Probably Approximately Correct Interpolants Generation

Bai Xue[1,2](✉) and Naijun Zhan[1,2]

[1] State Key Laboratory of Computer Science, Institute of Software, CAS, Beijing, China
{xuebai,znj}@ios.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China

Abstract. In this paper we propose a linear programming based method to generate interpolants for two Boolean formulas in the framework of probably approximately correct (PAC) learning. The computed interpolant is termed as a PAC interpolant with respect to a violation level $\epsilon \in (0,1)$ and confidence level $\beta \in (0,1)$: with at least $1-\beta$ confidence, the probability that the PAC interpolant is a true interpolant is larger than $1-\epsilon$. Unlike classical interpolants which are used to justify that two formulas are inconsistent, the PAC interpolant is proposed for providing a formal characterization of how inconsistent two given formulas are. This characterization is very important, especially for situations that the two formulas cannot be proven to be inconsistent. The PAC interpolant is computed by solving a scenario optimization problem, which can be regarded as a statistically sound formal method in the sense that it provides formal correct guarantees expressed using violation probabilities and confidences. The scenario optimization problem is reduced to a linear program in our framework, which is constructed by a family of independent and identically distributed samples of variables in the given two Boolean formulas. In this way we can synthesize interpolants for formulas that existing methods are not capable of dealing with. Three examples demonstrate the merits of our approach.

1 Introduction

Given two Boolean formulas ϕ and ψ, a classical Craig interpolant is a Boolean formula h in terms of only common symbols and variables of ϕ and ψ that over-approximates ϕ and remains inconsistent with ψ [7], implying that the two formulas ϕ and ψ are inconsistent. Interpolation-based techniques have been drawing both practical and theoretic attention in recent years. From a practical perspective, interpolant has been broadly applied to a variety of research areas, especially for formal verification, e.g., theorem proving [21,26], model-checking

This work has been supported through grants by NSFC under grant No. 61836005, 61625206, the CAS Pioneer Hundred Talents Program under grant No. Y8YC235015, and the MoE, Singapore, Tier-2 grant #MOE2019-T2-2-040.

J. Pang and L. Zhang (Eds.): SETTA 2020, LNCS 12153, pp. 143–159, 2020.
https://doi.org/10.1007/978-3-030-62822-2_9

[23] and predicate abstraction [16,18,24]. Because of their inherently modular and local reasoning interpolant-based techniques can substantially increase the scalability of formal techniques.

In existing literature there are various efficient algorithms for automatically synthesizing interpolants for various theories, e.g., decidable fragments of first-order logic, linear arithmetic, array logic, equality logic with uninterpreted functions (EUF), etc., and their combinations [6,18–20,24,25,27,28,35]. In contrast, interpolant generation for non-linear theory and its combination with the aforementioned theories is still in infancy, although nonlinear inequalities are ubiquitous in software involving sophisticated number theoretic functions as well as hybrid systems [36,37]. Many existing methods to synthesizing interpolants for non-linear theory are applicable to polynomial formulas, e.g., [8,11,12]. These methods encode the interpolant generation problem for two Boolean formulas as a semi-definite programming problem, which falls within the convex optimization framework and can be solved efficiently via interior-point methods in polynomial time. Although they are "polynomially" solvable, semidefinite programs with dimension above 10,000 have been extremely hard to solve in practice [1]. Recently, a method built on top of the SMT solver iSAT [10] was proposed to generate interpolants in the presence of non-linear constraints in [22]. [28] regarded interpolants as classifiers in supervised machine learning and used support vector machines (SVMs) in classification techniques and counterexample-guided techniques for linear interpolations generation. [5] extended the idea in [28] by using kernel trick to non-linear interpolant generation and by exploiting symbolic computation to guarantee the convergence, soundness and completeness of the approach. This method is promising since it can deal with superficially non-linear formulas and SVMs are routinely used in large scale data processing. However, the search for counterexamples is nontrivial generally, particularly, it cannot guarantee the convergence in case the two considered formulas are unbounded. [13,14] presented an approach to extract interpolants for non-linear formulas, which possibly contain transcendental functions and differential equations, from proofs of unsatisfiability generated by δ-decision procedure based on interval constraint propagation (ICP) [2]. Unfortunately, all of these approaches for nonlinear formulas suffer from so-called "curse of dimensionality" and thus cannot be applied to formulas with high-dimensional variables. Besides, if an interpolant is not found, these methods cannot give a characterization of how inconsistent the two formulas of interest are.

In this paper we attempt to motivate the integration of formal methods with machine learning [30], and propose a linear programming based method to generate reliable interpolants in the sense of featuring a rigorously quantified confidence for two Boolean formulas. Our method falls within the framework of PAC learning [17,33] and the generated interpolant provides a formal characterization of how inconsistent the two Boolean formulas of interest are. Given a violation level $\epsilon \in (0,1)$ and a confidence level $\beta \in (0,1)$, the objective is to compute PAC interpolants with respect to ϵ and β: With at least $1-\beta$ confidence, the probability that a PAC interpolant is a true interpolant is larger than $1-\epsilon$. Such interpolant in our method is computed based on scenario optimization [3], which encodes as a linear programming problem. In this scenario optimization framework, we first extract a set of independent and identically distributed samples of variables in the two Boolean formulas. A sufficient lower bound on the number of extracted samples can be computed from the specified violation level ϵ and confidence level β. After generating samples, we then construct a linear program for computing a PAC interpolant with respect to ϵ and β. Several examples demonstrate the performance of our approach.

