NIL: Learning Nonlinear Interpolants*

Mingshuai Chen[1,2(✉)][0000−0001−9663−7441], Jian Wang[1,2][0000−0002−8840−5605], Jie An[3][0000−0001−9260−9697], Bohua Zhan[1,2(✉)][0000−0001−5377−9351], Deepak Kapur[4][0000−0003−2464−2895], and Naijun Zhan[1,2(✉)][0000−0003−3298−3817]

[1] State Key Lab. of Computer Science, Institute of Software, CAS, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
{chenms,bzhan,znj}@ios.ac.cn
[3] School of Software Engineering, Tongji University, Shanghai, China
[4] Department of Computer Science, University of New Mexico, Albuquerque, USA

Abstract. Nonlinear interpolants have been shown useful for the verification of programs and hybrid systems in contexts of theorem proving, model checking, abstract interpretation, etc. The underlying synthesis problem, however, is challenging and existing methods have limitations on the form of formulae to be interpolated. We leverage classification techniques with space transformations and kernel tricks as established in the realm of machine learning, and present a counterexample-guided method named NIL for synthesizing polynomial interpolants, thereby yielding a unified framework tackling the interpolation problem for the general quantifier-free theory of nonlinear arithmetic, possibly involving transcendental functions. We prove the soundness of NIL and propose sufficient conditions under which NIL is guaranteed to converge, i.e., the derived sequence of candidate interpolants converges to an actual interpolant, and is complete, namely the algorithm terminates by producing an interpolant if there exists one. The applicability and effectiveness of our technique are demonstrated experimentally on a collection of representative benchmarks from the literature, where in particular, our method suffices to address more interpolation tasks, including those with perturbations in parameters, and in many cases synthesizes simpler interpolants compared with existing approaches.

Keywords: Nonlinear Craig interpolant · Counterexample-guided learning · Program verification · Support vector machines (SVMs)

1 Introduction

Interpolation-based technique provides a powerful mechanism for local and modular reasoning, thereby improving scalability of various verification techniques, e.g., theorem proving, model checking and abstract interpretation, to name just a few. The study of interpolation was pioneered by Krajíček [27] and Pudlák [35] in connection with theorem proving, by McMillan [30] in the context of model checking, by Graf and Saïdi [17], McMillan [31] and Henzinger et al. [20] pertaining to abstraction like CEGAR [8], and by Wang et al. [24] in the context of learning-based invariant generation.

* This work has been supported through grants by NSFC under grant No. 61625206 and 61732001, by the CAS Pioneer Hundred Talents Program under grant No. Y9RC585036, and by the National Science Foundation Award DMS-1217054.

Developing efficient algorithms for generating interpolants for various theories and their combination has become an active research area, see e.g., [7, 25, 26, 31, 32, 36, 46].

Though established methods addressing interpolant generation for Presburger arithmetic, decidable fragments of first-order logic, theory of equality over uninterpreted functions (EUFs) as well as their combination have been extensively studied in the literature, there appears to be little work on synthesizing nonlinear interpolants. Dai et al. proposed an algorithm in [11] for generating interpolants for nonlinear polynomial inequalities based on the existence of a witness guaranteed by Stengle's Positivstellensatz [16] that can be computed using semi-definite programming (SDP). A major limitation of this method is that the two mutually contradictory formulas to be interpolated must share the same set of variables. Okudono et al. extended [11] in [33] to cater for the so-called sharper and simpler interpolants by developing a continuous fraction-based algorithm that rounds off numerical solutions. In [14], Gan et al. considered the interpolation for inequalities combined with EUFs by employing the hierarchical calculus framework proposed in [38] (and its extension [39]), while the inequalities are limited to be of the concave quadratic form. In [15], Gao and Zufferey transformed proof traces from δ-complete decision procedures into interpolants, composed of Boolean combinations of linear constraints, which can deal with certain transcendental functions beyond polynomials. The techniques of encoding interpolants as logical combinations of linear constraints, including [15], [28] and [37], however, yield potentially large interpolants (requiring even an infinite length in the worst case) and their usage thus becomes difficult in practical applications (cf. Example 1).

Interpolants can be viewed as classifiers that distinguish, in the context of program verification for instance, positive program states from negative ones (unreachable/error states) and consequently the state-of-the-art classification algorithms can be leveraged for synthesizing interpolants. The universal applicability of classification techniques substantially extends the scope of theories admitting interpolant generation. This idea was first employed by Sharma et al. in [37], which infers linear interpolants through hyperplane-classifiers generated by support vector machines (SVMs) [3,45] whilst handles superficial nonlinearities by assembling interpolants in the form purely of conjunctions (or dually, disjunctions) of linear half-spaces, which addresses only a limited category of formulae featuring nonlinearities. The learning-based paradigm has also been exploited in the context of nonlinear constraint solving, see e.g., [12].

In this paper, we present a classification-based learning method for the synthesis of polynomial interpolants for the quantifier-free theory of nonlinear arithmetic. Our approach is based on techniques of space transformations and kernel tricks pertinent to SVMs that have been well-developed in the realm of machine learning. Our method is described by an algorithm called NIL (and its several variants) that adopts the counterexample-guided inductive synthesis framework [22, 40]. We prove the soundness of NIL and propose sufficient conditions under which NIL is guaranteed to converge, that is, the derived sequence of classifiers (candidate interpolants) converges to an actual interpolant, and is complete, i.e., if an interpolant exists, the method terminates with an actual interpolant. In contrast to related work on generation of nonlinear interpolants, which restrict the input formulae, our technique provides a uniform framework, tackling the interpolation problem for the general quantifier-free theory of

nonlinear arithmetic, possibly involving transcendental functions. The applicability and effectiveness of NIL are demonstrated experimentally on a collection of representative benchmarks from the literature; as is evident from experimental results, our method is able to address more demands on the nature of interpolants, including those with perturbations in parameters (due to the robustness inherited from SVMs); in many cases, it synthesizes simpler interpolants compared with other approaches, as shown by the following example.

Example 1 ([15]). Consider two mutually contradictory inequalities $\phi \mathrel{\widehat{=}} y \geq x^2$ and $\psi \mathrel{\widehat{=}} y \leq -\cos(x)+0.8$. Our NIL algorithm constructs a single polynomial inequality $I \mathrel{\widehat{=}} 15x^2 < 4+20y$ as the interpolant, namely, $\phi \models I$ and $I \wedge \psi$ is unsatisfiable; while the interpolant generated by the approach in [15], only when provided with sufficiently large finite domains, e.g., $x \in [-\pi, \pi]$ and $y \in [-0.2, \pi^2]$, is $y > 1.8 \vee (0.59 \leq y \leq 1.8 \wedge -1.35 \leq x \leq 1.35) \vee (0.09 \leq y < 0.59 \wedge -0.77 \leq x \leq 0.77) \vee (y \geq 0 \wedge -0.3 \leq x \leq 0.3)$. As will be discussed later, we do not need to provide a priori information to our algorithm such as bounds on variables.

The rest of the paper is organized as follows. Sect. 2 introduces some preliminaries on Craig interpolants and SVMs. In Sect. 3, we present the NIL algorithm dedicated to synthesizing nonlinear interpolants, followed by the analysis of its soundness, conditional completeness and convergence in Sect. 4. Sect. 5 reports several implementation issues and experimental results on a collection of benchmarks (with the robustness discussed in Sect. 6). The paper is then concluded in Sect. 7.

