears polynomials. Such abstractions were previously shown to be powerful to prune the search space and restrict the usage of sound and complete solvers to small search spaces. Compared with the previous efforts on using convex abstractions, PolyARBerNN provides three main contributions namely (i) a neural network guided abstraction refinement procedure that helps selecting the right abstraction out of a set of pre-defined abstractions, (ii) a Bernstein polynomial-based search space pruning mechanism that can be used to compute tight estimates of the polynomial maximum and minimum values which can be used as an additional abstraction of the polynomials, and (iii) an optimizer that transforms polynomial objective functions into polynomial constraints (on the gradient of the objective function) whose solutions are guaranteed to be close to the global optima. These enhancements together allowed the PolyARBerNN solver to solve complex instances and scales more favorably compared to the state-of-the-art nonlinear real arithmetic solvers while maintaining the soundness and completeness of the resulting solver. In particular, our test benches show that PolyARBerNN achieved 100X speedup compared with Z3 8.9, Yices 2.6, and PVS (a solver that uses Bernstein expansion to solve multivariate polynomial constraints) on a variety of standard test benches. Finally, we implemented an optimizer called PolyAROpt that uses PolyARBerNN to solve constrained polynomial optimization problems. Numerical results show that PolyAROpt is able to solve high-dimensional and high order polynomial optimization problems with higher speed compared to the built-in optimizer in the Z3 8.9 solver.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

Additional Key Words and Phrases: Neural networks, bernstein polynomials, abstraction refinement

**ACM Reference Format:**
Wael Fatnassi and Yasser Shoukry. 2024. PolyARBerNN: A Neural Network Guided Solver and Optimizer for Bounded Polynomial Inequalities. *ACM Trans. Embedd. Comput. Syst.* 23, 2, Article 22 (March 2024), 26 pages. https://doi.org/10.1145/3632970

This work was supported by the National Science Foundation under grant numbers #2002405 and #2139781.
Authors' address: W. Fatnassi and Y. Shoukry, University of California, Irvine, CA 92697, USA; e-mails: wfatnass@uci.edu, yshoukry@uci.edu.


ACM 1539-9087/2024/03-ART22
https://doi.org/10.1145/3632970

## 1 INTRODUCTION

Constraint solvers and optimizers have been used heavily in the design, synthesis, and verification of cyber-physical systems [1, 3, 4, 7, 17, 30, 35, 42, 45, 53, 54]. Examples include verification of **neural network** (**NN**) controlled autonomous systems [50], formal verification of human–robot interaction in healthcare scenarios [32], automated synthesis for distributed cyber-physical systems [44], design for cyber-physical systems under sensor attacks [48], air traffic management of unmanned aircraft systems [39], software verification for the next generation space-shuttle [41], and conflict detection for aircraft [40].

In this article, we will focus on the class of general multivariate polynomial constraints (also known as nonlinear real arithmetic). Multivariate polynomial constraints appear naturally in the design, synthesis, and verification of these systems. It is not then surprising that the amount of attention given to this problem in the last decade, as evidenced by the amount of off-the-self solvers that are designed to solve feasibility and optimization problems over general multivariate polynomial constraints, including Z3 [13], Coq [33], Yices [16], PVS [38], Cplex, [34], CVXOPT [52], and Quadprog [26]. Regardless of their prevalence in several synthesis and verification problems, well-known algorithms—that are capable of solving a set of polynomial constraints—are shown to be doubly exponential [18], placing a significant challenge to design efficient solvers for such problems.

Recently, NNs have shown impressive empirical results in approximating unknown functions. This observation motivated several researchers to ask how to use NNs to tame the complexity of NP-hard problems. Examples are the use of NNs to design scalable solvers for program synthesis [14], traveling salesman problem [5], first-order theorem proving [28], higher-order theorem proving [29], and Boolean satisfiability (SAT) problems [46]. While several of these solvers sacrifice either soundness or correctness guarantees, we are interested in this article on using such empirically powerful NNs to design a sound and complete solver for nonlinear real arithmetic.

In addition to NNs, polynomials constitute a rich class of functions for which several approximators have been studied. Two of the most famous approximators for polynomials are Taylor approximation and Bernstein polynomials. These two approximators have been successfully used in solvers like Coq and PVS [8, 38]. This opens the question of how to combine all those approximation techniques, i.e., NNs, Taylor, and Bernstein approximations, to come up with a scalable solver that can reason about general multivariate polynomial constraints. We introduce PolyARBerNN, a novel sound and complete solver for polynomial constraints that combines these three function approximators (NNs, Taylor, and Bernstein) to prune the search space and produce small enough instances in which existing sound and complete solvers (based on the well-known **Cylindrical Algebraic Decomposition** (**CAD**) algorithm) can easily reason about. In general, we provide the following contributions:

— We introduced a novel NN-guided abstraction refinement process in which an NN is used to guide the use of Taylor approximations to find a solution or prune the search space. We analyzed the theoretical characteristics of such an NN and provided empirical evidence on the generalizability of the trained NN in terms of its ability to guide the abstraction refinement process for unseen polynomials with various numbers of variables and orders.
— We complement the NN-guided abstraction refinement with a state-space pruning phase using Bernstein approximations that accelerates the process of removing portions of the state space in which the sign of the polynomial does not change.
— We validated our approach by first comparing the scalability of the proposed PolyARBerNN solver with respect to PVS, a library that uses Bernstein expansion to solve polynomial constraints. Second, we compared the execution times of the proposed tool with the latest

versions of the state-of-the-art nonlinear arithmetic solvers, such as Z3 8.9, Yices 2.6 by varying the order, the number of variables, and the number of the polynomial constraints for instances when a solution exists and when a solution does not exist. We also compared the scalability of the solver against Z3 8.9 and Yices 2.9 on the problem of synthesizing a controller for a cyber-physical system.

— We proposed PolyAROpt, an optimizer that uses PolyARBerNN to solve constrained multivariate polynomial optimization problems. Our theoretical analysis shows that PolyAROpt is capable of providing solutions that are $\epsilon$ close to the global optima (for any $\epsilon > 0$ chosen by the user). Numerical results show that PolyAROpt solves high-dimensional and high-order optimization problems with high speed compared to the built-in optimizer in Z3 8.9 solver. We also validated the effectiveness of PolyAROpt on the problem of computing the reachable sets of polynomial dynamical systems.

**Related work:** CAD was introduced by Collins [12] in 1975 and is considered to be the first algorithm to effectively solve general polynomial inequality constraints. Several improvements were introduced across the years to reduce the high time complexity of the CAD algorithm [9, 27, 36]. Although the CAD algorithm is sound and complete, it scales poorly with the number of polynomial constraints and their order. Other techniques to solve general polynomial inequality constraints include the use of transformations and approximations to scale the computations. For instance, the authors in [38] incorporated Bernstein polynomials in the **Prototype Verification System** (**PVS**) theorem prover; these developments are publicly available in the NASA PVS Library. The library uses the range enclosure propriety of Bernstein polynomials to solve quantified polynomial inequalities. However, the library is not complete for non-strict inequalities [38] and is not practical for higher dimensional polynomials. Another line of work that is related to our work is the use of machine learning to solve combinatorial problems [25, 46]. In particular, the authors in [25] proposed a **graph convolutional neural network** (**GCNN**) to learn heuristics that can accelerate **mixed-integer linear programming** (**MILP**) solvers. Similarly, the NeuroSAT solver [46] uses a **message-passing neural network** (**MPNN**) to solve Boolean SAT problems. The authors of [46] showed that NeuroSAT generalizes to novel distributions after training only on random SAT problems. Nevertheless, NeuroSAT is not competitive with state-of-the-art SAT solvers and it does not have a correctness guarantee.

## 2 PROBLEM FORMULATION

**Notation:** We use the symbols $\mathbb{N}$ and $\mathbb{R}$ to denote the set of natural and real numbers, respectively. We denote by $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ the vector of real-valued variables, where $x_i \in \mathbb{R}$. We denote by $I_n(\underline{d}, \overline{d}) = [\underline{d}_1, \overline{d}_1] \times \cdots \times [\underline{d}_n, \overline{d}_n] \subset \mathbb{R}^n$ the $n$-dimensional hyperrectangle where $\underline{d} = (\underline{d}_1, \ldots, \underline{d}_n)$ and $\overline{d} = (\overline{d}_1, \ldots, \overline{d}_n)$ are the lower and upper bounds of the hyperrectangle, respectively. For a real-valued vector $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and an index-vector $K = (k_1, \ldots, k_n) \in \mathbb{N}^n$, we denote by $x^K \in \mathbb{R}$ the scalar $x^K = x_1^{k_1} \cdots x_n^{k_n}$. Given two multi-indices $K = (k_1, \ldots, k_n) \in \mathbb{N}^n$ and $L = (l_1, \ldots, l_n) \in \mathbb{N}^n$, we use the following notation throughout this article: $K + L = (k_1 + l_1, \ldots, k_n + l_n)$, $\binom{L}{K} = \binom{l_1}{k_1} \cdots \binom{l_n}{k_n}$, and $\sum_{K \leq L} = \sum_{k_1 \leq l_1} \cdots \sum_{k_n \leq l_n}$. A real-valued multivariate polynomial $p : \mathbb{R}^n \to \mathbb{R}$ is defined as follows:

$$p(x_1, \ldots, x_n) = \sum_{k_1=0}^{l_1} \sum_{k_2=0}^{l_2} \cdots \sum_{k_n=0}^{l_n} a_{(k_1,\ldots,k_n)} x_1^{k_1} x_2^{k_2} \ldots x_n^{k_n} = \sum_{K \leq L} a_K x^K,$$

where $L = (l_1, l_2, \ldots, l_n)$ is the maximum degree of $x_i$ for all $i = 1, \ldots, n$. We denote by $a_p = (a_{(0,0,\ldots,0)}, \ldots, a_{(l_1,l_2,\ldots,l_n)})$ the vector of all the coefficients of polynomial $p$. We denote the

space of multivariate polynomials with coefficients in $\mathbb{R}$ by $\mathbb{R}[x_1, x_2, \ldots, x_n]$. Given a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$, we denote by $L_0^-(f)$ and $L_0^+(f)$ the zero sublevel and zero superlevel sets of f, i.e.,:

$$L_0^-(f) = \{x \in \mathbb{R}^n | f(x) \leq 0\}, \qquad L_0^+(f) = \{x \in \mathbb{R}^n | f(x) \geq 0\}.$$

Finally, a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called Lipschitz continuous if there exists a positive real constant $\omega_f \geq 0$ such that, for all $x_1 \in \mathbb{R}^n$ and $x_2 \in \mathbb{R}^n$, the following holds:

$$\|f(x_1) - f(x_2)\| \leq \omega_f \|x_1 - x_2\|$$

**Main Problem:** In this article, we focus on two problems namely (Problem 1) the feasibility problems that involve multiple *polynomial inequality constraints* with input ranges confined within closed hyperrectangles and (Problem 2) the constrained optimization problem which aims to maximize (or minimize) a polynomial objective function subject to other polynomial inequality constraints and input range constraints.

Problem 1.

$$\exists x \in I_n(\underline{d}, \overline{d}) \quad \textit{such that:} \quad \begin{aligned} p_1(x_1, \ldots, x_n) &\leq 0 \\ &\vdots \\ p_m(x_1, \ldots, x_n) &\leq 0 \end{aligned}$$

where $p_i(x) = p_i(x_1, \ldots, x_n) \in \mathbb{R}[x_1, x_2, \ldots, x_n]$ is a polynomial over variables $x_1, \ldots, x_n$. Without loss of generality, $p_i(x) \geq 0$ and $p_i(x) = 0$ can be encoded using the constraints above. Similarly, given a polynomial objective function $p(x) \in \mathbb{R}[x_1, x_2, \ldots, x_n]$, we define the optimization problem as follows:

Problem 2.

$$\min_{x \in I_n(\underline{d}, \overline{d})} p(x) \quad [\textbf{\textit{or}} \quad \max_{x \in I_n(\underline{d}, \overline{d})} p(x)]$$
$$\textit{subject to:} \quad \begin{aligned} p_1(x_1, \ldots, x_n) &\leq 0, \\ &\vdots \\ p_m(x_1, \ldots, x_n) &\leq 0 \end{aligned}$$

## 3 CONVEX ABSTRACTION REFINEMENT: BENEFITS AND DRAWBACKS

In this section, we overview our previously reported framework for using convex abstraction refinement process introduced in [55] along with some drawbacks that motivate the need for the proposed framework.