In this paper we make use of capabilities of machine learning to assess large data sets to enable interpolant synthesis for large-scale formulas. The advantages of our method are summarized below.

1. The PAC interpolant is able to provide a characterization of how inconsistent two formulas are. This characterization is very important, especially when classical interpolants cannot be obtained to decide whether the two formulas are inconsistent.
2. The scale of the constructed linear program does not directly depend on the dimension of the variables in the considered formulas. It only depends on ϵ, β and the number of unknown parameters in a pre-specified interpolant template. Moreover, linear programming problems with hundreds of thousands or even millions of variables are routinely solved [15] in polynomial time via interior-point methods. Thus, it can deal with formulas with variables of arbitrarily high-dimension.

This remainder of this paper is structured as follows. In Sect. 2 we formalize the concept of PAC interpolants. Section 3 elucidates our PAC interpolant generation method that is based on scenario optimization. After evaluating our method on several examples in Sect. 4, we conclude this paper in Sect. 5.

2 Preliminaries

In this section, we first give a brief introduction on some notions used throughout this paper and then describe the PAC interpolant generation problem of interest.

2.1 Interpolants

Craig showed that given two formulas ϕ and ψ in a first-order theory $\mathcal{T}$ such that $\phi \models \psi$, there always exists an *interpolant* I over the common symbols and variables of ϕ and ψ such that $\phi \models I, I \models \psi$. In the verification literature, this terminology has been abused following [24], where a reverse interpolant I over the common variables of ϕ and ψ is defined by

Definition 1 (Interpolant). *Given two formulas ϕ and ψ in a theory $\mathcal{T}$, a formula I is an* interpolant *of ϕ and ψ if (1) $\phi \models I$, (2) $I \wedge \psi \models \bot$ and (3) I only contains those symbols and non-logical variables that are common to both ϕ and ψ.*

From Definition 1, we conclude that if there exists an interpolant I for formulas ϕ and ψ, the two formulas ϕ and ψ are inconsistent, i.e., $\phi \wedge \psi \models \bot$. This is especially useful in the safety verification scenario [31]. For instance, if $\{(x,y) \in \mathbb{R}^2 \mid \phi(x,y)\}$ denotes the set of unsafe states and $\{(x,y) \in \mathbb{R}^2 \mid \psi(x,y)\}$ denotes the set of reachable states of a system of interest, we can conclude that the system is safe if we can find an interpolant I for the two formulas ϕ and ψ.

The interpolant synthesis problem is described in Definition 2.

Definition 2. *Let $\phi(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$ be two formulas defined as follows,*

$$\begin{aligned}
&\phi(\boldsymbol{x}) : \vee_{i=1}^{l} \phi_i(\boldsymbol{x}) \textit{ with}\\
&\phi_1(\boldsymbol{x}) : f_{1,1}(\boldsymbol{x}) \rhd 0 \wedge \cdots \wedge f_{1,m_1}(\boldsymbol{x}) \rhd 0,\\
&\cdots\\
&\phi_l(\boldsymbol{x}) : f_{l,1}(\boldsymbol{x}) \rhd 0 \wedge \cdots \wedge f_{l,m_l}(\boldsymbol{x}) \rhd 0
\end{aligned}$$

and

$$\begin{aligned}
&\psi(\boldsymbol{x}) : \vee_{i=1}^{k} \psi_i(\boldsymbol{x}) \textit{ with}\\
&\psi_1(\boldsymbol{x}) : g_{1,1}(\boldsymbol{x}) \rhd 0 \wedge \cdots \wedge g_{1,n_1}(\boldsymbol{x}) \rhd 0,\\
&\cdots\\
&\psi_k(\boldsymbol{x}) : g_{k,1}(\boldsymbol{x}) \rhd 0 \wedge \cdots \wedge g_{k,n_k}(\boldsymbol{x}) \rhd 0,
\end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^r$ are variable vectors, $r \in \mathbb{N}$, $\rhd \in \{<, \leq, >, \geq\}$, $f_{i,j}s$ are nonlinear functions from $\mathbb{R}^r$ to $\mathbb{R}$, $i = 1, \ldots, l$, $j = 1, \ldots, m_i$, and $g_{i,j}s$ are nonlinear functions from $\mathbb{R}^r$ to $\mathbb{R}$, $i = 1, \ldots, k$, $j = 1, \ldots, n_i$. Suppose both $\{\boldsymbol{x} \mid \phi(\boldsymbol{x})\}$ and $\{\boldsymbol{x} \mid \psi(\boldsymbol{x})\}$ are bounded subsets in $\mathbb{R}^r$. Find a real-valued function $h(\boldsymbol{x}) : \mathbb{R}^r \to \mathbb{R}$ such that $h(\boldsymbol{x}) > 0$ is an interpolant for ϕ and ψ, i.e.

$$\begin{aligned}
&h(\boldsymbol{x}) > 0, \forall \boldsymbol{x} \in \mathcal{F}_1\\
&h(\boldsymbol{x}) \leq 0, \forall \boldsymbol{x} \in \mathcal{F}_2,
\end{aligned} \tag{1}$$

where $\mathcal{F}_1 = \{\boldsymbol{x} \mid \phi(\boldsymbol{x})\}$ and $\mathcal{F}_2 = \{\boldsymbol{x} \mid \psi(\boldsymbol{x})\}$.