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$ be the set of natural, rational and real numbers, respectively. We denote by $\mathbb{R}[\mathbf{x}]$ the polynomial ring over $\mathbb{R}$ with variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, and $\|\mathbf{x}\|$ denotes the ℓ^2-norm [4]. For a set $X \subseteq \mathbb{R}^n$, its convex hull is denoted by $\mathtt{conv}(X)$. For $\boldsymbol{x}, \boldsymbol{x}' \in X$, $\mathtt{dist}(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|$ denotes the Euclidean distance between two points, which generalizes to $\mathtt{dist}(\boldsymbol{x}, X') = \min_{\boldsymbol{x}' \in X'} \mathtt{dist}(\boldsymbol{x}, \boldsymbol{x}')$. Given $\delta \geq 0$, define $\mathcal{B}(\boldsymbol{x}, \delta) = \{\boldsymbol{x}' \in \mathbb{R}^n \mid \|\boldsymbol{x}' - \boldsymbol{x}\| \leq \delta\}$ as the closed ball of radius δ centered at $\boldsymbol{x}$. Consider the quantifier-free fragment of a first-order theory of polynomials over the reals, denoted by $\mathcal{T}_P$, in which a formula φ is of the form

$$\varphi \mathrel{\widehat{=}} p(\mathbf{x}) \diamond 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

where $p(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ and $\diamond \in \{<, >, \leq, \geq, =\}$. A natural extension of our method to cater for more general nonlinearities involving transcendental functions will be demonstrated in subsequent sections. In the sequel, we use $\bot$ to stand for *false* and $\top$ for *true*. Let $\mathbb{R}[\mathbf{x}]_m$ consist of all polynomials $p(\mathbf{x})$ of degree $\leq m \in \mathbb{N}$. We abuse the notation $\varphi \in \mathbb{R}[\mathbf{x}]_m$ to abbreviate $\varphi \mathrel{\widehat{=}} p(\mathbf{x}) \diamond 0$ and $p(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]_m$ if no ambiguity arises.

Given formulas ϕ and ψ in a theory $\mathcal{T}$, ϕ is *valid* w.r.t. $\mathcal{T}$, written as $\models_{\mathcal{T}} \phi$, iff ϕ is true in all models of $\mathcal{T}$; ϕ *entails* ψ w.r.t. $\mathcal{T}$, written as $\phi \models_{\mathcal{T}} \psi$, iff every model of $\mathcal{T}$ that makes ϕ true makes ψ also true; ϕ is *satisfiable* w.r.t. $\mathcal{T}$, iff there is a model of $\mathcal{T}$ in which ϕ is true; otherwise *unsatisfiable*. It follows that ϕ is unsatisfiable iff $\phi \models_{\mathcal{T}} \bot$. The set of all the models that make ϕ true is denoted by $[\![\phi]\!]_{\mathcal{T}}$.

2.1 Craig Interpolant

Craig showed in [10] that given two formulas ϕ and ψ in a first-order logic $\mathcal{T}$ s.t. $\phi \models_{\mathcal{T}} \psi$, there always exists an *interpolant* I over the common symbols of ϕ and ψ s.t. $\phi \models_{\mathcal{T}} I$ and $I \models_{\mathcal{T}} \psi$. In the verification literature, this terminology has been abused by [31], which defined an interpolant over the common symbols of ϕ and ψ as

Definition 1 (Interpolant). *Given ϕ and ψ in a theory $\mathcal{T}$ s.t. $\phi \wedge \psi \models_{\mathcal{T}} \bot$, a formula I is a* (reverse) interpolant *of ϕ and ψ if* (i) $\phi \models_{\mathcal{T}} I$; (ii) $I \wedge \psi \models_{\mathcal{T}} \bot$; *and* (iii) I *contains only common symbols shared by ϕ and ψ.*

It is immediately obvious that $\phi \models_{\mathcal{T}} \psi$ iff $\phi \wedge \neg\psi \models_{\mathcal{T}} \bot$, namely, I is an interpolant of ϕ and ψ iff I is a reverse interpolant in McMillan's sense of ϕ and $\neg\psi$. We follow McMillan in continuing to abuse the terminology.

2.2 Support Vector Machines

In machine learning, support vector machines [3,45] are supervised learning models for effective classification based on convex optimization. In a binary setting, we are given a training dataset $X = X^+ \uplus X^-$ of n sample points $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$, and y_i is either 1, indicating a positive sample $\boldsymbol{x}_i \in X^+$, or -1, indicating a negative one in X^-. The goal of classification here is to find a potential hyperplane (a.k.a. *linear classifier*) to separate the positive samples from the negative ones. There however might be various or even infinite number of separating hyperplanes, and an SVM aims to construct a separating hyperplane that yields the largest distance (so-called *functional margin*) to the nearest positive and negative samples. Such a classification hyperplane is called the *optimal-margin classifier* while the samples closest to it are called the *support vectors*.

Linear SVMs. Assume that X^+ and X^- are *linearly separable*, meaning that there exists a *linear separating hyperplane* $\boldsymbol{w}^{\mathrm{T}}\mathbf{x} + b = 0$ such that $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) > 0$, for all $(\boldsymbol{x}_i, y_i) \in X$. Then the functional margin can be formulated as

$$\gamma \;\widehat{=}\; 2\min_{1\le i\le n} 1/\|\boldsymbol{w}\||\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b|.$$

Linear SVMs are committed to finding appropriate parameters $(\boldsymbol{w}, b)$ that maximize the functional margin while adhering to the constraints of separability, which reduces equivalently to the following convex quadratic optimization problem [2] that can be efficiently solved by off-the-shelf packages for quadratic programming:

$$\underset{\boldsymbol{w},b}{\text{minimize}} \;\; \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} \quad \text{subject to} \;\; y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \ge 1, \;\; i = 1, 2, \ldots, n. \tag{1}$$

Lemma 1 (Correctness of SVMs [37]). *Given positive samples X^+ which are linearly separable from negative samples X^-, SVMs produce, under computations of infinite precision, a half-space h s.t. $\forall \boldsymbol{x} \in X^+.\ h(\boldsymbol{x}) > 0$ and $\forall \boldsymbol{x} \in X^-.\ h(\boldsymbol{x}) < 0$.*

Corollary 1 (Separation of Convex Hulls [1]). *The half-space h in Lemma 1 satisfies that $\forall \boldsymbol{x} \in \mathtt{conv}(X^+).\ h(\boldsymbol{x}) > 0$ and $\forall \boldsymbol{x} \in \mathtt{conv}(X^-).\ h(\boldsymbol{x}) < 0$.*

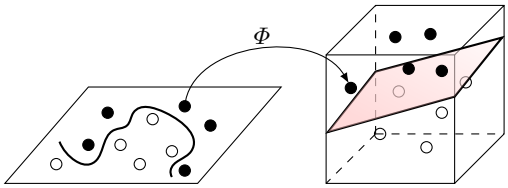


Fig. 1: Mapping from a two-dimensional input space into a three-dimensional feature space with linear separation thereof.

Nonlinear SVMs. When ϕ and ψ are formulas over nonlinear arithmetic, often after sampling X, it is not possible to find a linearly separable hyperplane in the common variables. However, a nonlinear surface that can be described as a linear hyperplane in the space of monomials of bounded degree may separate X^+ and X^-. The above construction is generalized by introducing a transformation from $\mathbb{R}^d$ to $\mathbb{R}^{\tilde{d}}$, the vector space of monomials in the common variables up to some bounded degree, with $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) \geq 1$ in (1) replaced by $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}_i) + b) \geq 1$, where $\boldsymbol{\Phi}$ is a linear expression in monomials in the common variables up to a bounded degree. Here, the vectors $\boldsymbol{\Phi}(\mathbf{x})$ span the *feature space*.