### 3.1 Overview of Convex Abstraction Refinement

Sound and complete algorithms that solve Problem 1 are known to be doubly exponential in $n$ with a total running time that it is bounded by $(m\ \overline{deg})^{2^n}$ [18], where $\overline{deg}$ is the maximum degree among the polynomials $p_1, \ldots, p_m$. Since the complexity of the problem grows exponentially, it is useful to remove (or prune) subsets of the search space in which the solution is guaranteed not to exist. Since Problem 1 asks for an $x$ in $\mathbb{R}^n$ for which all the polynomials are negative, a solution does not exist in subsets of $\mathbb{R}^n$ at which one of the polynomials $p_i$ is always positive (i.e., $L_0^+(p_i)$). In the same way, finding regions of the input space for which some of the polynomials are negative $L_0^-(p_i)$ helps with finding the solution faster.

To find subsets of $L_0^+(p_i)$ and $L_0^-(p_i)$ efficiently, the use of "convex abstractions" of the polynomials was previously proposed by the authors in [55]. Starting from a polynomial $p_i(x) \in \mathbb{R}[x]$ and a

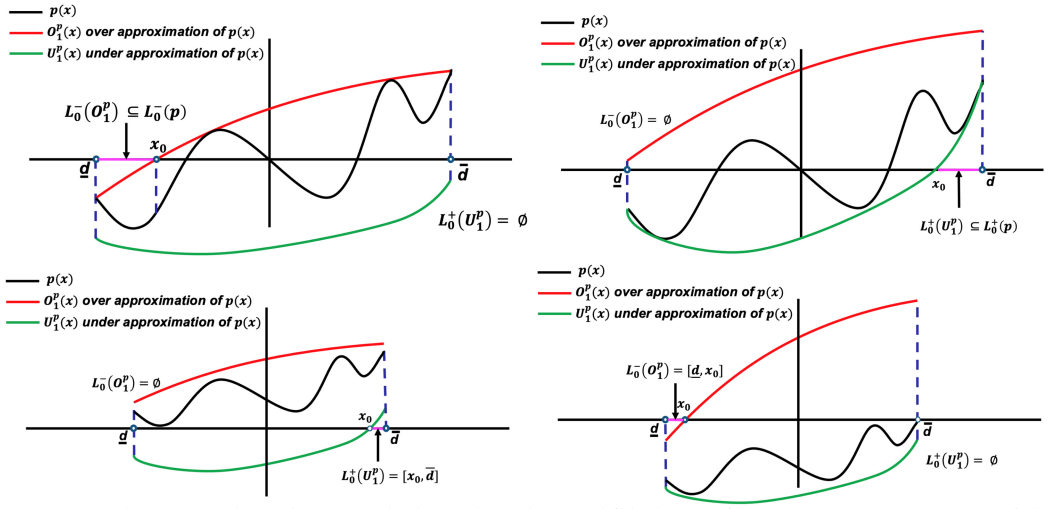


Fig. 1. Exemplary cases where abstracting higher order polynomial (black curves) using convex approximations fails to provide helpful information: **Top-Left:** under-approximation (green curve) is entirely negative and hence fails to identify any subsets of $L_0^+(p)$. **Top-Right:** over-approximation (red curve) is entirely positive and hence fails to identify subsets of $L_0^-(p)$. **Bottom:** under/over approximations failed to identify polynomials that are consistently positive (left) or negative (right).

hyperrectangle $I_n \subset \mathbb{R}^n$, the framework in [55] computes two quadratic polynomials $O_j^{p_i}$ and $U_j^{p_i}$ such that:

$$U_j^{p_i}(x) \leq p(x) \leq O_j^{p_i}(x), \qquad \forall x \in I_n, \tag{1}$$

where $O$ and $U$ stands for Over-approximate and Under-approximate quadratic polynomials, respectively, and the subscript $j$ in $O_j^{p_i}(x)$ and $U_j^{p_i}(x)$ encodes the iteration index of the abstraction refinement process. It is easy to notice that the zero superlevel set of $U_j^{p_i}(x)$ is a subset of $L_0^+(p_i)$, i.e., $L_0^+(U_j^{p_i}) \subseteq L_0^+(p_i)$. Similarly, the zero sublevel set of $O_j^{p_i}(x)$ is a subset of $L_0^-(p_i)$, i.e., $L_0^-(O_j^{p_i}) \subseteq L_0^-(p_i)$. Moreover, being convex polynomials, identifying the zero superlevel sets and zero sublevel sets of $O_j^{p_i}(x)$ and $U_j^{p_i}(x)$ can be computed efficiently using convex programming tools. By iteratively refining these upper and lower convex approximations, the framework in [55] was able to rapidly prune the search space until regions with relatively small volumes are identified, at which sound and complete tools such as Z3 8.9 and Yices 2.6 (which are based on the CAD algorithm) are used to search these small regions, efficiently, to find a solution. It is important to notice that these solvers (especially Yices) are optimized for the cases when the search space is a bounded hyperrectangle.

### 3.2 Drawbacks of Convex Abstraction Refinement

Although the prescribed convex abstraction refinement process was shown to provide several orders of magnitude speedup compared to the state-of-the-art [55], it adds unnecessary overhead in certain situations. In particular, and as shown in Figure 1, the quadratic abstractions $O_j^{p_i}(x)$ and $U_j^{p_i}(x)$ may fail to identify meaningful subsets of $L_0^-(p_i)$ and $L_0^+(p_i)$. One needs to split the input region to tighten the over-/under-approximation in such cases. Indeed, applying the convex abstraction refinement process, above, may lead to several unnecessary over-approximations or

under-approximations until a tight one that prunes the search space is found. These drawbacks call for a methodology that is capable of:

(1) Guiding the abstraction refinement process: To reduce the number of unnecessary computations of over/under approximations, one needs a heuristic that guides the convex abstraction refinement process. In particular, such a heuristic needs to consider the properties of the polynomials and the input region to estimate the volume of the sets that the convex under/over-approximation will identify.
(2) Alternative Abstraction: As shown in Figure 1 (bottom), abstracting high-order polynomials using convex ones may fail to identify easy cases when the polynomial is strictly positive or negative. Therefore, it is beneficial to use alternative ways to abstract high-order polynomials that can augment the convex abstractions.

Designing a strategy that addresses the two requirements above is the main topic for the following two sections.

## 4 NEURAL NETWORK GUIDED CONVEX ABSTRACTION REFINEMENT

In this section, we are interested in designing and training an NN that can be used to guide the abstraction refinement process. Such NN can be used as an oracle by the solver to estimate the volume of the zero super/sub-level sets (for each polynomial) within a given region $I_n(\underline{d}, \overline{d})$ and select the best approximation strategy out of three possibilities namely: (i) apply convex under-approximation, (ii) apply convex over-approximation, and (iii) split the region to allow for finer approximations in the subsequent iterations of the solver. In this section, we aim to develop a scientific methodology that can guide the design of such NN.

### 4.1 On the Relation between the NN Architecture and the Characteristics of the Polynomials

In this subsection, we aim to understand how the properties of the polynomials affect the design of the NN. We start by reviewing the following result from the machine learning literature:

Theorem 1 (Theorem 1.1 [47]). *There exists a Rectifier Linear Unit (ReLU)-based NN $\phi$ that can estimate a continuous function $f$ such that the estimation error is bounded by*

$$||\phi - f|| \leq \omega_f \sqrt{d}\, O(N^{-2/d} L^{-2/d})$$

*where $N, L, d$ are the NN depth, the NN width, and the number of NN inputs, respectively, and $\omega_f$ is the Lipschitz constant of the function $f$. Moreover, this bound is nearly tight.*

The above result can be interpreted as follows. The depth $N$ and width $L$ of an NN depend on the rate of change of the underlying function (captured by its Lipschitz constant $\omega_f$). That is, if we use an NN to estimate a function with a high $\omega_f$, then one needs to increase the depth $N$ and width $L$ of the NN to achieve an acceptable estimation error.

Now we aim to connect the result above with the characteristics of the polynomials. To that end, we recall the definition of "condition numbers" of a polynomial [19]:

*Definition 1.* Given a polynomial $p(x) = \sum_K a_K x^K$ and a root $x_0$ of $p$, the quantity $C_{a_p}(x_0)$ is called the condition number for the root $x_0$. The condition number characterizes the sensitivity of the root $x_0$ to a perturbation of the coefficients $a_p$. That is, if we allow a random perturbation of a fixed relative magnitude $\epsilon = |\frac{\delta a_K}{a_K}|$ in each coefficient $a_K$ in $a_p$, then the magnitude of the maximum displacement $\delta x_0$ of a root $x_0$ is bounded as: $|\delta x_0| \leq C_{a_p}(x_0)\,\epsilon$. For a polynomial with multiple roots, then we define the condition number of the polynomial $\overline{C}_{a_p}$ as the largest $C_{a_p}(x_0)$ among all roots, i.e., $\overline{C}_{a_p} = sup_{x_0 \in \{x | p(x)=0\}} C_{a_p}(x_0)$.

We are now ready to present our first theoretical result that connects the condition number of polynomials to the NN architecture. As stated before, we are interested in designing an NN that can estimate the zero sub/super level volume set within a given region. We show that the larger the condition number, the larger the NN depth and width, as captured by the following result.

Theorem 2. *Given a polynomial $p$ with coefficients $a_p$, a region $I_n(\underline{d}, \overline{d})$, and an estimate's quality of the volume of zero sub/super level sets $l$. There exists a neural network $NN(a_p, I_n)$ that estimates the volume of zero sub/super level sets from the polynomial coefficients $a_p$. The Lipschitz constant of this $NN(a_p, I_n)$, denoted by $\omega_{NN}$ , is bounded by $O(n_r \overline{n}_r \overline{C}_{a_p})$ where $\overline{C}_{a_p}$ is the condition number of the polynomial $p$, $\overline{n}_r = max(l^{\frac{1}{n}}, n_r)$ and $n_r$ is the number of roots of the polynomial $p$.*

To prove the result, we will proceed with an existential argument. We will show that an NN that matches the properties above exists without constructing such an NN. As shown in Figure 2, the neural network $NN(a_p, I_n)$ consists of multiple sub-neural networks. In particular, the first sub-neural network $NN_{a_p \to X_0}$ computes all the roots $X_0 = (x_0^1, \ldots, x_0^{n_r})$ of the polynomial (where $n_r$ is the number of roots) from the coefficients $a_p$, i.e.,:

$$X_0 = NN_{a_p \to X_0}(a_p). \tag{2}$$

Note that $NN_{a_p \to X_0}$ does not depend on the region $I_n$ and hence the roots $X_0$ may not lie inside the region $I_n$. Moreover, Theorem 2 asks for an NN that estimates the volume of the zero sub/super level sets and not the location of the roots. To that end, our strategy is to split the region $I_n$ into sub-regions of fixed volume and check if a root lies within each of these sub-regions. If a sub-region does not have a root (i.e., there is no zero crossing inside this sub-region) and the evaluation of the polynomial at any point in this region turns to be positive, then this sub-region belongs to the super level set of $p$ and similarly for the sublevel set of $p$. By counting the number of the sub-regions with no zero crossings and multiplying this count by the volume of these sub-regions, we can provide an estimate of the sub/super level sets. Such a process can be performed using the following three sub-neural networks:

— The sub-neural network $NN_{I_n \to I_n^i}$ splits the region $I_n$ into $l$ sub-regions $I_n^1, \ldots, I_n^l$ and returns the bounds of the $i$th sub-region, i.e.,:

$$(\underline{d}^i, \overline{d}^i) = NN_{I_n \to I_n^i}(I_n), \qquad i \in \{1, \ldots, l\}. \tag{3}$$

— The sub-neural network $NN_{X_0 \to ZC_i}$ checks the location of the roots $(x_0^1, \ldots, x_0^{n_r})$ and returns a binary indicator variable $ZC_i$ that indicates whether a zero-crossing takes place within the $i$th sub-region or whether the polynomial is always positive/negative within the $i$th sub-region, i.e.,:

$$ZC_i\left(a_p, I_n^i\right) = NN_{X_0 \to ZC_i}\left(NN_{a_p \to X_0}(a_p), NN_{I_n \to I_n^i}(I_n)\right). \tag{4}$$

— The final output $NN(a_p, I_n)$ is computed using the sub-neural network $NN_{ZC \to L^+/L^-}$ which counts the number of regions that has no zero-crossing (using the indicators $ZC_1, \ldots, ZC_l$) and compute the estimate of the zero sub/super level sets, i.e.,:

$$NN(a_p, I_n) = NN_{ZC \to L^+/L^-}\left(ZC_1\left(a_p, I_n^1\right), \ldots, ZC_l\left(a_p, I_n^l\right)\right). \tag{5}$$

The Lipschitz constants of these sub-neural networks are captured by the following four propositions whose proof can be found in the appendix.