Direct computations of interpolants for nonlinear formulas ϕ and ψ are non-trivial. In this paper we attempt to use finite randomization to learn interpolants. The learned interpolants are called PAC interpolants, whose concept is presented in Subsect. 2.2.

2.2 PAC Interpolants

This subsection introduces PAC interpolants.

Suppose that A_i and B_j are respectively endowed with a σ-algebra $\mathcal{D}_{A_i}$ and $\mathcal{D}_{B_j}$, and that probabilities $\texttt{Pr}_{A_i}$ and $\texttt{Pr}_{B_j}$ are respectively assigned over $\mathcal{D}_{A_i}$ and $\mathcal{D}_{B_j}$, where

$$A_i = \{\boldsymbol{x} \mid \phi_i(\boldsymbol{x})\}$$

and

$$B_j = \{\boldsymbol{x} \mid \psi_j(\boldsymbol{x})\},$$

$i = 1, \ldots, l$, $j = 1, \ldots, k$. Throughout this paper, we assume uniform distributions over both spaces A_i and B_j.

Suppose now that

$$\Delta = A_1 \times \ldots \times A_l \times B_1 \times \ldots \times B_k \tag{2}$$

is endowed with a $\sigma-$algebra $\mathcal{D}$. Obviously, $\cup_{i=1}^{l} A_i = \mathcal{F}_1$ and $\cup_{j=1}^{k} B_j = \mathcal{F}_2$. According to the product measure theorem, there exists a probability measure $\mathtt{Pr}$ over $\mathcal{D}$ such that

$$\mathtt{Pr} = \mathtt{Pr}_{A_1} \times \cdots \times \mathtt{Pr}_{A_l} \times \mathtt{Pr}_{B_1} \times \cdots \mathtt{Pr}_{B_k}. \tag{3}$$

Clearly, a uniform distribution is assigned on the space Δ.

Definition 3. *A real-valued function $h(\boldsymbol{x}) : \mathbb{R}^r \to \mathbb{R}$ is a* PAC *interpolant with respect to $\epsilon \in (0,1)$ and $\beta \in (0,1)$ for formulas ϕ and ψ, where ϕ and ψ are formulas in Definition 2, if with probability no smaller than $1-\beta$,*

$$\mathtt{Pr}(\{\boldsymbol{w} \in \Delta \mid h(\boldsymbol{x}_{A_i}) > 0, h(\boldsymbol{x}_{B_j}) \le 0, i = 1, \ldots, l, j = 1, \ldots, k\}) \ge 1 - \epsilon, \tag{4}$$

where

$$\boldsymbol{w} = (\boldsymbol{x}_{A_1}, \ldots, \boldsymbol{x}_{A_l}, \boldsymbol{x}_{B_1}, \ldots, \boldsymbol{x}_{B_k}) \in \Delta,$$

$\boldsymbol{x}_{A_i} \in A_i$ and $\boldsymbol{x}_{B_j} \in B_j$, $i = 1, \ldots, l$, $j = 1, \ldots, k$. We then say that the function $h(\boldsymbol{x})$ is $\mathtt{CI}(\epsilon, \beta)$.

The probability β in Definition 3, which is related to $\mathtt{Pr}$, refers to the confidence associated to the randomized solution algorithm. According to Definition 3, with at least $1-\beta$ confidence, the probability $\mathtt{Pr}$ that $\mathtt{CI}(\epsilon, \beta)$ satisfies the conditions (1) is $1-\epsilon$. That is, with at least $1-\beta$ confidence, the probability $\mathtt{Pr}$ that $\mathtt{CI}(\epsilon, \beta)$ indeed is a true interpolant for formulas ϕ and ψ is $1-\epsilon$.

Remark 1. It is worth pointing out here that $\mathtt{CI}(\epsilon, \beta)$ may not satisfy the conditions (1) even if $\epsilon = 0$ and $\beta = 0$. Therefore, $\mathtt{CI}(0,0)$ may not be the a true interpolant.

We further give an explanation of PAC interpolants below.

In some situations it is not sufficient to know whether the formulas ψ and ϕ are inconsistent or not. It is also important to know how inconsistent the two formulas are, by quantifying the amount of states such that the formulas are inconsistent. This characterization is not only useful when existing traditional methods cannot decide whether $\mathcal{F}_1$ intersects $\mathcal{F}_2$ or the two formulas ϕ and ψ are inconsistent, but also useful when the two formulas ϕ and ψ are not inconsistent. The PAC interpolants can help achieve such characterization.