Consider the Lagrangian dual [3] of the modified optimization problem:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \;& \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \boldsymbol{\Phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}_j) - \sum_{i=1}^{n}\alpha_i \\ \text{subject to} \;& \sum_{i=1}^{n}\alpha_i y_i = 0, \text{ and } \alpha_i \geq 0 \text{ for } i = 1, 2, \ldots, n. \end{aligned}$$

A *kernel function* $\kappa\colon \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is defined as $\kappa(\mathbf{x}, \mathbf{x}') \;\widehat{=}\; \boldsymbol{\Phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\Phi}(\mathbf{x}')$. The introduction of the dual problem and the kernel function [3] reduces the computational complexity essentially from $\mathcal{O}(\tilde{d})$ down to $\mathcal{O}(d)$. For the sake of post-verifying a candidate interpolant given by SVMs, we adopt an inhomogeneous polynomial kernel function of the form

$$\kappa(\mathbf{x}, \mathbf{x}') \;\widehat{=}\; (\beta\mathbf{x}^{\mathrm{T}}\mathbf{x}' + \theta)^m,$$

where m is the polynomial degree describing complexity of the feature space, $\theta \geq 0$ is a parameter trading off the influence of higher-order versus lower-order terms in the polynomial, and β is a scalar parameter. Henceforth, the optimal-margin classifier (if there exists one) can be derived as $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\mathbf{x}) = \sum_{i=1}^{n}\alpha_i\kappa(\boldsymbol{x}_i, \mathbf{x}) = 0$, with $\boldsymbol{x}_i$ being a support vector iff $\alpha_i > 0$. In practice, usually a large amount of α_is turn out to be zero and this leads to a simple representation of a classifier. Fig. 1 illustrates the intuitive idea of the transformation from the original input space to the feature space. We will show in the sequel that the resulting classifier can be viewed as a candidate interpolant, while its optimal-margin feature contributes to a certain "medium" logical strength of the interpolant, which is thus robust to perturbations (in the feature space) in the formulae to be interpolated.

3 Learning Interpolants

In this section, we present the NIL algorithm for synthesizing nontrivial (reverse) Craig interpolants for the quantifier-free theory of nonlinear arithmetic. It takes as input a

pair $\langle \phi, \psi \rangle$ of formulas in $\mathcal{T}_P$ as well as a positive integer m, and aims to generate an interpolant I of maximum degree m, i.e., $I \in \mathbb{R}[\mathbf{x}]_m$, if it exists, such that $\phi \models_{\mathcal{T}_P} I$ and $I \wedge \psi \models_{\mathcal{T}_P} \bot$. Here, $\langle \phi, \psi \rangle$ can be decorated as $\langle \phi(\mathbf{x}, \mathbf{y}), \psi(\mathbf{x}, \mathbf{z}) \rangle$ with variables involved in the predicates, and thus $\mathbf{x}$ denotes variables that are common to ϕ and ψ. In the sequel, we drop the subscript $\mathcal{T}_P$ in $\models_{\mathcal{T}_P}$ and $[\![\cdot]\!]_{\mathcal{T}_P}$ wherever the context is unambiguous.

Due to the decidability of the first-order theory of real-closed fields established by Tarski [44], $\mathcal{T}_P$ admits *quantifier elimination* (QE). This means that the satisfiability of any formula in $\mathcal{T}_P$ can be decided (in doubly exponential time in the number of variables for the worst case). If the formula is satisfiable, models satisfying the formula can also be constructed algorithmically (following the same time complexity). Though the introduction of general forms of transcendental functions renders the underlying theory undecidable, there does exist certain extension of $\mathcal{T}_P$ with transcendental functions (involving exponential functions, logarithms and trigonometric functions), e.g. that identified by Strzeboński in [43] and references therein, which still admits QE. This allows a straightforward extension of NIL to such a decidable fragment involving transcendental functions. Specifically, the decidability remains when the transcendental functions involved are real univariate exp-log functions [41] or tame elementary functions [42] which admit a real root isolation algorithm.

3.1 The Core Algorithm

The basic idea of NIL is to view interpolants as classifiers and use SVMs with the kernel trick to perform effective classification. The algorithm is based on the sampling-guessing-refining technique: in each iteration, it is fed with a classifier (candidate interpolant) for a finite set of sample points from $[\![\phi]\!]$ and $[\![\psi]\!]$ (line 5), and verify the candidate (line 10) by checking the entailment problem that defines an interpolant (as in Def. 1). If the verification succeeds, the interpolant is returned as the final result. Otherwise, a set of counterexamples is obtained (line 13 and 14) as new sample points to further refine the classifier. In what follows, we explain the steps of the interpolation procedure in more detail.

Initial sampling. The algorithm begins by checking the satisfiability of $\phi \wedge \psi$. If the formula is satisfiable, it is then impossible to find an interpolant, and the algorithm stops declaring no interpolant exists.

Next, the algorithm attempts to sample points from both $[\![\phi]\!]$ and $[\![\psi]\!]$. This initial sampling stage can usually be done efficiently using the Monte Carlo method, e.g. by (uniformly) scattering a number of random points over certain bounded range and then selecting those fall in $[\![\phi]\!]$ and $[\![\psi]\!]$ respectively. However, this method fails when one or both of the predicates is very unlikely to be satisfied. One common example is when the predicate involves equalities. For such situations, solving the satisfiability problem using QE is guaranted to succeed in producing the sample points.

To meet the condition that the generated interpolant can only involve symbols that are common to ϕ and ψ, we can project the points sampled from $[\![\phi]\!]$ (resp. $[\![\psi]\!]$) to the space of $\mathbf{x}$ by simply dropping the components that pertain to $\mathbf{y}$ (resp. $\mathbf{z}$) and thereby obtain sample points in X^+ (resp. X^-).

Algorithm NIL: Learning nonlinear interpolant

input : ϕ and ψ in $\mathcal{T}_P$ over common variables $\mathbf{x}$;
m, degree of the polynomial kernel, and hence maximum degree of the interpolant.

```
/* checking unsatisfiability */
1  if φ ∧ ψ ⊭ ⊥ then
       /* no interpolant exists */
2      abort;
/* generating initial sample points */
3  ⟨X⁺, X⁻⟩ ← Sampling(φ, ψ);
/* counterexample-guided learning */
4  while ⊤ do
       /* generating a classifier by SVMs */
5      C ← SVM(X⁺, X⁻, m);
       /* checking classification result */
6      if C = Failed then
           /* no interpolant exists in ℝ[x]_m */
7          abort;
       /* classifier as candidate interpolant */
8      else
9          I ← C;
       /* valid interpolant found */
10     if φ ⊨ I and I ∧ ψ ⊨ ⊥ then
11         return I;
       /* adding counterexamples */
12     else
13         X⁺ ← X⁺ ⊎ FindInstance(φ ∧ ¬I);
14         X⁻ ← X⁻ ⊎ FindInstance(I ∧ ψ);
```

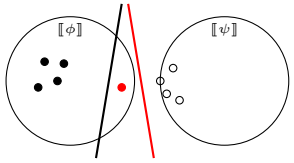


Fig. 2: In NIL, a candidate interpolant (black line as its boundary) is refined to an actual one (red line as its boundary) by adding a counterexample (red dot).

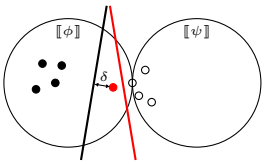


Fig. 3: In NIL$_\delta$, a counterexample (red dot) stays at least a distance of δ away from the candidate interpolant (black line as its boundary) to be refined, leading to an interpolant (red line as its boundary) with tolerance δ.

Entailment checking. The correctness of SVM given in Lemma 1 only guarantees that the candidate interpolant separates the finite set of points sampled from $[\![\phi]\!]$ and $[\![\psi]\!]$, not necessarily the entirety of the two sets. Hence, post-verification by checking the entailment problem (line 10) is needed for the candidate to be claimed as an interpolant of ϕ and ψ. This can be achieved by solving the equivalent QE problems $\forall \mathbf{x}.\, \phi(\mathbf{x},\mathbf{y})|_{\mathbf{x}} \implies I(\mathbf{x})$ and $\forall \mathbf{x}.\, I(\mathbf{x}) \wedge \psi(\mathbf{x},\mathbf{z})|_{\mathbf{x}} \implies \bot$, where $\cdot|_{\mathbf{x}}$ is the *projection* to the common space over $\mathbf{x}$. The candidate will be returned as an actual interpolant if both formulae reduce to $\top$ after eliminating the universal quantifiers. The satisfiability checking at line 1 can be solved analogously. Granted, the entailment checking can also be encoded in SMT techniques by asking the satisfiability of the negation of the universally quantified predicates, however, limitations of current SMT solvers in nonlinear arithmetic hinders them from being practically used in our framework, as demonstrated later in Sect. 5.