Proposition 1. *Consider the sub-neural network $NN_{a_p \to X_0}(a_p)$ defined in (2). The Lipschitz constant of $NN_{a_p \to X_0}(a_p)$ is bounded by $O(n_r \overline{C}_{a_p})$.*

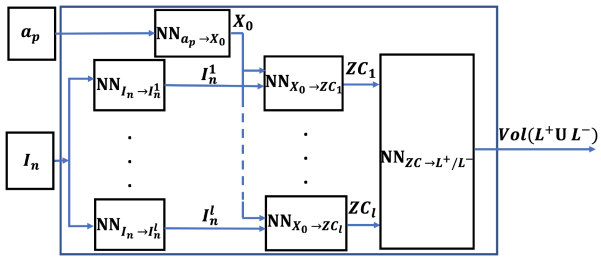


Fig. 2. The architecture of the neural network $NN(a_p, I_n)$ used to prove the Theorem 2.

Proposition 2. *Consider the sub-neural network $NN_{I_n \to I_n^i}$ defined in (3). The Lipschitz constant of $NN_{I_n \to I_n^i}$ is bounded by $O(l^{\frac{1}{n}})$.*

Proposition 3. *Consider the sub-neural network $NN_{X_0 \to ZC_i}\left(X_0, I_n^i\right)$ defined in (4). The Lipschitz constant of $NN_{X_0 \to ZC_i}\left(X_0, I_n^i\right)$ is bounded by $O(n_r)$.*

Proposition 4. *Consider the sub-neural network $NN_{ZC \to L^+/L^-}$ defined in (5). The Lipschitz constant of $NN_{ZC \to L^+/L^-}$ is bounded by $O(1)$.*

Proof of Theorem 2. Consider the NN shown in Figure 2 and defined using Equations (2)–(5). To bound the Lipschitz constant of $NN(a_p, I_n)$, we consider two sets of inputs $(a_p, I_n)$ and $(a_p', I_n')$ as follows:

$$\begin{aligned}
\left\|NN\left(a_p', I_n'\right) - NN(a_p, I_n)\right\|_2 &= \Big\|NN_{ZC \to L^+/L^-}\left(ZC_1\left(a_p', I_n'^1\right), \ldots, ZC_l\left(a_p', I_n'^l\right)\right) \\
&\quad - NN_{ZC \to L^+/L^-}\left(ZC_1\left(a_p, I_n^1\right), \ldots, ZC_l\left(a_p, I_n^l\right)\right)\Big\|_2, && (6)\\
&\leq O(1)\left\|\left(ZC_1\left(a_p', I_n'^1\right) - ZC_1\left(a_p, I_n^1\right), \ldots, ZC_l\left(a_p', I_n'^l\right) - ZC_l\left(a_p, I_n^l\right)\right)\right\|_2, && (7)\\
&= O(1)\left(\sum_{i=1}^{l}\left\|ZC_i\left(a_p', I_n'^i\right) - ZC_i\left(a_p, I_n^i\right)\right\|_2^2\right)^{\frac{1}{2}}, && (8)
\end{aligned}$$

where (7) follows from Proposition 4. Now, we upper bound $\|ZC_i(a_p', I_n'^i) - ZC_i(a_p, I_n^i)\|_2$ as follows:

$$\begin{aligned}
\left\|ZC_i\left(a_p', I_n'^i\right) - ZC_i\left(a_p, I_n^i\right)\right\|_2^2 &= \Big\|NN_{X_0 \to ZC_i}\left(NN_{a_p \to X_0}\left(a_p'\right), NN_{I_n \to I_n^i}\left(I_n'\right)\right) \\
&\quad - NN_{X_0 \to ZC_i}\left(NN_{a_p \to X_0}(a_p), NN_{I_n \to I_n^i}(I_n)\right)\Big\|_2^2 && (9)\\
&\leq O(n_r)\left\|\left(NN_{a_p \to X_0}\left(a_p'\right) - NN_{a_p \to X_0}(a_p), NN_{I_n \to I_n^i}\left(I_n'\right) - NN_{I_n \to I_n^i}(I_n)\right)\right\|_2^2, && (10)\\
&= O(n_r)\Big(\left\|\left(NN_{a_p \to X_0}\left(a_p'\right) - NN_{a_p \to X_0}(a_p)\right)\right\|_2^2 \\
&\quad + \left\|NN_{I_n \to I_n^i}\left(I_n'\right) - NN_{I_n \to I_n^i}(I_n)\right\|_2^2\Big), \\
&\leq O(n_r)\left(O\left(n_r \overline{C}_{a_p}\right)\left\|a_p' - a_p\right\|_2^2 + O(l^{\frac{1}{n}})\left\|I_n' - I_n\right\|_2^2\right) && (11)\\
&= O\left(n_r \overline{n}_r \overline{C}_{a_p}\right)\left\|\left(a_p', I_n'\right) - \left(a_p, I_n\right)\right\|_2^2, && (12)
\end{aligned}$$

where (10) follows from Proposition 3; (11) follows from Propositions 2 and 1 along with the definition of $\overline{n}_r = \max(l^{\frac{1}{n}}, n_r)$. Substituting (12) in (8) and noticing that $l$ is a constant that does not depend on $n$ yields:

$$\left\|NN\left(a_p', I_n'\right) - NN(a_p, I_n)\right\|_2 \leq O\left(n_r \overline{n}_r \overline{C}_{a_p}\right)\left\|\left(a_p', I_n'\right) - \left(a_p, I_n\right)\right\|_2, \tag{13}$$

from which we conclude that the Lipschitz constant of $NN(a_p, I_n)$ is in the order of $O(n_r \overline{n}_r \overline{C}_{a_p})$ which concludes the proof of Theorem 2. □

According to Theorem 1, the depth and width of an NN must be increased to achieve an acceptable estimation error proportional to the NN's Lipschitz constant. Additionally, Theorem 2 establishes that the Lipschitz constant of $NN\left(a_p, I_n\right)$ is upper-bounded by a constant dependent on the condition number of the polynomial $\overline{C}_{a_p}$. Consequently, we can deduce from Theorem 1 and Theorem 2 that higher condition numbers of polynomials necessitate larger network widths and depths for accurate estimation of zero sub/super level set volumes. Notably, the power basis representation (i.e., representing the polynomial as a summation $\sum_{K \leq L} a_K x^K$) has been identified to possess an unstable nature with significantly large condition numbers [19], thereby demanding NNs with substantial architectural complexities which motivates the need to use other polynomial representations.

### 4.2 Bernstein Polynomials: A Robust Representation of Polynomials

Motivated by the challenge above, we seek a representation of polynomials that is more robust to changes in coefficients, i.e., we seek a representation in which the roots of the polynomial change slowly with changes in the coefficients (and hence smaller condition numbers $\overline{C}_{a_p}$ and a smaller NN to estimate the volume of the sub/super level sets). We start with the following definition.

*Definition 2.* Let $p(x) = \sum_{K \leq L} a_K x^K \in \mathbb{R}[x_1, \ldots, x_n]$ be a multivariate polynomial over a hyperrectangle $I_n(\underline{d}, \overline{d})$ and of a maximal degree $L = (l_1, \ldots, l_n) \in \mathbb{N}^n$. The polynomial:

$$B_{p,L}(x) = \sum_{K \leq L} b_{K,L} Ber_{K,L}(x), \tag{14}$$

is called the Bernstein polynomial of $p$, where $Ber_{K,L}(x)$ and $b_{K,L}$ are called the Bernstein basis and Bernstein coefficients of $p$, respectively, and are defined as follows:

$$Ber_{K,L}(x) = \binom{L}{K} x^K (1-x)^{L-K}, \qquad b_{K,L} = \sum_{J=(0,\ldots,0)}^{K} \frac{\binom{K}{J}}{\binom{L}{J}} \left(\overline{d} - \underline{d}\right)^J \sum_{I=J}^{L} \binom{I}{J} \underline{d}^{I-J} a_I. \tag{15}$$

The Bernstein representation is known to be the most robust representation of polynomials which is captured by the next result [19].

Theorem 3 (Theorem [19]). *The Bernstein basis is optimally stable, i.e., there exists no other basis with a condition number smaller than the condition number of the Bernstein coefficients $\overline{C}_{b_p}$, where $b_p = (b_{(0,0,\ldots,0),L}, \ldots, b_{(l_1,l_2,\ldots,l_n),L})$ is the vector of all the Bernstein coefficients of polynomial $p$.*

Theorems 1–3 point to the optimal way of designing the targeted NN. Such an NN needs to take as input the Bernstein coefficients $b_p$ instead of the power basis coefficients $a_p$. To validate this conclusion, we report empirical evidence in Table 1. In this numerical experiment, we trained two NNs with the same exact architecture, using the same exact number of data points, and both networks have the same number of inputs. Both NNs are trained to estimate whether a zero-crossing occurs in a region (recall from our analysis in Theorem 2 that the Lipschitz constant of this NN is equal to the condition number of the polynomial). The only difference is that one NN is trained using power basis coefficients $a_p$ (column 3 of Table 1) while the second is trained using Bernstein basis coefficients $b_p$ (column 5 of Table 1). The coefficients are randomly generated via a uniform distribution between $-0.1$ and $0.1$, i.e., $\mathcal{U}(-0.1, 0.1)$. We generated 40,000 training samples and 10,000 validation samples for both bases. We evaluate the trained NN on three different benchmarks for

Table 1. Evaluation of Three Trained NNs on Three Different Benchmarks for the Different Polynomial Basis

| Benchmark | Coefficients | Power Basis | | Bernstein Basis | | Reduced Bernstein Basis | |
|---|---|---|---|---|---|---|---|
| | | Accuracy | Overhead | Accuracy | Overhead | Accuracy | Overhead |
| 1 | $\mathcal{U}(-0.1, 0.1)$ | 46% | 0 [s] | 91% | 0.01 [s] | 82% | 0.002 [s] |
| 2 | $\mathcal{U}(-0.5, 0.5)$ | 32% | 0 [s] | 87% | 0.03 [s] | 79% | 0.005 [s] |
| 3 | $\mathcal{U}(-1, 1)$ | 30% | 0 [s] | 88% | 0.04 [s] | 80% | 0.007 [s] |

Each benchmark has 10,000 samples. The coefficients of the polynomial within each basis are generated following a uniform distribution given in the table.

the two bases. Each evaluation benchmark has 10,000 samples. The results are summarized in Table 1. As it can be seen from Table 1, the NN trained with Bernstein coefficients generalizes better than the NN trained with power basis coefficients as reflected by the empirical "Accuracy" during evaluation. This empirical evidence matches our analysis in Theorem 2 along with the insights of Theorem 1 and Theorem 3.

## 5 TAMING THE COMPLEXITY OF COMPUTING BERNSTEIN COEFFICIENTS

In Section 4, we concluded that Bernstein's representation has a smaller condition compared to other representations, which helps build a more efficient NN. Nevertheless, computing this representation adds a significant overhead even by using the most efficient algorithms to calculate these coefficients [43, 49]. For example, computing all the Bernstein coefficients of a 6th-dimensional polynomial with 7 th order using Matrix method and Garloff's methods [43, 49] require $1.1e07$ and $7.1e06$ summation and multiplication operations [43]. To exacerbate the problem, the Bernstein coefficients depend on the region $I_n$ and need to be recomputed in every iteration of the abstraction refinement process. Reducing such overhead is the main focus of this section.

### 5.1 Range Enclosure Property of Bernstein Polynomials

Given a multivariate polynomial $p(x)$ that is defined over the $n$-dimensional box $I_n(\underline{d}, \overline{d})$, we can bound the range of $p(x)$ over $I_n(\underline{d}, \overline{d})$ using the range enclosure property of Bernstein polynomials as follows:

Theorem 4 (Theorem 2 [23]). *Let $p$ be a multivariate polynomial of degree $L$ over the $n$-dimensional box $I_n(\underline{d}, \overline{d})$ with Bernstein coefficients $b_{K,L}$, $0 \leq K \leq L$. Then, for all $x \in I_n$, the following inequality holds:*

$$\min_{K \leq L} b_{K,L} \leq p(x) \leq \max_{K \leq L} b_{K,L}. \tag{16}$$

The traditional approach to computing the range enclosure of $p$ is to compute all the Bernstein coefficients of $p$ to determine their minimum and maximum [23, 24, 56]. However, computing all the coefficients has a complexity of $O((l_{max}+1)^n)$, where $l_{max} = \max_{1 \leq i \leq n} l_i$, which increases exponentially with the dimension $n$. Luckily, the Bernstein coefficients enjoy monotonicity properties, whenever the region $I_n(\underline{d}, \overline{d})$ is restricted to be an orthant (i.e., the sign of $x_i$ does not change within $I_n(\underline{d}, \overline{d})$, for each $i \in \{1, \ldots, n\}$) [49]. Using such monotonicity properties, one can compute the minimum and maximum Bernstein coefficients (denoted by $\underline{B}_{p,L}$, $\overline{B}_{p,L}$) with a time complexity of $O\left(2(l_{max}+1)^2\right)$ which does not depend on the dimension $n$.