Corollary 1. *If $h(\boldsymbol{x})$ is $\mathtt{CI}(\epsilon, \beta)$ for formulas ψ and ϕ, where ψ and ϕ are formulas in Definition 2, then with confidence no smaller than $1-\beta$,*

$$\begin{aligned} &\mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in A_i \mid h(\boldsymbol{x}) > 0\}) \ge 1 - \epsilon \textit{ and} \\ &\mathtt{Pr}_{B_j}(\{\boldsymbol{x} \in B_j \mid h(\boldsymbol{x}) \le 0\}) \ge 1 - \epsilon \end{aligned} \tag{5}$$

for $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, k\}$.

Proof. Let $\Delta_i = \{\boldsymbol{w} \in \Delta \mid h(\boldsymbol{x}_{A_i}) > 0\}$ and $\tilde{\Delta} = \{\boldsymbol{w} \in \Delta \mid h(\boldsymbol{x}_{A_i}) > 0, h(\boldsymbol{x}_{B_j}) \leq 0, i = 1, \dots, l, j = 1, \dots, k\}$, where $\boldsymbol{w}$ is as in Definition 3. Obviously, $\tilde{\Delta} \subseteq \Delta_i$. According to (3),

$$\mathtt{Pr}(\Delta_i) = \mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in A_i \mid h(\boldsymbol{x}) > 0\})$$

holds. Also, since $\tilde{\Delta} \subseteq \Delta_i$ for $i = 1, \dots, l$ and (4), we have that with confidence no smaller than $1 - \beta$,

$$\mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in A_i \mid h(\boldsymbol{x}) > 0\}) \geq 1 - \epsilon.$$

Similarly, we have that with confidence no smaller than $1 - \beta$,

$$\mathtt{Pr}_{B_j}(\{\boldsymbol{x} \in B_j \mid h(\boldsymbol{x}) \leq 0\}) \geq 1 - \epsilon.$$

□

Corollary 1 tells that if a PAC interpolant $\mathtt{CI}(\epsilon, \beta)$ is obtained for formulas ϕ and ψ, then with confidence at least $1 - \beta$, the probability measure of states in $C_{i,j} = A_i \cap B_j$ satisfies

$$\mathtt{Pr}_{A_i}(C_{i,j}) \leq \min\{1, \epsilon + \frac{\epsilon\lambda(B_j)}{\lambda(A_i)}\}, \tag{6}$$

where $\lambda(A_i)$ and $\lambda(B_j)$ respectively represent the Lebesgue measure of the sets A_i and B_j, $i = 1, \dots, l$, $j = 1, \dots, k$. The states in $C_{i,j}$ are the ones such that the two formulas ϕ_i and ψ_j are consistent.

(6) is obtained as follows: Since $C_{i,j} = \{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) > 0\} \cup \{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) \leq 0\}$. According to (5), we have that

$$\begin{aligned} &\mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) \leq 0\}) \\ &\leq \mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in A_i \mid h(\boldsymbol{x}) \leq 0\}) \leq \epsilon \end{aligned} \tag{7}$$

and

$$\begin{aligned} &\mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) > 0\}) \\ &= \mathtt{Pr}_{B_j}(\{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) > 0\}) \cdot P \\ &\leq \frac{\epsilon\lambda(B_j)}{\lambda(A_i)}, \end{aligned} \tag{8}$$

where $P = \frac{\mathtt{Pr}_{A_i}(\{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) > 0\})}{\mathtt{Pr}_{B_j}(\{\boldsymbol{x} \in C_{i,j} \mid h(\boldsymbol{x}) > 0\})}$. Combining (7) and (8), we have (6).

Consequently, the probability measure of the states $\boldsymbol{x}$ in $\{\boldsymbol{x} \mid \phi_i(\boldsymbol{x})\}$ such that $\phi_i(\boldsymbol{x}) \wedge \psi_j(\boldsymbol{x}) \models \perp$ is larger than or equal to

$$1 - \min\{1, \epsilon + \frac{\epsilon\lambda(B_j)}{\lambda(A_i)}\},$$

with confidence at least $1 - \beta$, $i = 1, \dots, l$, $j = 1, \dots, k$. Consequently, the probability measure of the states $\boldsymbol{x}$ in $\{\boldsymbol{x} \mid \phi_i(\boldsymbol{x})\}$ such that $\phi_i(\boldsymbol{x}) \wedge \psi(\boldsymbol{x}) \models \perp$ is larger than or equal to

$$\max\{0, 1 - \sum_{j=1}^{k}(\epsilon + \frac{\epsilon\lambda(B_j)}{\lambda(A_i)})\},$$

with confidence at least $1-\beta$, $i = 1, \dots, l$.

We also have the similar conclusion for ψ_j, $j = 1, \dots, k$. That is, the probability measure of states in $\{\boldsymbol{x} \mid \psi_j(\boldsymbol{x})\}$ such that $\psi_j(\boldsymbol{x}) \wedge \phi(\boldsymbol{x}) \models \perp$ is larger than or equal to

$$\max\{0, 1 - \sum_{i=1}^{l}(\epsilon + \frac{\epsilon\lambda(A_i)}{\lambda(B_j)})\},$$

with confidence at least $1-\beta$.