Counterexample generation. If a candidate interpolant cannot be verified as an actual one, then at least one witness can be found as a counterexample to that candidate, which can be added to the set of sample points in the next iteration to refine further candidates (cf. Fig. 2). Multiple counterexamples can be obtained at a time thereby effectively reducing the number of future iterations.

In general, we have little control over which counterexample will be returned by QE. In the worst case, the counterexample can lie almost exactly on the hyperplane found by SVM. This poses issues for the termination of the algorithm. We will address this theoretical issue by slightly modifying the algorithm, as explained in Sect. 3.3 and 4.

3.2 Comparison with the Naïve QE-Based Method

Simply performing QE on $\exists \mathbf{y}.\ \phi(\mathbf{x}, \mathbf{y})$ yields already an interpolant for mutually contradictory ϕ and ψ. Such an interpolant is actually the *strongest* in the sense of [13], which presents an ordered family of interpolation systems due to the logical strength of the synthesized interpolants. Dually, the negation of the result when performing QE over $\exists \mathbf{z}.\ \psi(\mathbf{x}, \mathbf{z})$ is the *weakest* interpolant. However, as argued by D'Silva et al. in [13], a good interpolant (approximation of ϕ or ψ) –when computing invariants of transition systems using interpolation-based model checking– should be coarse enough to enable rapid convergence but strong enough to be contained within the weakest inductive invariant. In contrast, the advantages of NIL are two-fold: first, it produces better interpolants (in the above sense) featuring "medium" strength (due to the way optimal-margin classifier is defined) which are thus more effective in practical use and furthermore resilient to perturbations in ϕ and ψ (i.e., the robustness shown later in Sect. 6); second, NIL always returns a single polynomial inequality as the interpolant which is often simpler than that derived from the naïve QE-based method, where the direct projection of $\phi(\mathbf{x}, \mathbf{y})$ onto the common space over $\mathbf{x}$ can be as complex as the original ϕ.

These issues can be avoided by combining this method with a template-based approach, which in turn introduces fresh quantifiers over unknown parameters to be eliminated. Note that in NIL the candidate interpolants $I \in \mathbb{R}[\mathbf{x}]_m$ under verification are polynomials without unknown parameters, and therefore, in contrast to performing QE over an assumed template, the learning-based technique can practically generate polynomial interpolants of higher degrees (with acceptable rounds of iterations). For example, NIL is able to synthesize an interpolant of degree 7 over 2 variables (depicted later in Fig. 4(b)), which would require a polynomial template with $\binom{7+2}{2} = 36$ unknown parameters that goes far beyond the capability of QE procedures.

On the other hand, performing QE within every iteration of the learning process, for entailment checking and generating counterexamples, limits the efficiency of the proposed method, thereby confining NIL currently to applications only of small scales. Potential solutions to the efficiency bottleneck will be discussed in Sect. 5.

3.3 Variants of NIL

While the above basic algorithm is already effective in practice (as demonstrated in Sect. 5), it is guaranteed to terminate only when there is an interpolant with positive functional margin between $[\![\phi]\!]$ and $[\![\psi]\!]$. In this section, we present two variants of the algorithm that have nicer theoretical properties in cases where the two sets are only separated by an interpolant with zero functional margin, e.g., cases where $[\![\phi]\!]$ and $[\![\psi]\!]$ share parallel, adjacent, or even coincident boundaries.

Entailment checking with tolerance δ. When performing entailment checking for a candidate interpolant I, instead of using, e.g., the formula $p(\mathbf{x}) \geq 0$ for I, we can introduce a tolerance of δ. That is, we check the satisfiability of $\phi \wedge (p(\mathbf{x}) < -\delta)$ and $(p(\mathbf{x}) \geq \delta) \wedge \psi$ instead of the original $\phi \wedge (p(\mathbf{x}) < 0)$ and $(p(\mathbf{x}) \geq 0) \wedge \psi$. This means that a candidate that is an interpolant "up to a tolerance of δ" will be returned as a true interpolant, which may be acceptable in some applications. If the candidate interpolant is still not verified, the counterexample is guaranteed to be at least a distance

of δ away from the separating hyperplane. Note the distance δ is taken in the feature space $\mathbb{R}^{\tilde{d}}$, not in the original space. We let NIL$_\delta(\phi,\psi,m)$ denote the version of NIL with this modification (cf. Fig. 3). In the next section, we show NIL$_\delta(\phi,\psi,m)$ terminates as long as $[\![\phi]\!]$ and $[\![\psi]\!]$ are bounded, including the case where they are separated only by interpolants of functional margin zero.

Varying tolerance during the execution. A further refinement of the algorithm can be made by varying the tolerance δ during the execution. We also introduce a bounding box B of the varying size to handle unbounded cases. Define algorithm NIL$^*_{\delta,B}(\phi,\psi,m)$ as follows. Let $\delta_1 = \delta$ and $B_1 = B$. For each iteration i, execute the core algorithm, except that the counterexample must be a distance of at least δ_i away from the separating boundary, and have absolute value in each dimension at most B (both in $\mathbb{R}^{\tilde{d}}$). After the termination of iteration i, begin iteration $i+1$ with $\delta_{i+1} = \delta_i/2$ and $B_{i+1} = 2B_i$. This continues until an interpolant is found or until a pre-specified cutoff. For any $[\![\phi]\!]$ and $[\![\psi]\!]$ (without the boundedness condition), this variant of the algorithm *converges* to an interpolant in the limit, which will be made precise in the next section.

4 Soundness, Completeness and Convergence

In this section, we present theoretical results obtained for the basic NIL algorithm and its variants. Proofs are available in the appendix of [6].

First, the basic algorithm is sound, as captured by Theorem 1.

Theorem 1 (Soundness of NIL). *NIL(ϕ,ψ,m) terminates and returns I if and only if I is an interpolant in $\mathbb{R}[\mathbf{x}]_m$ of ϕ and ψ.*

Under certain conditions, the algorithm is also terminating (and hence complete). We prove two such situations below. In both cases, we require boundedness of the two sets that we want to separate. In the first case, there exists an interpolant with positive functional margin between the two sets.

Theorem 2 (Conditional Completeness of NIL). *If $[\![\phi]\!]$ and $[\![\psi]\!]$ are bounded and there exists an interpolant in $\mathbb{R}[\mathbf{x}]_m$ of ϕ and ψ with positive functional margin γ when mapped to $\mathbb{R}^{\tilde{d}}$, then NIL(ϕ,ψ,m) terminates and returns an interpolant I of ϕ and ψ.*

The standard algorithm is not guaranteed to terminate when $[\![\phi]\!]$ and $[\![\psi]\!]$ are only separated by interpolants of functional margin zero. However, the modified algorithm NIL$_\delta(\phi,\psi,m)$ does terminate (with the cost that the resulting answer is an interpolant with tolerance δ).

Theorem 3 (Completeness of NIL$_\delta$ with zero margin). *If $[\![\phi]\!]$ and $[\![\psi]\!]$ are bounded, and $\delta > 0$, then NIL$_\delta$(ϕ,ψ,m) terminates. It returns an interpolant I of ϕ and ψ with tolerance δ whenever such an interpolant exists.*

By iteratively decreasing δ during the execution of the algorithm, as well as introducing an iteratively increasing bounding box, as in NIL$^*_{\delta,B}(\phi,\psi,m)$, we can obtain more and more accurate candidate interpolants. We now show that this algorithm *converges* to an interpolant without restrictions on ϕ and ψ. We first make this convergence property precise in the following definition.