### 5.2 Zero Crossing Estimation Using Only a Few Bernstein Coefficients

Now we discuss how to use the range enclosure property above to reduce the number of computed Bernstein coefficients. First, we note that the zero crossing of a polynomial $p$ in a given input

region $I_n$ depends on its estimate range given by $\underline{B}_{p,L}$ and $\overline{B}_{p,L}$. More specifically, if $\underline{B}_{p,L} > 0$ ($\overline{B}_{p,L} < 0$), then the entire polynomial is positive (negative), which means that there is no zero-crossing. If $\underline{B}_{p,L}$ and $\overline{B}_{p,L}$ have different signs, and because of the estimation error of these bounds, the polynomial $p$ may still be positive, negative, or have a zero crossing in the region. In this case, we need additional information such as the bounds of the gradient of the polynomial $p$ within the input region, that are given by $\underline{B}_{\nabla p,L}$ and $\overline{B}_{\nabla p,L}$ (which can be computed efficiently thanks to the fact that gradients of polynomials are polynomials themselves). Such additional information about the worst-case gradient of the polynomial leads to a natural estimate of whether a zero crossing occurs in a region.

Due to space constraints, we omit the analysis of bounding the estimation error introduced by relying only on the maximum and minimum of the polynomial $\underline{B}_{p,L}$ and $\overline{B}_{p,L}$ along with the maximum and minimum of the gradient $\underline{B}_{\nabla p,L}$ and $\overline{B}_{\nabla p,L}$. Instead, we support our claim using the empirical evidence shown in Table 1. Using the same benchmarks used in Section 4.2, we train a third NN that takes as input only the four inputs $\underline{B}_{p,L}, \overline{B}_{p,L}, \underline{B}_{\nabla p,L}, \overline{B}_{\nabla p,L}$ and compare its generalization performance (column 7 of Table 1). As shown in the table, the third NN sacrifices some accuracy compared to the ones that use all Bernstein coefficients. But on the other side, it reduces the overhead to compute the Bernstein coefficients by order of magnitude as can be seen by comparing the "execution overhead" reported in columns 4, 6, and 8 for the power basis, the Bernstein basis, and the reduced Bernstein basis, respectively.

### 5.3 Search Space Pruning Using Bernstein Coefficients

The range enclosure property and the discussion above open the door for a natural solution of the "alternative abstraction" problem mentioned in Section 3.2. The maximum and minimum Bernstein coefficients can be used as an abstraction (in addition to convex upper and lower bounds) of high-order polynomials. Such abstractions can be refined with every iteration of the solver. They can be used to identify portions of the search space for which one of the polynomials is guaranteed to be positive (and hence a solution does not exist). More details about integrating this abstraction and the convex abstraction are given in the implementation section below.

## 6 ALGORITHM ARCHITECTURE AND IMPLEMENTATION DETAILS

In this section, we describe the implementation details of our solver PolyARBerNN. As a pre-processing step, the tool divides the input region $I_n$ into several regions such that each one is an orthant. This allows the tool to process each orthant in parallel or sequentially. The tool keeps track of all regions for which the sign of a polynomial is not fixed. These regions are called ambiguous regions, and they are stored in a list called *Ambig*. As long as the volume of the regions in this list is larger than a user-defined threshold $\epsilon$, then our tool will continuously use abstractions to identify portions in which one of the polynomials is always positive (and hence removed from the search space) or negative (and hence the tool will give higher priority for this region). The abstraction refinement is iteratively applied in Lines 5–17 of Algorithm 1. In each abstraction refinement step, the tool picks a polynomial $p$ and a region *region* based on several heuristics (Lines 6–7). In Lines 8–14, we compute the maximum/minimum Bernstein coefficients followed by checking the sign of the polynomial within this region. Suppose the Bernstein coefficients indicate that the polynomial is always positive in this region. In that case, this provides a guarantee that a solution does not exist in this region (recall that Problem 1 searches for a point where *all* polynomials are negative). Similarly, if the polynomial is always negative, then it will be added to the list of negative regions. For those polynomials for which the Bernstein abstraction failed to identify

**ALGORITHM 1:** PolyARBerNN

**Input:** $I_n(\underline{d}, \overline{d}), p_1, p_2, \ldots, p_m, \epsilon,$ **Output**: $x_{\text{Sol}}$

1: $orthants$ := Partition_Region($I_n$)
2: $Neg := \{\}$
3: $Ambig := \{orthants\}$
4: List_pols := $\{p_1, \ldots, p_m\}$
5: **while** Compute_Maximum_Volume($Ambig$) $\geq \epsilon$ **do**
6: $p$ := Select_Poly (List_pols, $Neg$)
7: $region$ := Remove_Ambiguous_Region_From_List ($Ambig$)
8: $\left(\underline{B}_{p,L}, \overline{B}_{p,L}, \underline{B}_{\nabla p,L}, \overline{B}_{\nabla p,L}\right)$ := Compute_Bern_Coeff($p, region$)
9: **if** $\underline{B}_{p,L} > 0$ **then**
10: **break**
11: **else if** $\underline{B}_{p,L} < 0$ **then**
12: $Neg := Neg \cup (p, L_0^-(p))$
13: **break**
14: **end if**
15: (under_approx, over_approx, split) := $NN(\underline{B}_{p,L}, \overline{B}_{p,L}, \underline{B}_{\nabla p,L}, \overline{B}_{\nabla p,L}, region)$
16: $action$ := Select_best_action(under_approx, over_approx, split)
17: $L_0^-(p), L_0^+(p), L_0^{+/-}(p)$ := Convex_Abst_Refin_PolyAR ($p, action, region$)
18: $Ambig := Amibg \cup L_0^{+/-}(p)$
19: $Neg := Neg \cup (p, L_0^-(p))$
20: **end while**
21: **if** is_List_Empty($Ambig$) **then**
22: **if** A negative region in $Neg$ has all the polynomials **then**
23: $x_{\text{Sol}}$ := any point in the negative region
24: **else**
25: **return** the problem is UNSAT
26: **end if**
27: **else**
28: $x_{\text{Sol}}$ := CAD_Solver_Parallel ($Ambig, p_1, \ldots, p_m$)
29: **end if**

their signs, we query the trained NN to estimate the best convex abstraction possible (Lines 15–16). Based on the NN suggestion, we use the PolyAR tool [55] to compute the convex abstraction (Line 17), which returns portions of this region that are guaranteed to belong to the zero sublevel set $L_0^-(p)$, those who belong to the zero superlevel set $L_0^+(p)$, and those remain ambiguous $L_0^{+/-}(p)$. The process of using Bernstein abstraction and the convex abstraction (which is guided by the trained NN) continues until all remaining ambiguous regions are smaller than a user-defined threshold $\epsilon$ in which case it will be processed in parallel using a sound and complete tool that implements CAD such as Z3 and Yices (Line 28 in Algorithm 1).

The NN itself is trained using randomly generated, quadratic, two-dimensional polynomials where the coefficients follow a uniform distribution between $-1$ and 1. For each randomly generated polynomial, we used PolyAR to compute the volumes of the $L_0^+(p), L_0^-(p), L_0^{+/-}(p)$ regions. We use a fully connected NN that contains an input layer, three hidden layers, and one output layer (shown in Figure 3). The input layer has 4 neurons, the hidden layers have 40 neurons each, and the output layer has 3 neurons. We use a dropout of probability 0.5 in the first and second hidden

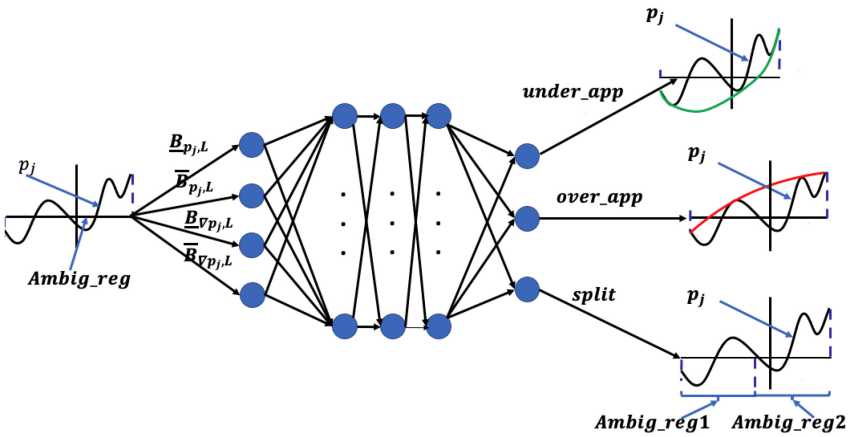


Fig. 3. The architecture of the trained NN that is used to guide the abstraction refinement process within PolyARBerNN. We used a fully connected NN that contains an input layer with 4 neurons, three hidden layers with 40 neurons each, and one output layer with 3 neurons. All neurons are ReLU-based except for the output neurons which uses SoftMax nonlinearity.

layers to avoid overfitting. We use the ReLU activation function for all the hidden layers and the Softmax activation function for the output layer. We use Adam as an optimizer and cross-entropy as a loss function. Although the NN is trained on simple quadratic two-dimensional polynomials, we observed it generalizes well to higher-order polynomials with several variables. This will become apparent during the numerical evaluation in which polynomials of different orders and several variables will be used to evaluate the tool.

**Correctness Guarantees:** We conclude our discussion with the following result which captures the correctness guarantees of the proposed tool:

Theorem 5. *The PolyARBerNN solver is sound and complete.*

Proof. This result follows from the fact that search space is pruned using sound abstractions (convex upper bounds or Bernstein-based). The NN and the convex lower bound polynomials are just used as heuristics to guide the refinement process. Finally, CAD-based algorithms (which are sound and complete) are used to process the portions of the search space which are not pruned by the abstraction refinement. □

## 7 GENERALIZATION TO POLYNOMIAL OPTIMIZATION PROBLEMS

In this section, we focus on providing a solution to Problem 2. Our approach is to turn the optimization problem (Problem 2) into a feasibility problem (Problem 1). First, we recall that the gradient of $p$, $\nabla p = [\frac{\partial p}{\partial x_1}, \cdots, \frac{\partial p}{\partial x_n}]$, where $\frac{\partial p}{\partial x_i}$ is the partial derivative of $p$ with respect to $x_i$, is a vector of $n$ polynomials. The optimal value of $p$ occurs either (i) when the vector of partial derivatives are all equal to zero or (ii) at the boundaries of the input region.

To find the critical points $x^*$ of $p$ where $\nabla p(x^*) = 0$, we add the $n$ polynomial constraints $\frac{\partial p}{\partial x_i} \leq 0, 1 \leq i \leq n$ and $-\frac{\partial p}{\partial x_i} \leq 0, 1 \leq i \leq n$ to the constraints of the optimization problem. Now, we modify the PolyARBernNN solver to output *all* possible regions in which all the constraints are satisfied. This can be easily computed by taking the intersections within the regions stored in the data structure *Neg* in Algorithm 1. These regions enjoy the property that *all* points in these regions are critical points of $p$. In addition, we modify PolyARBernNN to output *all* the remaining

**ALGORITHM 2:** PolyAROpt

**Input:** $I_n(\underline{d}, \overline{d}), p, p_1, p_2, \ldots, p_m, \epsilon$
**Output**: $\hat{p}_{min}, \hat{p}_{max}$

1: $\nabla p = \texttt{Grad\_Poly}(p)$
2: $r\hat{e}g_{\text{list}} = \texttt{PolyARBerNN}(I_n(\underline{d}, \overline{d}), \nabla p, p_1, \ldots, p_m, \epsilon)$
3: $\hat{x}_{\text{list}} = \texttt{center}(r\hat{e}g_{\text{list}})$
4: $x^{end}_{list} = \texttt{Sample\_boundaries}(I_n(\underline{d}, \overline{d}), \epsilon)$
5: $\hat{p}_{list} = p(\hat{x}_{\text{list}}); \quad p^{end}_{list} = p(x^{end}_{\text{list}})$
6: $p_{list} = \hat{p}_{list} \cup p^{end}_{list}$
7: $\hat{p}_{min} = \min(p_{list}); \quad \hat{p}_{max} = \max(p_{list})$

ambiguous regions whose volumes are smaller than the user-specified threshold $\epsilon$ and for which the CAD-based solvers returned a solution. These regions enjoy the property that *there exists* a point inside these regions which is a critical point. These modifications are captured in Line 2 of Algorithm 2.