Let's take (6) as an instance to give a further explanation of PAC interpolants in the safety verification scenario [32]. Let A_i be a set of possibly reachable states of a system of interest and $\mathcal{F}_2$ be a set of unsafe states. Thus, the probability of reaching the unsafe region $\mathcal{F}_2$ is less than or equal to

$$\min\{1, \sum_{j=1}^{k}(\epsilon + \frac{\epsilon\lambda(B_j)}{\lambda(A_i)})\},$$

with at least $1-\beta$ confidence. If β is extremely small (smaller than 10^{-10}), then we have a priori practical certainty that the unsafe probability does not exceed

$$\min\{1, \sum_{j=1}^{k}(\epsilon + \frac{\epsilon\lambda(B_j)}{\lambda(A_i)})\}.$$

Further, if ϵ is below a threshold as well, it is reasonable to believe that the system is practically safe in real applications.

Especially, if a computed $\texttt{CI}(\epsilon, \beta)$ is verified to satisfy (1) based on existing methods such as SMT solving and semi-definite programming, the computed $\texttt{CI}(\epsilon, \beta)$ is a true interpolant and thus the formulas ϕ and ψ are inconsistent.

3 PAC Interpolants Generation

In this section we present our method for generating PAC interpolants. The method is based on the scenario optimization yielding a linear program to be solved.

3.1 Scenario Optimization

The scenario optimization is an intuitive and effective way to deal with robust optimization problems based on finite randomization of the constraints [3]. Concretely, consider the following robust optimization problem:

$$\begin{aligned} &\min_{\gamma \in \Gamma \subseteq \mathbb{R}^m} \boldsymbol{c}^T \boldsymbol{\gamma} \\ &\text{such that } \boldsymbol{f}_{\boldsymbol{\delta}}(\boldsymbol{\gamma}) \leq 0, \forall \boldsymbol{\delta} \in \tilde{\Delta}, \end{aligned} \tag{9}$$

where Γ is a convex and closed set, and $\boldsymbol{f}_{\boldsymbol{\delta}}(\boldsymbol{\gamma})$ are convex functions over the decision variable $\boldsymbol{\gamma}$ for every $\boldsymbol{\delta}$ in a closed set $\tilde{\Delta} \subseteq \mathbb{R}^r$.

Definition 4 (Scenario Optimization). *Extract N independent and identically distributed samples $(\boldsymbol{\delta}^{(i)})_{i=1}^N$ from $\tilde{\Delta}$ according to probability* $\mathtt{P}$ *and solve the convex program* (10)*:*

$$\begin{aligned} &\min_{\gamma \in \Gamma \subseteq \mathbb{R}^m} \boldsymbol{c}^T \boldsymbol{\gamma} \\ &s.\ t.\ \wedge_{i=1}^N \boldsymbol{f}_{\boldsymbol{\delta}^{(i)}}(\boldsymbol{\gamma}) \le 0. \end{aligned} \tag{10}$$

(10) *is a relaxation of* (9) *and the process of solving* (10) *to obtain an approximate solution to* (9) *is called* scenario optimization *of* (9)*. Correspondingly, its optimal solution γ^* is called* scenario solution *to* (10)*.*

Theorem 1 shows that the solution $\boldsymbol{\gamma}_N^*$ to (10) satisfies all constraints in (9) except a fraction.

Theorem 1. *[4] Choose a violation level $\epsilon \in (0, 1)$ and a confidence level $\beta \in (0, 1)$. If* (10) *is feasible and attains a unique optimal solution, and*

$$N \ge \frac{2}{\epsilon}(\ln \frac{1}{\beta} + m), \tag{11}$$

where m is the number of optimization variables $\boldsymbol{\lambda}$, then with confidence at least $1 - \beta$, $\boldsymbol{\gamma}_N^*$ satisfies all constraints in $\tilde{\Delta}$ but at most a fraction of probability measure ϵ, i.e., $\mathtt{P}(\{\boldsymbol{\delta} \mid \boldsymbol{f}_{\boldsymbol{\delta}}(\boldsymbol{\gamma}_N^*) > 0\}) \le \epsilon$.

In Theorem 1, $1 - \beta$ is the $N-$fold probability $\mathtt{P}^N$ in $\tilde{\Delta}^N = \tilde{\Delta} \times \cdots \times \tilde{\Delta}$, which is the set to which the extracted sample $(\boldsymbol{\delta}^{(1)}, \ldots, \boldsymbol{\delta}^{(N)})$ belongs. A unique optimal solution can be selected from the Tie-break rule if multiple optimal solutions occur for (10), Theorem 1 still holds if the uniqueness of optimal solutions to (10) in Theorem 1 is removed [3]. The minimum number of samples depends logarithmically on β^{-1}, we can choose a high confidence without increasing the required samples too much. Moreover, we observe from Theorem 1 that the number N of required samples does not depend on the dimension of the universally quantified variables $\boldsymbol{\delta}$. This facilitates application of the scenario optimization approach to systems with high-dimensional variables $\boldsymbol{\delta}$. Recently, scenario optimization was used to compute probably approximately safe inputs for a given black-box system such that the system's final outputs fall within a safe range in [33], perform safety verification of hybrid systems in [32] and black-box continuous time dynamical systems in [34].