Definition 2 (Convergence of a sequence of equations to an interpolant). *Given two sets $[\![\phi]\!]$ and $[\![\psi]\!]$ that we want to separate, and an infinite sequence of equations $I_1, I_2, \ldots$, we say the sequence I_n converges to an interpolant of ϕ and ψ if, for each point p in the interior of $[\![\phi]\!]$ or $[\![\psi]\!]$, there exists some integer K_p such that I_k classifies p correctly for all $k \geq K_p$.*

Theorem 4 (Convergence of NIL$^*_{\delta,B}$). *Given two regions $[\![\phi]\!]$ and $[\![\psi]\!]$. Suppose there exists an interpolant of ϕ and ψ, then the infinite sequence of candidates produced by NIL$^*_{\delta,B}(\phi, \psi, m)$ converges to an interpolant in the sense of Definition 2.*

5 Implementation and Experiments

5.1 Implementation Issues

We have implemented the core algorithm NIL as a prototype[1] in Wolfram Mathematica with LIBSVM [5] being integrated as an engine to perform SVM classifications. Despite featuring no completeness for adjacent $[\![\phi]\!]$ and $[\![\psi]\!]$ nor convergence for unbounded $[\![\phi]\!]$ or $[\![\psi]\!]$, the standard NIL algorithm yields already promising results as shown later in the experiments. Key Mathematica functions that are utilized include REDUCE, for entailment checking, e.g., the unsatisfiability checking of $\phi \wedge \psi$ and the post-verification of a candidate interpolant, and FINDINSTANCE, for generating counterexamples and sampling initial points (when the random sampling strategy fails). The REDUCE command implements a decision procedure for $\mathcal{T}_P$ and its appropriate extension to catering for transcendental functions (cf. [43]) based on *cylindrical algebraic decomposition* (CAD), due to Collins [9]. The underlying quantifier-elimination procedure, albeit inducing rather high computation complexity, cannot in practice be replaced by SMT-solving techniques (by checking the negation of a universally quantified predicate) as in the linear arithmetic. For instance, the off-the-shelf SMT solver Z3 fails to accomplish our tasks particularly when the coefficients occurring in the entailment problem to be checked get larger[2].

Numerical errors and rounding. LIBSVM conducts floating-point computations for solving the optimization problems induced by SVMs and consequently yields numerical errors occurring in the candidate interpolants. Such numerical errors may block an otherwise valid interpolant from being verified as an actual one and additionally bring down the simplicity and thereby the effectiveness of the synthesized interpolant, thus not very often proving humans with clear-cut understanding. This is a common issue for approaches that reduce the interpolation problem to numerical solving techniques, e.g. SDP solvers exploited in [11, 14, 33], while an established method to tackle it is known as *rational recovery* [29, 47], which retrieves the nearest rational number from the continued fraction representation of its floating-point approximation at any given accuracy (see e.g. [47] for theoretical guarantees and [33] for applications in interpolation). The algorithm implementing rational recovery has been integrated in our implementation and the consequent benefits are two-fold: (i) NIL can now cope with interpolation tasks

[1] Available at `http://lcs.ios.ac.cn/~chenms/tools/NIL.tar.bz2`

[2] As can be also observed at `https://github.com/Z3Prover/z3/issues/1765`

where only exact coefficients suffice to constitute an actual interpolant while any numerical error therein will render the interpolant invalid, e.g., cases where $[\![\phi]\!]$ and $[\![\psi]\!]$ share parallel, adjacent, or even coincident boundaries, as demonstrated later by examples with ID 10–17 in Table 1; (ii) rationalizing coefficients moreover facilitates simplifications over all of the candidate interpolants and therefore practically accelerating the entailment checking and counterexample generation processes, which in return yields simpler interpolants, as shown in Table 2 in the following section.

5.2 Benchmark and Experimental Results

Table 1 collects a group of benchmark examples from the literature on synthesizing nonlinear interpolants as well as some geometrically contrived ones. All of the experiments have been evaluated on a 3.6GHz Intel Core-i7 processor with 8GB RAM running 64-bit Ubuntu 16.04.

In Table 1, we group the set of examples into four categories comprising 20 cases in total. For each example, **ID** numbers the case, $\boldsymbol{\phi}$, $\boldsymbol{\psi}$ and $\boldsymbol{I}$ represent the two formulas to be interpolated and the synthesized interpolant by our method respectively, while **Time/s** indicates the total time in seconds for interpolation. The categories are described as follows, and the visualization of a selected set of typical examples thereof is further depicted in Fig. 4.

Cat. I: with/without rounding. This category includes 9 cases, for which our method generates the polynomial interpolants correctly with or without the rounding operation.

Cat. II: with rounding. For cases 10 to 17 in this category, where $[\![\phi]\!]$ and $[\![\psi]\!]$ share parallel, adjacent, or even coincident boundaries, our method produces interpolants successfully with the rouding process based on rational recovery.

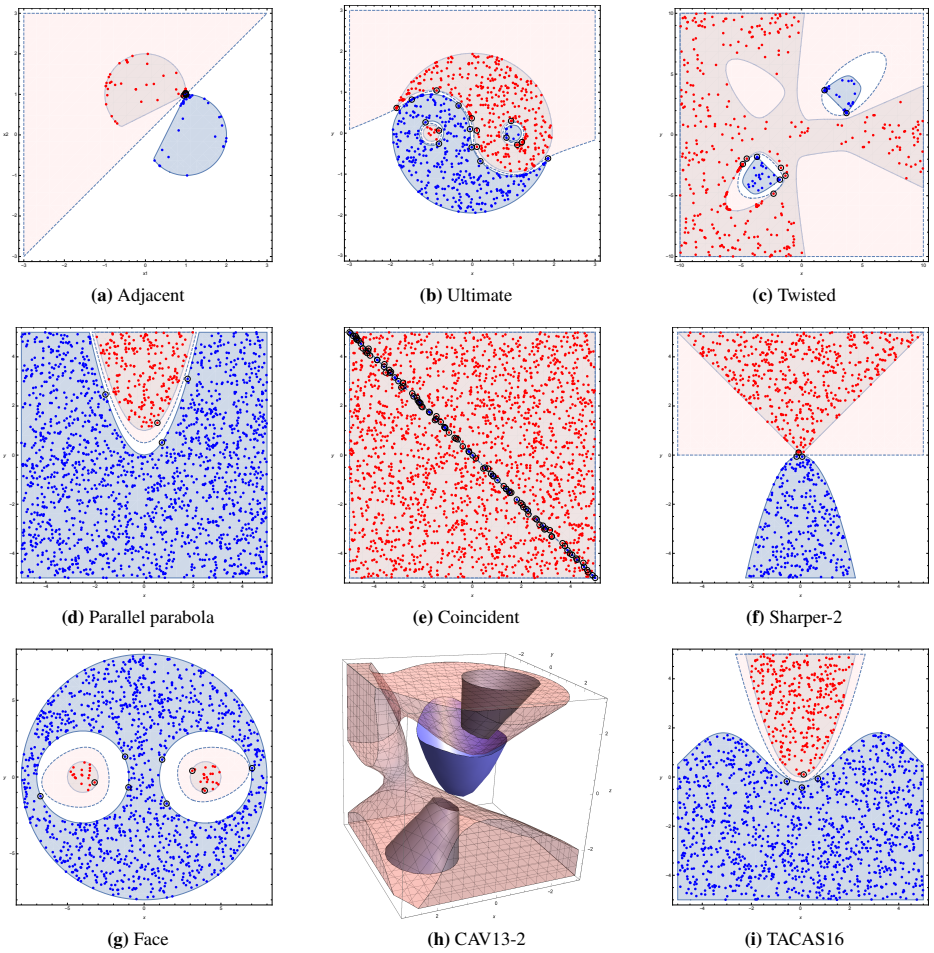
Cat. III: beyond polynomials. This category encloses two cases beyond the theory $\mathcal{T}_P$ of polynomials: for case 18, a verified polynomial interpolant is obtained in spite of the transcendental term in ψ; while for case 19, the SVM classification fails since $[\![\phi]\!]$ and $[\![\psi]\!]$ are not linearly separable in any finite-dimensional feature space and hence no polynomial interpolant exists for this example. Note that our counterexample-guided learning framework admits a straightforward extension to a decidable fragment of more general nonlinear theories involving transcendental functions, as investigated in [43].