Since the minimum/maximum of $p$ may occur at the boundaries of the region $I_n(\underline{d}, \overline{d})$, our solver samples from the boundaries of the region $I_n(\underline{d}, \overline{d})$ (Line 4 in Algorithm 2). The solver uses $\delta = 2\sqrt{n}(\epsilon)^{1/n}$ as sampling distance between two successive boundary samples—recall $\epsilon$ is a user-defined parameter and was used in Algorithm 1 as a threshold on the refinement process. Next, we evaluate the polynomial p in the obtained samples (Line 5 Algorithm 2). Then, we take the minimum and the maximum over the obtained values (Line 7 in Algorithm 2). All the details can be found in Algorithm 2.

We conclude our discussion with the following result which captures the error between the solutions provided by PolyAROpt and the global optima.

Theorem 6. *Let $p^*_{min}$ and $p^*_{max}$ be the global optimal points for the solution of Problem 2. The solutions obtained by Algorithm 2, denoted by $\hat{p}_{min}$ and $\hat{p}_{max}$, satisfy the following:*

$$|\hat{p}_{min} - p^*_{min}| \leq \omega_p \delta, \qquad |\hat{p}_{max} - p^*_{max}| \leq \omega_p \delta, \tag{17}$$

*where $\omega_p$ is the Lipschitz constant of the polynomial p, $\delta = 2\sqrt{n}(\epsilon)^{1/n}$ is the sampling distance, and $\epsilon > 0$ is a user-defined error.*

Proof. Let us denote by $x_{list}$ the input domain points that correspond to $p_{list}$. Let us denote by $\overline{x}_{min} = \min_{x \in x_{list}} \|x - x^*_{min}\|$ and $\overline{x}_{max} = \min_{x \in x_{list}} \|x - x^*_{max}\|$, the nearest point to the actual optimal points $x^*_{min}$ and $x^*_{max}$. We note that there are three cases that Algorithm 2 uses to compute the set of critical points, $x_{list}$:

(1) Using the center of the regions in the *Neg* list
(2) Using the center of the regions in the *Ambig* list
(3) Using samples from the boundaries

We proceed by case analysis. **Case 1:** First, we note that *all* the points within the *Neg* regions satisfy that $\nabla p = 0$ and hence the value of the polynomial takes the same exact value overall the region, hence the value of $p$ at the center $\hat{x}$ of the region is the same at the global optima $x^*$. **Case 2 and Case 3:** First, we can show that $\overline{x}_{min}$ and $\overline{x}_{max}$ are bounded from the actual optimal points $x^*_{min}$ and $x^*_{max}$ by

$$\left\|\overline{x}_{min} - x^*_{min}\right\| \leq \delta \qquad \left\|\overline{x}_{max} - x^*_{max}\right\| \leq \delta \tag{18}$$

where $\delta = 2\sqrt{n}(\epsilon)^{1/n}$. To show that inequalities (18) hold we proceed by case analysis. However (18) follow directly in Case 2 from the fact that *Ambig* regions have a volume that is smaller than $\epsilon$ (Line 7 in Algorithm 1) and hence the distance between any two points within the regions is bounded by $\delta = 2\sqrt{n}(\epsilon)^{1/n}$. Similarly, Case 3 follows from the fact that Algorithm 2 samples from the boundaries with a maximum distance between the samples that is equal to $\delta = 2\sqrt{n}(\epsilon)^{1/n}$. Second, we obtain (17) as follows:

$$\hat{p}_{min} \leq p(\overline{x}_{min}) \Rightarrow |\hat{p}_{min} - p^*_{min}| \leq |p(\overline{x}_{min}) - p^*_{min}| \leq \omega_p \delta, \tag{19}$$

where (19) comes from the definition of $\hat{p}_{min}$, the Lipschitz continuity of polynomial $p$, and (18). The inequality for $|\hat{p}_{max} - p^*_{max}|$ is obtained in a similar manner. □

## 8 NUMERICAL RESULTS—NN TRAINING

In this section, we show the details of training and evaluating the NN used to help PolyARBerNN selecting the best convex abstraction. We evaluate the trained NN on six different benchmarks. The benchmarks are different than the training benchmarks with respect to the input region, the degree of the polynomial, and the number of variables of the polynomial. All the experiments were executed on an Intel Core i7 2.6-GHz processor with 16 GB of memory.

### 8.1 Training Data Collection and Pre-Processing

*8.1.1 Data collection.* To collect the data, we generated random quadratic two-dimensional polynomials:

$$q(x_1,\ x_2) = c_1x_1^2 + c_2x_2 + c_3x_1x_2 + c_4x_2 + c_5y + c_6,$$

where the coefficients $c_1, \ldots, c_6$ follow a uniform distribution between $-1$ and 1.The random generated polynomials are defined over the domain $I_2 = [-2, 2]^2$. For each randomly generated polynomial, we perform the abstraction refinement on the domain $I_2$, iteratively. In every iteration, we perform under-approximation, over-approximation of the original polynomial over a selected ambiguous region, and a split of the ambiguous region. Next, we compute the volume of the remaining ambiguous region after each action was implemented. The labels are a one-hot vector of dimension three where each component represents the action that leads to the maximum reduction in the volume of the ambiguous region, either under-approximation, over-approximation or divide the region into two regions. We ran the abstraction refinement process on all the generated polynomials to collect the data $\left(\underline{B^i}_{p_j,L}, \overline{B^i}_{p_j,L}, \underline{B^i}_{\nabla p_j,L}, \overline{B^i}_{\nabla p_j,L}\right)$, where $i$ denotes the index of the sample. We generate 50,000 samples for training, 10,000 samples for validation, and 10,000 for testing.

*8.1.2 Data Normalization.* In the literature of NN [31], it is important to normalize the data when the data vary across a wide range of values. This normalization leads to faster training and improves the generalization performance of the NN [31]. Therefore, we normalize all the input data to a zero mean and unit variance by adopting a simple affine transformation data_sample $\leftarrow \frac{\text{data\_sample}-\mu}{\sigma}$, where $\mu$ and $\sigma$ are the mean of the data and its standard deviation. The $\mu$ and $\sigma$ parameters are initialized with, respectively, the empirical mean and standard deviation of the dataset and they are computed offline before the training.

### 8.2 NN's Evaluation

We evaluated the NN on six different benchmarks as follows:

— In the first and second benchmarks, we generate the same random quadratic polynomials but in a different domain: $[-4, 4]^2$ and $[-10, 10]^2$. This choice is made to test the generalization of the NN outside the data domains that were used in its training. This is important since

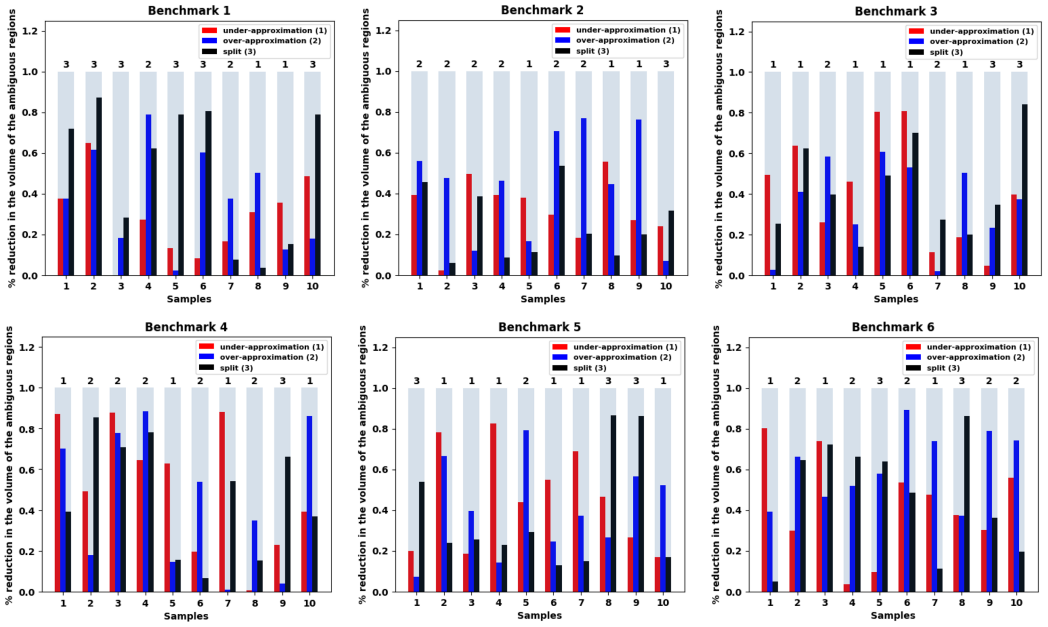


Fig. 4. Percentage in reduction of the volume of ambiguous regions along with the NN output number (the number is at the top of histograms) for 20 samples for 6 evaluation benchmarks described in Table 2.

the Bernstein coefficients of a polynomial (the input to the NN) depend on the input region $I_n$.

— In the third and fourth benchmarks, we generated random polynomials with degrees 4 and 10 over the domain $[-2, 2]^2$. These benchmarks are used to validate the generalization of the NN to polynomials of orders higher than the ones used in its training.

— Finally, in the fifth and sixth benchmarks, we generated random polynomials with higher dimensions, i.e., with dimension $n = 4$ and $n = 7$.

In summary, these benchmarks will help us to answer the following question: Can the trained NN generalize to new data with different domains (benchmarks 1 and 2), higher orders (benchmarks 3 and 4), and higher dimensions (benchmarks 5 and 6)? More detail about the different benchmarks is shown in Table 2.

Figure 4 shows the performance of the trained NN over 20 random samples of each of the six benchmarks. For each sample, we used the framework in [55] to compute the ground-truth percentage in the reduction of the volume of ambiguous regions after applying every action under-approximation, over-approximation, or split. We then evaluated the NN on each sample and reported in Figure 4 both the ground-truth reduction of the ambiguous regions (as bars) against the index of the action suggested by the NN (as the text above the bars). As it can be seen from Figure 4, except for the second sample of the first benchmark, the NN outputs represent the actions that lead to the maximum reduction of the ambiguous region's volume.

Finally, we ran the same experiment for 1000 samples and report the percentage of samples for which the NN was able to predict the action that leads to the maximum reduction in the ambiguous region's volume. As it can be seen from Table 2, the trained NN is able to generalize on the different benchmarks. For instance, evaluating the NN on different domains results in the lowest accuracy of 93%. Furthermore, evaluating the NN on polynomials with higher-order results in an accuracy of 87%. Finally, the NN achieves 80% on higher dimension benchmarks.

Table 2. Evaluation of the Trained NN on the Six Different Benchmarks

| Benchmark | $p(x)$ | $n$ | order | region | Accuracy |
|---|---|---|---|---|---|
| 1 | $c_1x_1^2 + c_2x_2 + c_3x_1x_2 + c_4x_2 + c_5y + c_6$ | 2 | 2 | $[-4, 4]^2$ | 95% |
| 2 | $c_1x_1^2 + c_2x_2 + c_3x_1x_2 + c_4x_2 + c_5y + c_6$ | 2 | 2 | $[-10, 10]^2$ | 93% |
| 3 | $c_1x_1^4 + c_2x_2^3 + c_3x_1^4x_2^3 + c_4x_1^3 + c_5$ | 2 | 4 | $[-2, 2]^2$ | 88% |
| 4 | $c_1x_1^{10} + c_2x_2^5 + c_3x_1^5x_2^3 + c_4x_1^5 + c_5$ | 2 | 10 | $[-2, 2]^2$ | 87% |
| 5 | $c_1x_1^3 + c_2x_2^3 + c_3x_3^3 + c_4x_4^3$ | 4 | 3 | $[-2, 2]^4$ | 81% |
| 6 | $c_1x_1^3 + c_2x_2^3 + c_3x_3^3 + c_4x_4^3 + c_5x_5^3 + c_6x_6^3 + c_7x_7^3$ | 7 | 3 | $[-2, 2]^7$ | 80% |

## 9 NUMERICAL RESULTS—SCALABILITY RESULTS

In this section, we study the scalability of PolyARBerNN in terms of execution times by varying the order, the number of variables, and the number of the polynomial constraints for instances when a solution exists (the problem is Satisfiabile or SAT for short) and when a solution does not exist (or UNSAT for short). We will perform this study in comparison with state-of-the-art solvers including our previous solver PolyAR, Z3 8.9, and Yices 2.6. Next, we compare the performance of PolyARBerNN against a theorem prover named PVS which implements a Bernstein library to solve multivariate polynomial constraints [38]. Finally, we compare the scalability of the PolyAROpt optimizer against the built-in optimization library in Z3 8.9 to solve an unconstrained multivariate polynomial optimization problem with varying order and number of variables.