3.2 PAC Interpolant Generation

In this subsection we elucidate our linear programming based approach for computing the PAC interpolants.

We first select an interpolant template $h(c_1, \dots, c_{l'}, \boldsymbol{x})$ such that $h(c_1, \dots, c_{l'}, \boldsymbol{x})$ is for every $\boldsymbol{x} \in \mathbb{R}^n$ a linear function in $c_1, \dots, c_{l'}$, where $(c_i)_{i=1,\dots,l'}$ are unknown parameters. For instance, for a two-dimensional state variable $\boldsymbol{x} = (x_1, x_2)^\top$, $w(c_1, c_2, \boldsymbol{x}) = c_1 x_1 + c_2 x_2^2$ is a linear function in c_1 and c_2, and $w(c_1, c_2, \boldsymbol{x}) = c_1 e^{x_1 x_2} + c_2 \ln(x_2)$ is also a linear function over c_1 and c_2. How to select a best interpolant template is not the focus of this paper. Generally, we would use an interpolant template of the polynomial form over $\boldsymbol{x}$. If a given template fails to generate a PAC interpolant, a polynomial template with higher degree would be recommended.

In the following we show how to use scenario optimization from Definition 4 to solve (1) and obtain an approximate solution $(c_j)_{j=1,\dots,l'}$ such that with confidence at least $1-\beta$, the probability that $h(c_1, \dots, c_{l'}, \boldsymbol{x})$ satisfies (1) is larger than or equal to $1-\epsilon$. That is, $h(c_1, \dots, c_{l'}, \boldsymbol{x})$ is $\texttt{CI}(\epsilon, \beta)$.

According to Definition 4, we extract N independent and identically distributed samples

$$\{(\boldsymbol{x}_{A_1,i}, \dots, \boldsymbol{x}_{A_l,i}, \boldsymbol{x}_{B_1,i}, \dots, \boldsymbol{x}_{B_k,i}) \in \Delta\}_{i=1}^{N}$$

from the product space Δ in (2) according to the probability distribution $\texttt{Pr}$, where $\boldsymbol{x}_{A_j,i} \in A_j = \{\boldsymbol{x} \mid \phi_j(\boldsymbol{x})\}$ for $j = 1, \dots, l$ and $\boldsymbol{x}_{B_j,i} \in B_j = \{\boldsymbol{x} \mid \psi_j(\boldsymbol{x})\}$ for $j = 1, \dots, k$, we obtain a linear program (12) over $(c_i)_{i=1,\dots,l'}$ and γ,

$$\begin{aligned}
&\min_{c_i, i=1,\dots,l', \gamma} -\gamma \\
&\text{such that for each } i = 1, \dots, N: \\
&-h(c_1, \dots, c_{l'}, \boldsymbol{x}_{A_1,i}) + \gamma + \varepsilon \le 0, \\
&\dots \\
&-h(c_1, \dots, c_{l'}, \boldsymbol{x}_{A_l,i}) + \gamma + \varepsilon \le 0 \\
&h(c_1, \dots, c_{l'}, \boldsymbol{x}_{B_1,i}) + \gamma \le 0, \\
&\dots \\
&h(c_1, \dots, c_{l'}, \boldsymbol{x}_{B_k,i}) + \gamma \le 0, \\
&-U_c \le c_j \le U_c, j = 1, \dots, l', \\
&0 \le \gamma \le U_\gamma,
\end{aligned} \tag{12}$$

where ε is a given positive value, positive values U_c and U_γ are respectively pre-specified upper bounds for $|c_j|$, $j = 1, \dots, l'$, and γ. (12) can determine the thickest slab separating the two family of samples $(\boldsymbol{x}_{A_j,i})_{j=1,\dots,l}^{i=1,\dots,N}$ and $(\boldsymbol{x}_{B_j,i})_{j=1,\dots,k}^{i=1,\dots,N}$, where 2γ represents the thickness of the slab. ε is to ensure the positivity of $h(c_1, \dots, c_{l'}, \boldsymbol{x})$ over $\boldsymbol{x}_{A_j,i}$, $j = 1, \dots, l$, $i = 1, \dots, N$.

If $N \ge \frac{2}{\epsilon}(\ln \frac{1}{\beta} + l' + 1)$ and (12) has feasible solutions, we conclude that the function $h(c_1^*, \dots, c_{l'}^*, \boldsymbol{x})$ is $\texttt{CI}(\epsilon, \beta)$, where $(c_i^*)_{i=1,\dots,l'}$ is an optimal solution to (12).