Cat. IV: unbalanced. The case 20, called Unbalanced, instantiates a particular scenario where ϕ and ψ have extraordinary "unbalanced" number of models that make them true respectively. For this example, there are an infinite number of models satisfying ϕ yet one single model (i.e., $x = 0$) satisfying ψ. The training process in SVMs may fail when encountering extremely unbalanced number of positive/negative samples. This is solved by specifying a *weight* factor for the positive set of samples as the number of negative ones, and dually for the other way around, to balance biased number of training samples before triggering the classification. Such a balancing trick is supported in LIBSVM.

Remark that examples named CAV13-1/3/4 are taken from [11] (and the latter two originally from [28] and [18] respectively), where interpolation is applied to discovering *inductive invariants* in the verification of programs and hybrid systems. For instance, CAV13-3 is a program fragment describing an accelerating car and the synthesized interpolant by NIL suffices to prove the safety property of the car concerning its velocity.

Table 1: Benckmark examples for synthesizing nonlinear interpolants.

Category	ID	Name	ϕ	ψ	I	Time/s
with/without rounding	1	Dummy	$x < -1$	$x \geq 1$	$x < 0$	0.11
	2	Necklace	$y - x^2 - 1 = 0$	$y + x^2 + 1 = 0$	$-y < 0$	0.21
	3	Face	$(x+4)^2 + y^2 - 1 \leq 0 \vee$ $(x-4)^2 + y^2 - 1 \leq 0$	$x^2 + y^2 - 64 \leq 0 \wedge$ $(x+4)^2 + y^2 - 9 \geq 0 \wedge$ $(x-4)^2 + y^2 - 9 \geq 0$	$\frac{x^4}{223} - \frac{x^3 y}{356} + x^2(\frac{y^2}{45} - \frac{y}{170} - \frac{2}{9}) +$ $x(\frac{y^3}{89} + \frac{y^2}{68} - \frac{y}{74} - \frac{1}{55}) + \frac{y^4}{146} +$ $\frac{y^3}{95} + \frac{y^2}{37} + \frac{y}{366} + 1 < 0$	0.33
	4	Twisted	$x^2 - 2xy^2 + 3xz - y^2$ $- yz + z^2 - 1 \geq 0 \wedge$ $\frac{1}{120}\left(-x^6 - y^6\right) + x^2 z^2 -$ $x^2 + \frac{1}{6}\left(x^4 + 2x^2 y^2 + y^4\right) +$ $y^2 z^2 - y^2 - 4 \leq 0$	$w^2 + 4(x-y)^4 + (x+y)^2 - 80 \leq 0 \wedge$ $-w^2(x-y)^4 + 100(x+y)^2 - 3000 \geq 0$	$-\frac{x^4}{160} + x^3\left(\frac{y}{170} - \frac{1}{113}\right) + x^2\left(-\frac{y^2}{225} + \frac{y}{76} + \frac{2}{27}\right) +$ $x\left(\frac{y^3}{259} + \frac{y^2}{63} + \frac{5y}{51} - \frac{1}{316}\right) - \frac{y^4}{183} - \frac{y^3}{94} + \frac{y^2}{14} + \frac{y}{255} - 1 < 0$	140.62
	5	Ultimate	$(x^2 + y^2 - 3.8025 \leq 0 \wedge y \geq 0 \vee$ $(x-1)^2 + y^2 - 0.9025 \leq 0) \wedge$ $(x-1)^2 + y^2 - 0.09 > 0 \wedge$ $(x+1)^2 + y^2 - 1.1025 \geq 0 \vee$ $(x+1)^2 + y^2 - \frac{1}{25} \leq 0$	$(-3.8025 + x^2 + y^2 \leq 0 \wedge -y \geq 0 \vee$ $-0.9025 + (-1-x)^2 + y^2 \leq 0) \wedge$ $-0.09 + (-1-x)^2 + y^2 > 0 \wedge$ $-1.1025 + (1-x)^2 + y^2 \geq 0 \vee$ $-\frac{1}{25} + (1-x)^2 + y^2 \leq 0$	$\frac{x^7}{27} + x^6(-\frac{y}{5} - \frac{1}{96}) + x^5(\frac{2y^2}{9} - \frac{y}{32} - \frac{1}{2}) +$ $x^4(-\frac{2y^3}{9} + \frac{y}{3} + \frac{1}{31}) + x^3(\frac{y^4}{11} + \frac{y^3}{10} - \frac{10y^2}{13} + \frac{y}{18} + \frac{15}{16}) +$ $x^2(-\frac{y^5}{25} - \frac{y^4}{18} - \frac{y^3}{3} + \frac{y^2}{10} - \frac{1}{32}) +$ $x\left(\frac{y^6}{71} + \frac{2y^4}{11} - \frac{y^3}{25} - y^2 - \frac{y}{45} - \frac{3}{8}\right) +$ $\frac{y^6}{48} - \frac{y^5}{7} + \frac{y^4}{6} - \frac{y^3}{2} - \frac{y^2}{6} - \frac{y}{59} + \frac{1}{85} < 0$	48.82
	6	IJCAR16-1 [14]	$-x_1^2 + 4x_1 + x_2 - 4 \geq 0 \wedge$ $-x_1 - x_2 + 3 - y^2 > 0$	$-3x_1^2 - x_2^2 + 1 \geq 0 \wedge x_2 - z^2 \geq 0$	$1 - \frac{3x_1}{4} - \frac{x_2}{2} < 0$	0.16
	7	CAV13-1 [11]	$1 - a^2 - b^2 > 0 \wedge a^2 + b - 1 - x = 0 \wedge$ $b + bx + 1 - y = 0$	$x^2 - 2y^2 - 4 > 0$	$-1 + \frac{x^2}{2} - \frac{y}{3} + \frac{xy}{3} - \frac{y^2}{4} < 0$	3.25
	8	CAV13-2 [11]	$x^2 + y^2 + z^2 - 2 \geq 0 \wedge$ $1.2x^2 + y^2 + xz = 0$	$20 - 3x^2 - 4y^3 - 10z^2 \geq 0 \wedge$ $x^2 + y^2 - z - 1 = 0$	$105x^4 + x^2(140y^2 + 24y(5z+7) + 35z(3z+8)) +$ $2(70y^3 z + 5y^2(12z^2 + 21z + 28) - 14y(6z^3 + 5z^2 +$ $10) - 35(3z^4 + 8z^2 + 4z - 9)) < 14x(20x^2(z+1) +$ $10y^2(z+2) - 3y(4z^2 - 5z + 4) - 20z(z^2 + 2))$	3857.89
	9	CAV13-3 [11]	$vc < 49.61 \wedge fa = 0.5418vc^2 \wedge$ $fr = 1000 - fa \wedge ac = 0.0005fr \wedge$ $vc_1 = vc + ac$	$vc_1 \geq 49.61$	$-1 + \frac{2vc_1}{99} < 0$	40.63
with rounding	10	Parallel parabola	$y - x^2 - 1 \geq 0$	$y - x^2 < 0$	$\frac{1}{2} + x^2 < y$	4.50
	11	Parallel halfplane	$y - x - 1 \geq 0$	$y - x + 1 < 0$	$x < y$	2.46
	12	Sharper-1 [33]	$y + 1 < 0$	$x^2 + y^2 - 1 \leq 0$	$2 + y < y^2$	2.19
	13	Sharper-2 [33]	$y - x > 0 \wedge x + y > 0$	$y + x^2 <= 0$	$y > 0$	2.38
	14	Coincident	$x + y > 0 \vee x + y < 0$	$x + y = 0$	$(x+y)^2 > 0$	0.18
	15	Adjacent	$y - x^2 > 0$	$y - x^2 <= 0$	$x^2 < y$	0.25
	16	IJCAR16-2 [14]	$-y_1 + x_1 - 2 \geq 0 \wedge 2x_2 - x_1 - 1 > 0 \wedge$ $-y_1^2 - x_1^2 + 2x_1 y_1 - 2y_1 + 2x_1 \geq 0 \wedge$ $-y_2^2 - y_1^2 - x_2^2 - 4y_1 + 2x_2 - 4 \geq 0$	$-z_1 + 2x_2 + 1 \geq 0 \wedge 2x_1 - x_2 - 1 > 0 \wedge$ $-z_2^2 - 4x_2^2 + 4x_2 z_1 + 3z_1 - 6x_2 - 2 \geq 0 \wedge$ $-z_2^2 - x_1^2 - x_2^2 + 2x_1 + z_1 - 2x_2 - 1 \geq 0$	$x_1 < x_2$	12.33
	17	CAV13-4 [11]	$xa_1 + 2ya_1 \geq 0 \wedge xa_1 + 2ya_1 - x_1 = 0 \wedge$ $-2xa_1 + ya_1 - y_1 = 0 \wedge x - x_1 - 1 = 0 \wedge$ $y = y_1 + x \wedge xa = x - 2y \wedge ya = 2x + y$	$xa + 2ya < 0$	$2xa + 4ya > 5$	3.10
beyond polynomials	18	TACAS16 [15]	$y - x^2 \geq 0$	$y + \cos x - 0.8 \leq 0$	$15x^2 < 4 + 20y$	12.71
	19	Transcendental	$\sin x \geq 0.6$	$\sin x \leq 0.4$	SVM failed	–
unbalanced	20	Unbalanced	$x > 0 \vee x < 0$	$x = 0$	$x^2 > 0$	0.11