### 9.1 Scalability of PolyARBerNN Against Other SMT Solvers

In this experiment, we compare the execution times of PolyARBerNN against the PolyAR tool [55], Z3 8.9, and Yices 2.6. We consider two instances of Problem 1: an UNSAT and SAT problems. For each instance, we consider three scenarios, $m = 1$, $m = 5$, and $m = 10$ where $m$ is the number of polynomial constraints. First, we vary the order of the polynomials from 0 to 1000 while fixing the number of variables (and hence the dimension of the search space) to two. Alternatively, we also fix the order of the polynomials to 30 while varying the number of variables from 1 to 200. We set the timeout of the simulations to be 1 hour. Figure 5 reports the execution times for all the experiments whenever the problem is UNSAT and SAT. As evidenced by the figures, PolyARBerNN succeeded to solve the instances of Problem 1 for all orders and numbers of variables in a few seconds. For instance, solving 10 polynomial constraints with 200 variables and a maximum order of 30 took around 20 $s$ leading to a speed-up of 200× compared to Z3 and Yices. On the other hand, other solvers are incapable of solving the polynomial constraints for all orders or number of variables and they time out after 1 hour. These results show the scalability of the proposed approach by including Bernstein coefficients to prune the search space and an NN to guide the abstraction refinement.

### 9.2 Scalability of PolyARBerNN Against Other Bernstein-Based Solvers

PolyARBerNN was compared against Bernstein-based solvers such as PVS [38] and Realroot [37] on computing a root/solution for the following multivariate polynomial equation system. The scalability was evaluated by fixing the number of variables, changing the maximum order, fixing the maximum order, and varying the number of variables. PolyARBerNN successfully solved the systems for all orders and variables, while Realroot and PVS encountered timeouts. The execution times are shown in Figure 6 for both scenarios, with PolyARBerNN outperforming the other solvers.

### 9.3 Scalability of PolyAROpt Against Other Solvers

We compare the scalability results of PolyAROpt with the Z3 solver since Z3 has a built-in optimization library, Bernstein-based optimizer Borderbasix [51], and SOSTool [2]. Unfortunately,

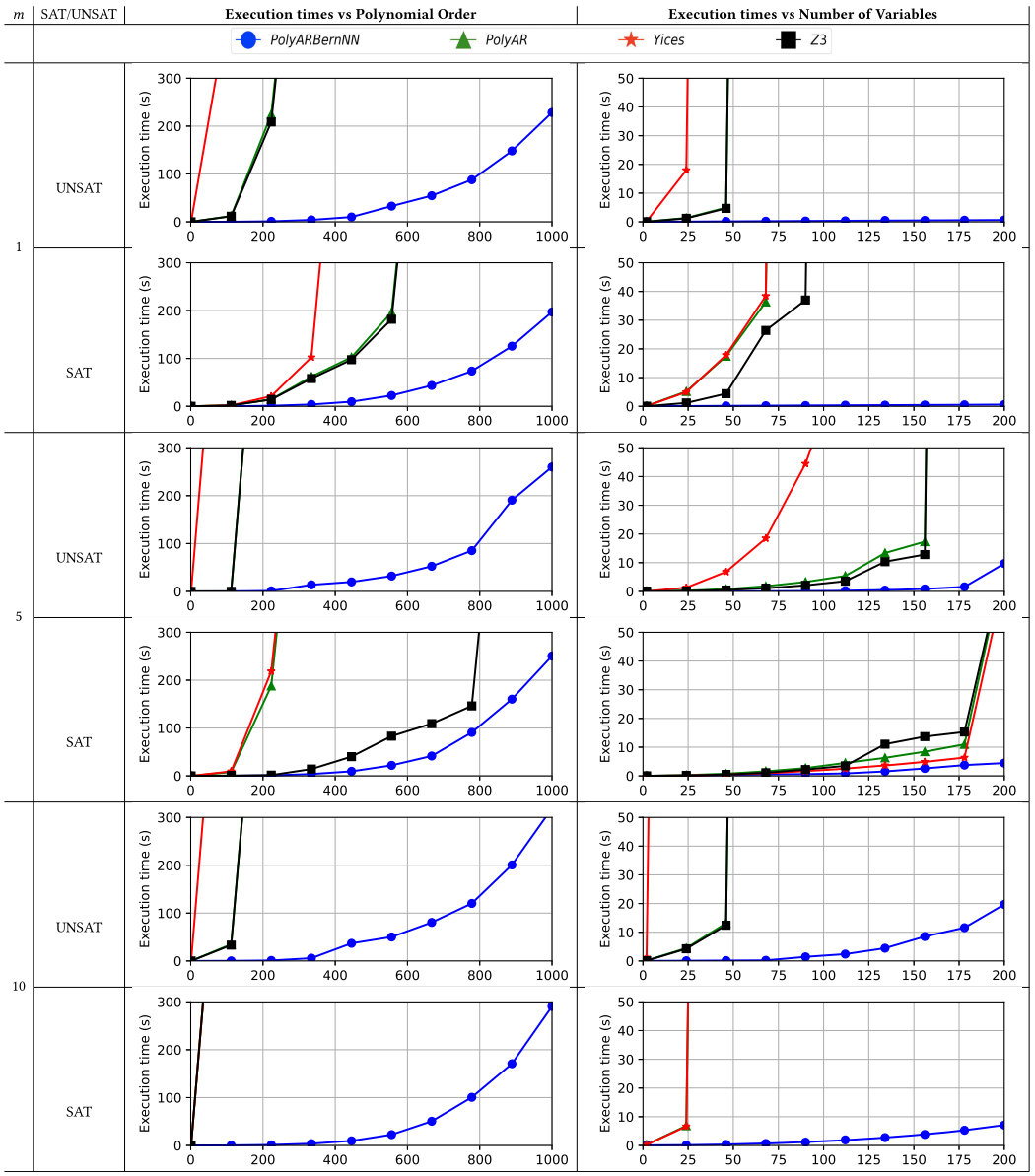


Fig. 5. Scalability results of PolyARBerNN in the UNSAT case for 1, 5, and 10 constraints. (Left) evolution of the execution time in seconds as a function of the order of the polynomials, (right) evolution of the execution time in seconds as a function of the number of variables. The timeout is equal to 1 hour.

Yices does not have such an optimizer. We set the timeout of the experiment to be 1 hour. Figure 7 reports the execution times of two experiments that compute unconstrained optimization's minimum and maximum. As evidenced by the two figures, PolyAROpt succeeded in solving the unconstrained optimization problem for all orders and numbers of variables. For instance, solving the unconstrained optimization problem with 70 variables and a maximum order of 3 took around 50 *seconds*. On the other hand, the Z3 Borderbasix, SOSTool solvers cannot solve the unconstrained optimization problem for all orders or number of variables and time out.

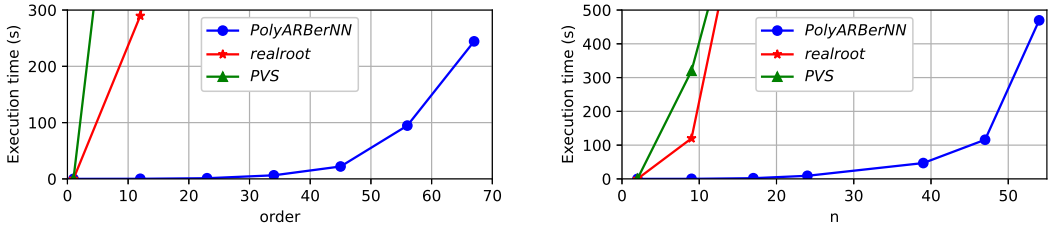


Fig. 6. Scalability results of PolyARBerNN for multivariate polynomial equation system over the interval $I_n = [-1, 1]^n$. (Left) evolution of the execution time in seconds as a function of the order of the polynomial with the number of variables $n = 3$. (Right) evolution of the execution time in seconds as a function of the number of variables with maximum order equal to 3. The timeout is equal to 1 hour.

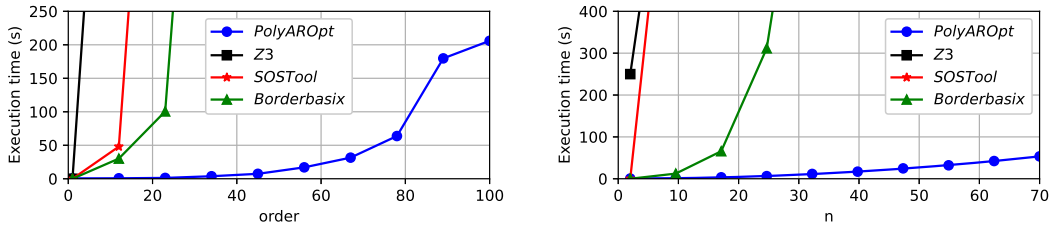


Fig. 7. Scalability results of PolyAROpt for unconstrained optimization over the interval $I_n = [-1, 1]^n$. (Left) evolution of the execution time in seconds as a function of the polynomial order. (Right) evolution of the execution time in seconds as a function of the number of variables. The timeout is equal to 1 hour.

## 10 NUMERICAL RESULTS—USE CASES

In this section, we provide two engineering use cases. The first one focuses on the use of PolyARBerNN to synthesize stabilizing non-parametric controllers for nonlinear dynamical systems. The second use case focuses on the use of PolyAROPT to perform reachability analysis of polynomial dynamical systems.

### 10.1 Use Case 1: Nonlinear Controller Design for a Duffing Oscillator

In this subsection, we assess the scalability of the PolyARBerNN solver compared to state-of-the-art solvers for synthesizing a non-parametric controller for a Duffing oscillator reported by [22]. All the details of the dynamics of the oscillator and how we generated the polynomial constraints can be found in [55]. We denote by $n$ the dimension of the Duffing oscillator. We consider two instances of the controller synthesis problem for the Duffing oscillator with the following parameters:

- $n = 3$, $\zeta = 1.0$, $x(0) = [0.15, 0.15, 0.15]$, $L_1(x(k), u(k)) = \left(-x_1^3(k) + x_3^3(k) + u(k) - 2\right)^{51}$, $L_2(x(k), u(k)) = x_1^{51}(k)\, x_3^7(k) + x_1^9(k)\, x_3^5(k) - 5x_2^4(k) - x_2^2(k)\, u^2(k)$, which results in 9 polynomial constraints with 4 variables and max polynomial order of 153.
- $n = 4$, $\zeta = 1.75$, $x(0) = [0.1, 0.1, 0.01, 0.1]$, $L_1(x(k), u(k)) = x_1^4(k) + x_2^4(k) + x_3^4(k) + x_4^4(k) - u^4(k)$, $L_2(x(k), u(k)) = -x_1^{51}(k)\, x_3^{20}(k) - 5x_2^4(k) - x_2^2(k)\, u^2(k)$, $L_3(x(k), u(k)) = \left(x_1 x_2^2 - u(k) - 100\right)^{41}$, which results in 12 polynomial constraints with 5 variables and max polynomial order of 82.

We feed the resultant polynomial inequality constraint to PolyARBerNN, Yices, and Z3. We solve the feasibility problem for $n = 3$ and $n = 4$. We set the timeout to be 60$s$. Figure 8 (left) shows the state-space evolution of the controlled Duffing oscillator for different solvers for number of variables $n$ of 3 and 4. Figure 8 (right) shows the evolution of the execution time of the solvers

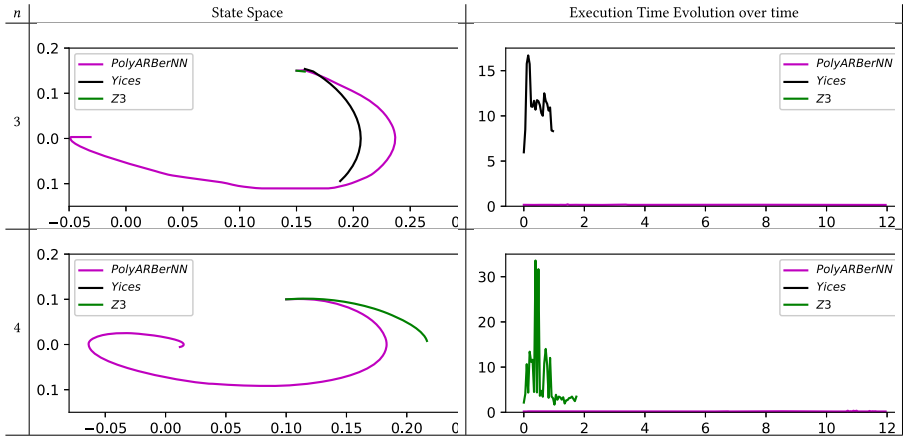


Fig. 8. Results of controlling the Duffing oscillator with different $n$ (left) evolution of the states $x_1(k)$ and $x_2(k)$ for the solvers in the state-space, (right) evolution of the execution time of solvers during the 12 seconds. The timeout is equal to 60$s$. Trajectories are truncated once the solver exceeds the timeout limit.

during the 12 seconds. As it can be seen from Figure 7, our solver PolyARBerNN succeeded to find a control input $u$ that regulates the state to the origin for all $n$. However, off-the-shelf solvers are incapable of solving all two instances, and they early time out after 60 *seconds* out of the simulated 12 *seconds*. This shows the scalability of the proposed approach.