Theorem 2. *Suppose that* (12) *is feasible with* $N \ge \frac{2}{\epsilon}(\ln \frac{1}{\beta} + l' + 1)$ *and* $(c_i^*)_{i=1,\dots,l'}$ *is an optimal solution to* (12)*, then the function* $h(c_1^*, \dots, c_{l'}^*, \boldsymbol{x})$ *is* $\texttt{CI}(\epsilon, \beta)$*.*

Proof. We reformulate (12) equivalently as the following linear program over $c_1, \dots, c_{l'}$ and γ,

$$\begin{aligned}
&\min_{c_i, i=1,\dots,l',\gamma} \gamma \\
&\text{such that for each } i = 1, \dots, N: \\
&-h'_{A_1,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) + \gamma + \varepsilon \leq 0, \\
&\dots \\
&-h'_{A_l,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) + \gamma + \varepsilon \leq 0, \\
&h'_{B_1,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) + \gamma \leq 0, \\
&\dots \\
&h'_{B_k,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) + \gamma \leq 0, \\
&-U_c \leq c_j \leq U_c, j = 1, \dots, l', \\
&0 \leq \gamma \leq U_\gamma,
\end{aligned} \tag{13}$$

where $h'_{A_j,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) = h(c_1, \dots, c_{l'}, \boldsymbol{x}_{A_j,i})$ for $j = 1, \dots, l$, $h'_{B_j,i}(c_1, \dots, c_{l'}, \boldsymbol{w}_i) = h(c_1, \dots, c_{l'}, \boldsymbol{x}_{B_j,i})$ for $j = 1, \dots, k$, and

$$\boldsymbol{w}_i = (\boldsymbol{x}_{A_1,i}, \dots, \boldsymbol{x}_{A_l,i}, \boldsymbol{x}_{B_1,i}, \dots, \boldsymbol{x}_{B_k,i}).$$

The number of decision variables in (13) is $l'+1$, i.e., $c_1, \dots, c_{l'}$ and γ.

Optimal solutions to (12) are optimal ones to (13), and vice versa. If the optimal solution $(c_1^*, \dots, c_{l'}^*, \gamma^*)$ to (13) is unique, the function $h(c_1^*, \dots, c_{l'}^*, \boldsymbol{x})$ is $\texttt{CI}(\epsilon,\beta)$ according to Theorem 1 and Definition 3. If the linear optimization (13) has multiple optimal solutions then we can take one of optimal solution and set $c_i := c_i^*$ in (13) to obtain a new linear program optimizing over the only variable γ. Obviously, γ^* is still the optimal solution to the resulting new linear optimization problem and its solution is unique. Therefore, by Theorem 1, the function $h(c_1^*, \dots, c_{l'}^*, \boldsymbol{x})$ is $\texttt{CI}(\epsilon, \beta)$. □

Our approach for synthesizing PAC interpolants is formally summarized in Algorithm 1.

4 Experiments

In this section we evaluate Algorithm 1 on three examples. Parameters that determine the performance of Algorithm 1 are presented in Fig. 1. All computations were performed on an i7-7500U 2.70 GHz CPU with 32 GB RAM running Windows 10.

Example 1. To enhance the understanding of Algorithm 1, we consider a simple example as follows:

$$\begin{aligned}
&\phi(x, y) : (f_1(x, y) \geq 0 \wedge f_2(x, y) \geq 0), \\
&\psi(x, y) : (g_1(x, y) \geq 0 \wedge g_2(x, y) \geq 0),
\end{aligned}$$

Algorithm 1. PAC Interpolant Generation

```
Input: formulas $\phi$ and $\psi$; $\varepsilon$ in (12); upper bounds $U_c$ and $U_\gamma$ in (12); violation level
  $\epsilon \in (0,1)$ and confidence value $\beta \in (0,1)$.
Output: If a $\mathtt{CI}(\epsilon, \beta)$ is computed, return "YES" and a $\mathtt{CI}(\epsilon, \beta)$; Otherwise, return
  "UNKNOWN".

1. Select an interpolant template $h(c_1, \dots, c_{l'}, \boldsymbol{x})$;
2. Compute the number $N$ of samples with respect to $\epsilon$ and $\beta$ according to (11);
3. Extract independent and uniformly distributed $N$ samples from the product
   space $\Delta$ in (2) based on a rejection sampling algorithm [29];
4. if an optimal solution $(c_j^*)_{j=1,\dots,l'}$ is computed via solving (12) then
      Return "YES" and $\mathtt{CI}(\epsilon, \beta) = h(c_1^*, \dots, c_{l'}^*, \boldsymbol{x})$;
5. else
      Return "UNKNOWN";
6. end if
```

			Alg.1							
Benchmarks	d_ϕ	d_ψ	ϵ	β	N	m	ε	U	T
Ex.1	2	2	0.01	0.01	2322	7	0.1	10	350.76
Ex.2	2	2	0.001	0.001	27816	7	0.1	10	383.69
Ex.3	100	100	0.01	10^{-12}	25927	102	0.1	10	563.91

Fig. 1. *Parameters and performance of our method on the listed examples.* $\mathtt{d}_\phi$ *and* $\mathtt{d}_\psi$*: dimensions of variables in formulas* ϕ *and* ψ *respectively;* ϵ*: violation level;* β*: confidence level;* N*: the number of extracted samples;* m*: the number of unknown variables in* (12)*;* ε*:* ε *in* (12)*;* U*: the upper bound for both* U_c *and* U_γ *in* (12)*;* T*: the computation time for computing* $\mathtt{CI}(\epsilon, \beta)$*.*

where $f_1(x,y) = 4 - x^2 - y^2$, $f_2(x,y) = y - x^2$, $g_1(x,y) = 4 - x^2 - y^2$ and $g_2(x,y) = x^2 - y - 0.5$.