(a) Adjacent **(b)** Ultimate **(c)** Twisted

(d) Parallel parabola **(e)** Coincident **(f)** Sharper-2

(g) Face **(h)** CAV13-2 **(i)** TACAS16

Fig. 4: Visualization in NIL on a selected set of examples. Legends: gray region: $[\![\phi]\!]$, blue region: $[\![\psi]\!]$, pink region: $[\![I]\!]$ with a valid interpolant I, red dots: X^+, blue dots: X^-, circled dots: support vectors. Sample points are hidden in 3D-graphics for a clear presentation.

Applicability and comparison with existing approaches. As shown in Table 1, our learning-based technique succeeds in all of the benchmark examples that admit polynomial interpolants. Due to theoretical limitations of existing approaches as elaborated in Sect. 1, none of the aforementioned methods can cope with as many cases in Table 1 as NIL can. For instances, the Twisted example as depicted in Fig. 4(c) falls beyond of the scope of concave quadratic formulas and thus cannot be addressed by the approach in [14], while the Parallel parabola example as shown in Fig. 4(d) needs an infinite combination of linear constraints as an interpolant when performing the technique in [15] and hence not of practical use, to name just a few. Moreover, we list in Table 2 a comparison of the synthesized interpolants against works where the benchmark examples are collected from. As being immediately obvious from Table 2, our technique often

Table 2: Comparison of the synthesized interpolants.

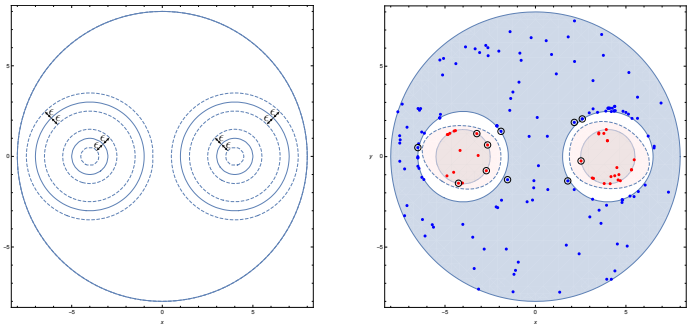
Name	Interpolants by NIL	Interpolants from the sources
IJCAR16-1 [14]	$1 - \frac{3x_1}{4} - \frac{x_2}{2} < 0$	$-3 + 2x_1 + x_1^2 + \frac{1}{2}x_2^2 > 0$
CAV13-1 [11]	$-1 + \frac{x^2}{2} - \frac{y}{3} + \frac{xy}{3} - \frac{y^2}{4} < 0$	$436.45(x^2 - 2y^2 - 4) + \frac{1}{2} \leq 0$
CAV13-2 [11]	$105x^4 + x^2(140y^2 + 24y(5z + 7) + 35z(3z + 8)) + 2(70y^3z + 5y^2(12z^2 + 21z + 28) - 14y(6z^3 + 5z^2 + 10) - 35(3z^4 + 8z^2 + 4z - 9)) < 14x(20x^2(z + 1) + 10y^2(z + 2) - 3y(4z^2 - 5z + 4) - 20z(z^2 + 2))$	$-14629.26 + 2983.44x_3 + 10972.97x_3^2 + 297.62x_2 + 297.64x_2x_3 + 0.02x_2x_3^2 + 9625.61x_2^2 - 1161.80x_2^2x_3 + 0.01x_2^2x_3^2 + 811.93x_2^3 + 2745.14x_2^4 - 10648.11x_1 + 3101.42x_1x_3 + 8646.17x_1x_3^2 + 511.84x_1x_2 - 1034x_1x_2x_3 + 0.02x_1x_2x_3^2 + 9233.66x_1x_2^2 + 1342.55x_1x_2^2x_3 - 138.70x_1x_2^3 + 11476.61x_1^2 - 3737.70x_1^2x_3 + 4071.65x_1^2x_3^2 - 2153.00x1_2x_2 + 373.14x_1^2x_2x_3 + 7616.18x_1^2x_2^2 + 8950.77x_1^3 + 1937.92x_1^3x_3 - 64.07x_1^3x_2 + 4827.25x_1^4 > 0$
CAV13-3 [11]	$-1 + \frac{2vc_1}{99} < 0$	$-1.3983vc_1 + 69.358 > 0$
Sharper-1 [33]	$2 + y < y^2$	$34y^2 - 68y - 102 \geq 0$
Sharper-2 [33]	$y > 0$	$8y + 4x^2 > 0$
IJCAR16-2 [14]	$x_1 < x_2$	$-x_1 + x_2 > 0$
CAV13-4 [11]	$2xa + 4ya > 5$	$716.77 + 1326.74(ya) + 1.33(ya)^2 + 433.90(ya)^3 + 668.16(xa) - 155.86(xa)(ya) + 317.29(xa)(ya)^2 + 222.00(xa)^2 + 592.39(xa)^2(ya) + 271.11(xa)^3$
TACAS16 [15]	$15x^2 < 4 + 20y$	$y > 1.8 \vee (0.59 \leq y \leq 1.8 \wedge -1.35 \leq x \leq 1.35) \vee (0.09 \leq y < 0.59 \wedge -0.77 \leq x \leq 0.77) \vee (y \geq 0 \wedge -0.3 \leq x \leq 0.3)$

produces interpolants of simpler forms, particularly for examples CAV13-2, CAV13-4 and TACAS16. Such a simplicity benefits from both the rounding effect and the form of interpolant (i.e., a single polynomial inequality) that we tend to construct.

Bottleneck of efficiency and potential solutions. The current implementation of NIL works promisingly for small examples; it does not scale to interpolation problems with large numbers of common variables, as reported in Table 1. The bottleneck stems from quantifier eliminations performed within every iteration of the learning process, for entailment checking and generating counterexamples. We pose here several potential solutions that are expected to significantly reduce computational efforts: (i) substitute general purpose QE procedure that perform CAD by the so-called variant quantifier-elimination (VQE) algorithm [21], which features singly-exponential complexity in the number of variables. This however requires a careful inspection of whether our problem meets the geometric conditions imposed by VQE; (ii) incorporate relaxation schemes, e.g., Lagrangian relaxation and sum-of-squares decompositions [34], and complement with QE only when the relaxation fails to produce desired results.