### 10.2 Use Case 2: Reachability Analysis of a Discrete Polynomial Dynamical Systems

In this section, we show how to use PolyAROpt to compute the reachable set of states for discrete-time polynomial systems. We consider a discrete polynomial dynamical system of the following form:

$$x_{k+1} = f(x_k), \; k \in \mathbb{N}, x_0 \in Q_0, \tag{20}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a multivariate polynomial map of a maximum degree $L = (l_1, \ldots, l_n)$, and $Q_0$ is a bounded polyhedron in $\mathbb{R}^n$. In this subsection, we consider a bounded-time reachability analysis of the system in (20). Computing the exact reachability sets of this type of dynamic system is hard. Therefore, we overapproximate the exact set with a simplified set such as bounded polyhedra. Bounded polyhedra are easy to handle and analyze. Computing the reachability set, after a finite time $K$, involves computing sequentially the reachability set at every timestep $k$ using the following relation $Q_{k+1} = f(Q_k), k = 0, \ldots, K-1$, where $Q_k$ is a bounded polyhedra. A bounded polyhedra is represented with an H-representation $Q_k = (A_k, b_k) = \{x \in \mathbb{R}^n | A_k x \leq b_k\}$, where the inequality is a point-wise inequality. The template $A_k$ represents the directions of $Q_k$'s faces and $b_k$ represents their positions. Given a polyhedra $Q_k = (A_k, b_k)$, we need to compute $Q_{k+1} = f(Q_k)$. We assume that $A_k \in \mathbb{R}^{m \times n}$ is given. Now, we need to compute $b_{k+1} \in \mathbb{R}^m$ which can be obtained through the following optimization problem [6]:

$$-b_{k+1,i} \leq u_{k+1,i} = \min_{x \in Q_k} -A_{k+1,i} f(x), \forall i = 1, \ldots, m, \tag{21}$$

where $A_{k+1,i}$ and $b_{k+1,i}$ are the $i$ th row and component of the templates $A_{k+1}$ and $b_{k+1}$, respectively. In every step $k \in \mathbb{N}$, we compute an upper bound for $-b_{k+1,i}$, by solving the optimization problem (21) using PolyAROpt and then an overapproximation of the reachability set $Q_{k+1}$ is computed. In order to use PolyAROpt to solve (21), we need to overapproximate the polyhedra $Q_k$ with a

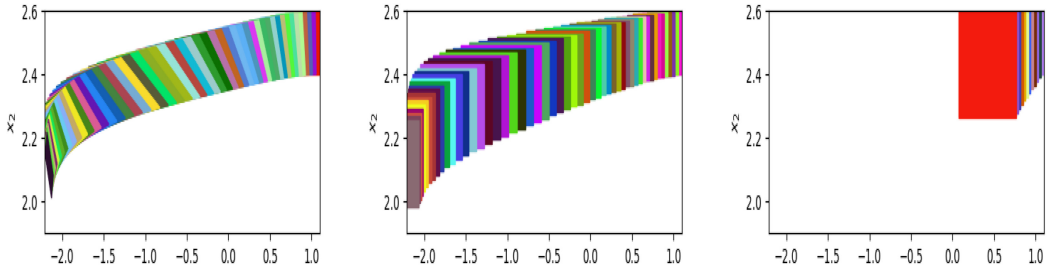


Fig. 9. Reachability computation for the FitzHugh-Nagumo neuron model. **Left:** using PolyAROpt for number of steps $K = 50$. **Center:** using Sapo for number of steps $K = 50$. **Right:** using Flowstar for number of steps $K = 50$.

hyperrectangle $R_k$. Therefore, the optimization problem (21) is modified as follows:

$$-b_{k+1,i} \leq u_{k+1,i} = \min_{x \in R_k} -A_{k+1,i} f(x), \forall i = 1, \ldots, m, \quad \text{subject to } x \in Q_k. \tag{22}$$

We implemented the reachability computation method and tested it on three dynamical systems:

— FitzHugh-Nagumo Neuron: is a polynomial dynamic system that models the electrical activity of a neuron [21]. We performed reachability analysis for $K = 50$ timesteps with the initial set of states $Q_0 = [0.9, 1.1] \times [2.4, 2.6]$,
— Duffing Oscillator: is a discrete-time version of a nonlinear oscillator model [55]. We performed reachability analysis for $K = 50$ timesteps with the initial set of states $Q_0 = [2.49, 2.51] \times [1.49, 1.51]$,
— Jet Flight: is a discrete-time version of a jet flight model [10]. We performed reachability analysis for $K = 50$ timesteps with the initial set of states $Q_0 = [0.9, 1.2] \times [0.9, 1.2]$.

All the details of the dynamics of the three dynamical systems and how we generated the polynomial constraints can be found in [6, 10, 21, 55]. We computed the reachable sets for these dynamical systems and compared our results with Sapo [15] and Flowstar [11]. Sapo is a tool proposed for the reachability analysis of the class of discrete-time polynomial dynamical systems. Sapo linearizes the optimization problem (21) using the Bernstein form of the polynomial and was shown to outperform state-of-the-art reachability analysis tools like Flowstar [11]. Flowstar [11] is a tool used in the ARCH workshop competition for hybrid systems' reachability. This tool is a state-of-the-art reachability analysis that uses Taylor models to compute the reachable sets. Figure 9 shows the reachable sets computed by PolyAROpt compared to Sapo and Flowstar for the FitzHugh-Nagumo Neuron model. Inspecting the results in Figure 9 qualitatively shows that PolyAROpt is capable of computing tighter sets compared to Sapo and Flowstar. Flowstar stopped the computation of reachable sets at the $10 - th$ step due to large overestimation errors. To quantitatively compare the results of PolyAROpt, Sapo, and Flowstar, we compute the volume of each reachable set (for different timesteps). Figure 10 shows these volumes for all the three dynamical systems mentioned above. As evident by the results in Figure 10, PolyAROpt results in reachable sets that are tighter than the one obtained from the Sapo and Flowstar thanks to PolyAROpt's ability to solve the polynomial optimization problem without any relaxations or using Taylor models. Such ability to avoid relaxation results in several orders of magnitude reduction in the volume of the reachable sets compared to Sapo and Flowstar; a significant improvement in the analysis of such dynamical systems.

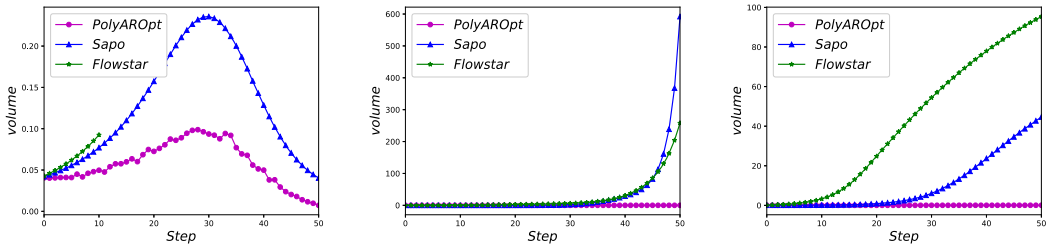


Fig. 10. The volume of the reachable set of states that are obtained using PolyAROpt, Sapo, and Flowstar (left) FitzHugh-Nagumo, (center) Duffing oscillator, and (right) Jet flight.

**Conclusions.** In this article, we proposed PolyARBerNN, a solver for polynomial inequality constraints. We proposed a systematic methodology to design NNs that can be used to guide the abstraction refinement process by bridging the gap between NN properties and the properties of polynomial representations. We showed that the use of Bernstein coefficients leads the way to designing better NN guides and provides an additional abstraction that can be used to accelerate the solver. We generalized the solver to reason about optimization problems. We demonstrated that the proposed solver outperforms state-of-the-art tools by several orders of magnitude.

## APPENDICES

## A PROOF OF PROPOSITION 1

Proof. Note that $NN_{a_p \to X_0}(a_p)$ is a vector-valued function which returns the roots $X_0 = (x_0^1, x_0^2, \ldots, x_0^{n_r})$ of the polynomial $p$. Therefore, to upper bound the Lipschitz constant of $NN_{a_p \to X_0}(a_p)$, we will start by upper bounding the Lipschitz constant of its components functions $NN^j_{a_p \to x_0^j}(a_p)$, $1 \leq j \leq n_r$. To that end, consider two polynomials with coefficients $a_p$ and $a'_p$ such that $\|a_p - a'_p\| \leq \epsilon$. Therefore:

$$\left\| NN^j_{a_p \to x_0^j}(a_p) - NN^j_{a_p \to x_0^j}(a'_p) \right\|_2 = \left\| x_0^j(a_p) - x_0^j(a'_p) \right\|_2 \leq C_{a_p}(x_0^j) \left\| a_p - a'_p \right\| \leq \overline{C}_{a_p} \epsilon \tag{23}$$

where $x_0^j(a_p)$ and $x_0^j(a'_p)$ are the location of the $j$th root for the polynomials with coefficients $a_p$ and $a'_p$, respectively. The last two inequalities follow from the definition of the condition number (Definition 1). Now,

$$\left\| NN_{a_p \to X_0}(a_p) - NN_{a_p \to X_0}(a'_p) \right\|_2 = \sqrt{\sum_{j=1}^{n_r} \left\| NN^j_{a_p \to x_0^j}(a_p) - NN^j_{a_p \to x_0^j}(a'_p) \right\|_2^2} \tag{24}$$

$$\leq \sqrt{n_r \overline{C}_{a_p}^2 \epsilon^2} = \sqrt{n_r} \overline{C}_{a_p} \epsilon. \tag{25}$$

From which we conclude that the Lipschitz constant of $NN_{a_p \to X_0}(a_p)$ is bounded by $\sqrt{n_r}\overline{C}_{a_p} = O(n_r \overline{C}_{a_p})$. □

## B PROOF OF PROPOSITION 2

Proof. We assume that the number of sub-regions $l$ is fixed for each dimension $n$, and $l^{\frac{1}{n}} = k \in \mathbb{N}$. Partitioning the input space into $l$ sub-regions occurs by dividing the interval for each dimension into $k$ sub-intervals. Without loss of generality, the sub-neural network $NN_{I_n \to I_n^i}(I_n)$

for $n = 1$ can be defined as follows:

$$(\underline{d}^i, \overline{d}^i) = NN_{I_n \to I_n^i}(I_n) = NN_{I_n \to I_n^i}(\underline{d}, \overline{d}) = \left(\underline{d} + \frac{i}{k}\left(\overline{d} - \underline{d}\right),\ \overline{d} + \frac{i+1}{k}\left(\overline{d} - \underline{d}\right)\right)$$
$$= \begin{bmatrix} 1 + i/k & -i/k \\ -(i+1)/k & 1 + (i+1)/k \end{bmatrix} \begin{bmatrix} \underline{d} \\ \overline{d} \end{bmatrix} \tag{26}$$

A generalization to a higher dimension is straightforward by replacing $i$ with a multi-index in each dimension. Note that $NN_{I_n \to I_n^i}(I_n)$ is a multivariate linear function in its inputs and hence its Lipschitz constant can be computed as the largest singular value. Indeed, the linear function depends only on the constant $k$ (which depends on the constant $l$ and the dimension $n$) from which we conclude that the Lipschitz constant of $NN_{I_n \to I_n^i}$ is $O(l^{\frac{1}{n}})$. □

## C PROOF OF PROPOSITION 3

Proof. It follows from Equations (2)–(4) that the sub-neural network $NN_{X_0 \to ZC_i}$ can be written as follows:

$$ZC_i\left(a_p, I_n^i\right) = NN_{X_0 \to ZC_i}\left(NN_{a_p \to X_0}(a_p), NN_{I_n \to I_n^i}(I_n)\right) = NN_{X_0 \to ZC_i}\left(\left(x_0^1, \ldots, x_0^{n_r}\right), \left(\underline{d}^i, \overline{d}^i\right)\right). \tag{27}$$