6 Taming Perturbations in Parameters.

An interpolant synthesized by the SVM-based technique features inherent robustness due to the way optimal-margin classifier is defined (Sect. 2). That is, the validity of such an interpolant is not easily perturbed by changes (in the feature space) in the formulae



(a) ϵ-perturbations in the radii **(b)** Interpolant resilient to perturbations

Fig. 5: ϵ-Face: introducing perturbations (with ϵ up to 0.5) in the Face example. The synthesized interpolant is resilient to any ϵ-perturbation in the radii satisfying $-0.5 \leq \epsilon \leq 0.5$.

to be interpolated. It is straightforward in NIL to deal with interpolation problems under explicitly specified perturbations, which are treated as constraints over fresh variables. An example named ϵ-Face is depicted in Fig. 5, which perturbs $\langle \phi, \psi \rangle$ in the Face example as $\phi \mathrel{\widehat{=}} -0.5 \leq \epsilon_1 \leq 0.5 \wedge ((x+4)^2 + y^2 - (1+\epsilon_1)^2 \leq 0 \vee (x-4)^2 + y^2 - (1+\epsilon_1)^2 \leq 0)$ and $\psi \mathrel{\widehat{=}} -0.5 \leq \epsilon_2 \leq 0.5 \wedge x^2+y^2-64 \leq 0 \wedge (x+4)^2+y^2-(3+\epsilon_2)^2 \geq 0 \wedge (x-4)^2 + y^2 - (3+\epsilon_2)^2 \geq 0$. The synthesized interpolant over common variables of ϕ and ψ is $\frac{x^4}{139} + \frac{x^3 y}{268} + x^2 \left(\frac{y^2}{39} - \frac{11}{36}\right) + x \left(-\frac{y^3}{52} - \frac{y^2}{157} - \frac{y}{52} - \frac{1}{116}\right) + \frac{y^4}{25} - \frac{y^3}{182} + \frac{2y^2}{19} - \frac{y}{218} + 1 < 0$ which is hence resilient to any ϵ-perturbation in the radii satisfying $-0.5 \leq \epsilon \leq 0.5$, as illustrated in Fig. 5(b).

7 Conclusions

We have presented a unified, counterexample-guided method named NIL for generating polynomial interpolants over the general quantifier-free theory of nonlinear arithmetic. Our method is based on classification techniques with space transformations and kernel tricks as established in the community of machine-learning. We proved the soundness of NIL and proposed sufficient conditions for its completeness and convergence. The applicability and effectiveness of our technique are demonstrated experimentally on a collection of representative benchmarks from the literature, including those extracted from program verification. Experimental results indicated that our method suffices to address more interpolation tasks, including those with perturbations in parameters, and in many cases synthesizes simpler interpolants compared with existing approaches.

For future work, we would like to improve the efficiency of NIL by substituting the general purpose quantifier-elimination procedure with alternative methods previously discussed in Sect. 5. An extension of our approach to cater for the combination of nonlinear arithmetic with EUFs, by resorting to predicate-abstraction techniques [23], will be of particular interest. Additionally, we plan to investigate the performance of NIL over different classification techniques, e.g., the widespread regression-based methods [19], though SVMs are expected to be more competent concerning the robustness and predictability, as also observed in [37].

References

1. K. P. Bennett and E. J. Bredensteiner. Duality and geometry in SVM classifiers. In *ICML'00*, pages 57–64, 2000.
2. C. M. Bishop. *Pattern recognition and machine learning*, pages 326–328. Springer, 2006.
3. B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT'92*, pages 144–152, 1992.
4. N. Bourbaki. *Topological Vector Spaces*. Elements of Mathematics. Springer-Verlag, 1987.
5. C. Chang and C. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27, 2011.
6. M. Chen, J. Wang, J. An, B. Zhan, D. Kapur, and N. Zhan. NIL: Learning nonlinear interpolants (full version). [Online]. Available: `http://lcs.ios.ac.cn/~chenms/papers/CADE-27_FULL.pdf`.
7. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient interpolant generation in satisfiability modulo theories. In *TACAS'08*, pages 397–412, 2008.
8. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, pages 154–169, 2000.
9. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In H. Brakhage, editor, *Automata Theory and Formal Languages 2nd GI Conference Kaiserslautern*, pages 134–183, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.
10. W. Craig. Linear reasoning. A new form of the herbrand-gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.
11. L. Dai, B. Xia, and N. Zhan. Generating non-linear interpolants by semidefinite programming. In *CAV'13*, pages 364–380, 2013.
12. S. Dathathri, N. Arechiga, S. Gao, and R. M. Murray. Learning-based abstractions for nonlinear constraint solving. In *IJCAI'17*, pages 592–599, 2017.
13. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI'10*, pages 129–145, 2010.
14. T. Gan, L. Dai, B. Xia, N. Zhan, D. Kapur, and M. Chen. Interpolant synthesis for quadratic polynomial inequalities and combination with EUF. In *IJCAR'16*, pages 195–212, 2016.
15. S. Gao and D. Zufferey. Interpolants in nonlinear theories over the reals. In *TACAS'16*, pages 625–641, 2016.
16. S. Gilbert. A nullstellensatz and a positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, 1974.
17. S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *CAV'97*, pages 72–83, 1997.
18. B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. Automatically refining abstract interpretations. In *TACAS'08*, pages 443–458, 2008.
19. T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction, 2nd Edition*. Springer series in statistics. Springer, 2009.
20. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL'04*, pages 232–244, 2004.
21. H. Hong and M. S. E. Din. Variant quantifier elimination. *J. Symb. Comput.*, 47(7):883–901, 2012.
22. S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari. Oracle-guided component-based program synthesis. In *ICSE'10*, pages 215–224, 2010.
23. R. Jhala, A. Podelski, and A. Rybalchenko. Predicate abstraction for program verification. In *Handbook of Model Checking.*, pages 447–491. 2018.
24. Y. Jung, W. Lee, B. Wang, and K. Yi. Predicate generation for learning-based quantifier-free loop invariant inference. In *TACAS'11*, pages 205–219, 2011.

25. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *FSE'06*, pages 105–116, 2006.
26. L. Kovács and A. Voronkov. Interpolation and symbol elimination. In *CADE'09*, pages 199–213, 2009.
27. J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *J. Symb. Log.*, 62(2):457–486, 1997.
28. S. Kupferschmid and B. Becker. Craig interpolation in the presence of non-linear constraints. In *FORMATS'11*, pages 240–255, 2011.
29. S. Lang. *Introduction to Diophantine Approximations: New Expanded Edition*. Springer New York, 2012.
30. K. L. McMillan. Interpolation and sat-based model checking. In *CAV'03*, pages 1–13, 2003.
31. K. L. McMillan. An interpolating theorem prover. In *TACAS'04*, pages 16–30, 2004.
32. K. L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *TACAS'08*, pages 413–427, 2008.
33. T. Okudono, Y. Nishida, K. Kojima, K. Suenaga, K. Kido, and I. Hasuo. Sharper and simpler nonlinear interpolants for program verification. In *APLAS'17*, pages 491–513, 2017.
34. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, 96(2):293–320, 2003.
35. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.
36. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. In *VMCAI'07*, pages 346–362, 2007.
37. R. Sharma, A. V. Nori, and A. Aiken. Interpolants as classifiers. In *CAV'12*, pages 71–87, 2012.
38. V. Sofronie-Stokkermans. Interpolation in local theory extensions. In *IJCAR'06*, pages 235–250, 2006.
39. V. Sofronie-Stokkermans. On interpolation and symbol elimination in theory extensions. In *IJCAR'16*, pages 273–289, 2016.
40. A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In *PLDI'05*, pages 281–294, 2005.
41. A. W. Strzeboński. Real root isolation for exp-log functions. In *ISSAC'08*, pages 303–314, 2008.
42. A. W. Strzeboński. Real root isolation for tame elementary functions. In *ISSAC'09*, pages 341–350, 2009.
43. A. W. Strzeboński. Cylindrical decomposition for systems transcendental in the first variable. *J. Symb. Comput.*, 46(11):1284–1290, 2011.
44. A. Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, 1951.
45. V. Vladimir. Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780, 1963.
46. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In *CADE'05*, pages 353–368, 2005.
47. J. Zhang and Y. Feng. Obtaining exact value by approximate computations. *Science in China Series A: Mathematics*, 50(9):1361, 2007.