The indicator variable $ZC_i$ should be set to zero whenever all the roots $x_0^j$ lies outside the hyperrectangle $I_n^i(\underline{d}^i, \overline{d}^i)$. First note that a root $x_0^j$ lies outside $I_n^i(\underline{d}^i, \overline{d}^i)$ if and only if the following condition holds:

$$x_0^j \notin I_n^i\left(\underline{d}^i, \overline{d}^i\right) \iff \sum_{k=1}^{k=n} \left|x_{0,k}^j - \underline{d}_k^i\right| + \left|x_{0,k}^j - \overline{d}_k^i\right| - \sum_{k=1}^{n} \left(\overline{d}_k^i - \underline{d}_k^i\right) > 0 \tag{28}$$

where $x_{0,k}^j$, $\underline{d}_k^i$, and $\overline{d}_k^i$ are the $k$th element in the vectors $x_0^j$, $\underline{d}^i$ , and $\overline{d}^i$, respectively. Hence, the indicator variable $ZC_i$ should be set to zero whenever the following conditions hold:

$$ZC_i(a_p, I_n^i) = 0 \iff \max_{j \in \{1, \ldots, n_r\}} \left(\sum_{k=1}^{k=n} \left|x_{0,k}^j - \underline{d}_k^i\right| + \left|x_{0,k}^j - \overline{d}_k^i\right| - \sum_{k=1}^{n} \left(\overline{d}_k^i - \underline{d}_k^i\right)\right) = 0 \tag{29}$$

Before we compute the Lipschitz constant of $NN_{X_0 \to ZC_i}$ in Equation (29), we recall the following identities. Consider two functions $f(x)$ and $g(x)$ with Lipschitz constants $L_f$ and $L_g$, respectively. Then:

— The Lipschitz constant of $\max(f(x), g(x))$ is bounded by $L_f + L_g$.
— The Lipschitz constant of $f(x) + g(x)$ is bounded by $\max(L_f, L_g)$.

Now notice that $|x_{0,k}^j - \underline{d}_k^i| = max(x_{0,k}^j - \underline{d}_k^i, 0) + max(-x_{0,k}^j + \underline{d}_k^i, 0)$. Applying the identities above along with the fact that the Lipschitz constant of $x_{0,k}^j - \underline{d}_k^i$ is $O(1)$, we conclude that the Lipschitz constant of $|x_{0,k}^j - \underline{d}_k^i|$ is $O(1)$. Hence, the Lipschitz constant of $\sum_{k=1}^{k=n} |x_{0,k}^j - \underline{d}_k^i| + |x_{0,k}^j - \overline{d}_k^i| - \sum_{k=1}^{n} (\overline{d}_k^i - \underline{d}_k^i)$ is also $O(1)$. Finally, the Lipschitz constant of the right hand side of Equation (29) is $O(n_r)$. We conclude our proof by noticing that all the operators in Equation (29)—namely the absolute value, the max operator, summation, and checking the final value against a constant—can be implemented exactly using ReLU NNs [20] and hence the neural network $NN_{X_0 \to ZC_i}$ will also have a Lipschitz constant equal to $O(n_r)$. □

## D PROOF OF PROPOSITION 4

Proof. This result follows directly by noticing that $NN_{ZC\rightarrow L^+/L^-}$ can be computed as a linear function:

$$NN_{ZC\rightarrow L^+/L^-}(ZC_1,\ldots,ZC_l) = v\sum_{i=1}^{l}(1-ZC_i) \tag{30}$$

where $v$ is a constant that depends on the volume of the hyperrectangle $I_n$. Since $l$ is a constant, we conclude the result. □

## REFERENCES


[1] Bai Xue, Martin Fränzle and Naijun Zhan. 2018. Under-approximating reach sets for polynomial continuous systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. 51–60.

[2] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, P. Parrilo, M. Peet, and J. Jagt. 2021. SOSTOOLS version 4.00 sum of squares optimization toolbox for MATLAB. ArXivorg (2021). https://par.nsf.gov/biblio/10353822

[3] Jie An, Naijun Zhan, Xiaoshan Li, Miaomiao Zhang, and Wang Yi. 2018. Model checking bounded continuous-time extended linear duration invariants. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. 81–90.

[4] Stanley Bak, Hoang-Dung Tran, and Taylor T. Johnson. 2019. Numerical verification of affine systems with up to a billion dimensions. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 23–32.

[5] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940. Retrieved from https://arxiv.org/abs/1611.09940

[6] Mohamed Amin Ben Sassi, Romain Testylier, Thao Dang, and Antoine Girard. 2012. Reachability analysis of polynomial systems using linear programming relaxations. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis*. Springer, 137–151.

[7] Michele Boreale. 2018. Algorithms for exact and approximate linear abstractions of polynomial continuous systems. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. 207–216.

[8] Nicolas Brisebarre, Mioara Joldeş, Érik Martin-Dorel, Micaela Mayero, Jean-Michel Muller, Ioana Paşca, Laurence Rideau, and Laurent Théry. 2012. Rigorous polynomial approximation using Taylor models in Coq. In *Proceedings of the NASA Formal Methods Symposium*. Springer, 85–99.

[9] Christopher W. Brown. 2001. Improved projection for cylindrical algebraic decomposition. *Journal of Symbolic Computation* 32, 5 (2001), 447–465.

[10] Xin Chen. 2015. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. Ph. D. Dissertation. Fachgruppe Informatik, RWTH Aachen University.

[11] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 258–263.

[12] George E. Collins. 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern*. Lecture Notes in Computer Science, Springer, 134–183.

[13] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 337–340.

[14] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. Robustfill: Neural program learning under noisy I/O. In *Proceedings of the International Conference on Machine Learning*. PMLR, 990–998.

[15] Tommaso Dreossi. 2017. Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. 29–34.

[16] Bruno Dutertre. 2014. Yices 2.2. In *Proceedings of the International Conference on Computer Aided Verification*. Springer, 737–744.

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

[18] Matthew England and James H. Davenport. 2016. The complexity of cylindrical algebraic decomposition with respect to polynomial degree. In *Proceedings of the International Workshop on Computer Algebra in Scientific Computing*. Springer, 172–192.
[19] Rida T. Farouki and V. T. Rajan. 1987. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design* 4, 3 (1987), 191–216.
[20] James Ferlez and Yasser Shoukry. 2020. AReN: Assured ReLU NN architecture for model predictive control of LTI systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–11.
[21] Richard FitzHugh. 1961. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal* 1, 6 (1961), 445–466.
[22] Ioannis A. Fotiou, Philipp Rostalski, Pablo A. Parrilo, and Manfred Morari. 2006. Parametric optimization and optimal control using algebraic geometry methods. *International Journal of Control* 79, 11 (2006), 1340–1358.
[23] Jürgen Garloff. 1985. Convergent bounds for the range of multivariate polynomials. In *Proceedings of the International Symposium on Interval Mathematics*. Springer, 37–56.
[24] Jürgen Garloff and Andrew P. Smith. 2004. An improved method for the computation of affine lower bound functions for polynomials. In *Frontiers in Global Optimization. Nonconvex Optimization and Its Applications*. Springer, 135–144.
[25] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*.
[26] Donald Goldfarb and Ashok Idnani. 1983. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming* 27, 1 (1983), 1–33.
[27] H. Hong. 1990. An improvement of the projection operator in cylindrical algebraic decomposition. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '90)*. ACM, New York, NY, USA, 261–264. DOI: https://doi.org/10.1145/96877.96943
[28] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. 2016. Deepmath-deep sequence models for premise selection. In *Proceedings of the 30th Conference on Neural Information Processing Systems*. 2235–2243.
[29] Cezary Kaliszyk, François Chollet, and Christian Szegedy. 2017. Holstep: A machine learning dataset for higher-order logic theorem proving. *International Conference on Learning Representations*. https://openreview.net/forum?id=ryuxYmvel
[30] Niklas Kochdumper and Matthias Althoff. 2020. Reachability analysis for hybrid systems with nonlinear guard sets. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–10.
[31] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. 2012. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science Springer, 9–48.
[32] Livia Lestingi, Mehrnoosh Askarpour, Marcello M. Bersani, and Matteo Rossi. 2020. Formal verification of human-robot interaction in healthcare scenarios. In *Proceedings of the International Conference on Software Engineering and Formal Methods*. Springer, 303–324.
[33] Assia Mahboubi. 2006. Programming and certifying a CAD algorithm in the Coq system. In *Dagstuhl Seminar Proceedings*, Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy (Eds.). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
[34] Cplex, IBM ILOG. 2009. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.
[35] Tobia Marcucci and Russ Tedrake. 2019. Mixed-integer formulations for optimal control of piecewise-affine systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 230–239.
[36] Scott McCallum. 1998. An improved projection operation for cylindrical algebraic decomposition. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, Springer, 242–268.
[37] Bernard Mourrain and Jean Pascal Pavone. 2009. Subdivision methods for solving polynomial equations. *Journal of Symbolic Computation* 44, 3 (2009), 292–306.
[38] César Munoz and Anthony Narkawicz. 2013. Formalization of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning* 51, 2 (2013), 151–196.
[39] Cesar A. Munoz. 2015. Formal methods in air traffic management: The case of unmanned aircraft systems. In *Proceedings of the International Colloquium on Theoretical Aspects of Computing (ICTAC 2015)*.
[40] Anthony Narkawicz and César A. Munoz. 2012. Formal verification of conflict detection algorithms for arbitrary trajectories. *Reliable Computing* 17 (2012), 209–237.
[41] Stacy D. Nelson and Charles Pecheur. 2002. Formal verification for a next-generation space shuttle. In *Proceedings of the International Workshop on Formal Approaches to Agent-Based Systems*. Springer, 53–67.
[42] Maben Rabi. 2020. Piece-wise analytic trajectory computation for polytopic switching between stable affine systems. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–11.

[43] Shashwati Ray and P. S. V. Nataraj. 2012. A matrix method for efficient computation of Bernstein coefficients. *Reliable Computing* 17, 1 (2012), 40–71.

[44] Debayan Roy, Michael Balszun, Thomas Heurung, and Samarjit Chakraborty. 2018. Multi-domain coupling for automated synthesis of distributed cyber-physical systems. In *Proceedings of the 2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[45] Sadra Sadraddini and Russ Tedrake. 2020. Robust output feedback control with guaranteed constraint satisfaction. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. 1–10.

[46] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=HJMC_iA5tm

[47] Zuowei Shen. 2020. Deep network approximation characterized by number of neurons. *Communications in Computational Physics* 28, 5 (2020), 1768–1811. DOI: 10.4208/cicp.OA-2020-0149

[48] Yasser Shoukry, Michelle Chong, Masashi Wakaiki, Pierluigi Nuzzo, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia, Joao P. Hespanha, and Paulo Tabuada. 2018. SMT-based observer design for cyber-physical systems under sensor attacks. *ACM Transactions on Cyber-Physical Systems* 2, 1 (2018), 1–27.

[49] Andrew Paul Smith. 2009. Fast construction of constant bound functions for sparse polynomials. *Journal of Global Optimization* 43, 2 (2009), 445–458.

[50] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 147–156.

[51] Philippe Trébuchet, Bernard Mourrain, and Marta Abril Bucero. 2016. Border basis for polynomial system solving and optimization. *Mathematical Software–ICMS 2016: 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings 5*, Springer, 212–220.

[52] Lieven Vandenberghe. 2010. The CVXOPT linear and quadratic cone program solvers. Retrieved March 20, 2010 from http://cvxopt.org/documentation/coneprog.pdf.

[53] Abraham P. Vinod and Meeko M. K. Oishi. 2018. Scalable underapproximative verification of stochastic LTI systems using convexity and compactness. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (Part of CPS Week)*. 1–10.

[54] Bai Xue, Qiuye Wang, Naijun Zhan, and Martin Fränzle. 2019. Robust invariant sets generation for state-constrained perturbed polynomial systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 128–137.

[55] Wael Fatnassi and Yasser Shoukry. 2021. PolyAR: A highly parallelizable solver for polynomial inequality constraints using convex abstraction refinement. *IFAC-PapersOnLine* 54, 5 (2021), 43–48.

[56] Michael Zettler and Jürgen Garloff. 1998. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Transactions on Automatic Control* 43, 3 (1998), 425–431.

Received 27 February 2023; revised 19 July 2023; accepted 21 October 2023