The interpolant template is assumed as $c_1 + c_2x + c_3y + c_4x^2 + c_5xy + c_6y^2$, where $c_1, \dots, c_6$ are the unknown parameters. Thus, the number of optimization variables in (12) is 7. Let $\epsilon = 0.01$ and $\beta = 0.01$. According to Theorem 1, we obtain that the number N of samples is at least 2322.

We take 2322 independent and identically distributed samples

$$\{(x_{A_1,k}, y_{A_1,k}, x_{B_1,k}, y_{B_1,k})\}_{k=1}^{2322}$$

from the product space $A_1 \times B_1$ according to the uniform distribution, where

$$(x_{A_1,k}, y_{A_1,k}) \in A_1 = \{(x,y) \mid f_1(x,y) \geq 0, f_2(x,y) \geq 0\}$$

and

$$(x_{B_1,k}, y_{B_1,k}) \in B_1 = \{(x,y) \mid g_1(x,y) \geq 0, g_2(x,y) \geq 0\}$$

with $k = 1, \ldots, N$. This results in the following linear program over $c_1, \ldots, c_6, \gamma$:

$$\begin{aligned} &\min_{c_1,\ldots,c_6,\gamma} \gamma \\ &\text{such that for each } k = 1, \ldots, 2322: \\ &-h(x_{A_1,k}, y_{A_1,k}) + \gamma + \varepsilon \le 0, \\ &h(x_{B_1,k}, y_{B_1,k}) + \gamma \le 0, \\ &-10 \le c_i \le 10, i = 1, \ldots, 6, \\ &0 \le \gamma \le 10, \end{aligned} \tag{14}$$

where $\varepsilon = 0.1$. These extracted samples are shown in Fig. 2.

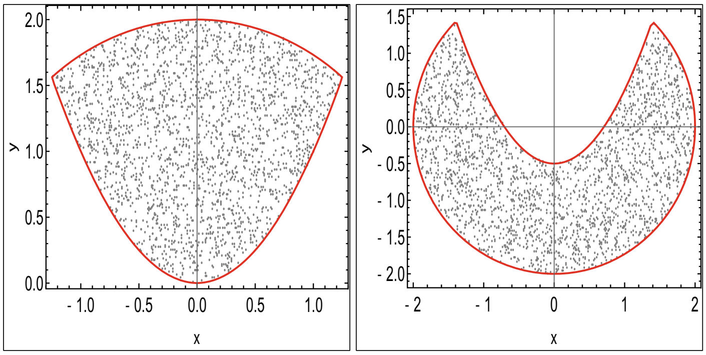

Fig. 2. An illustration of extracted samples. Red curves in the left and right figures are respectively the boundary of $\{(x, y) \mid \phi(x, y)\}$ and $\{(x, y) \mid \psi(x, y)\}$. Gray points in the left and right figures are respectively the extracted samples from $\{(x, y) \mid \phi(x, y)\}$ and $\{(x, y) \mid \psi(x, y)\}$.

Via solving (14), we obtain a $\mathtt{CI}(0.01, 0.01)$:

$$\begin{aligned} h(x, y) =& 2.51547516736 + 0.0165156178714x + 9.99999999665y \\ &- 9.95141962642x^2 + 0.0513084342584y^2 - 0.00560477791004xy. \end{aligned} \tag{15}$$

That is, with at least 99% confidence, the probability that $h(x, y) > 0$ is an interpolant for $\phi(x, y)$ and $\psi(x, y)$ is larger than or equal to 0.99. The plots of $\{(x, y) \mid \psi(x, y)\}$, $\{(x, y) \mid \phi(x, y)\}$ and $\{(x, y) \mid h(x, y) > 0\}$ are presented in Fig. 3.

Actually, a check using the REDUCE Computer Algebra System [9] proves that $h(x, y)$ in (15) is a true (not just probably approximately correct) interpolant satisfying

$$\begin{aligned} &h(x.y) > 0, \forall (x, y) \in \{(x, y) \mid \phi(x, y)\}, \\ &h(x, y) \le 0, \forall (x, y) \in \{(x, y) \mid \psi(x, y)\}. \end{aligned} \tag{16}$$

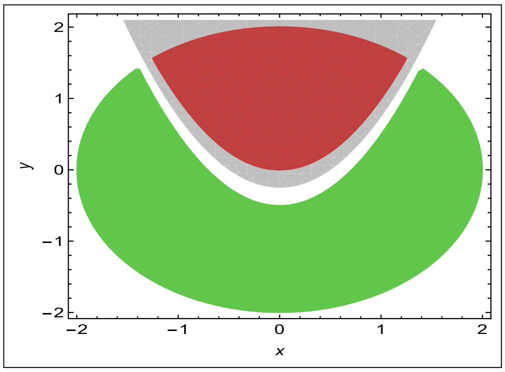

Fig. 3. An illustration of $\texttt{CI}(0.01, 0.01)$ for Example 1. Red, green and gray regions are respectively the set $\{(x, y) \mid \phi(x, y)\}$, $\{(x, y) \mid \psi(x, y)\}$ and $\{(x, y) \mid h(x, y) > 0\}$ (Color figure online).

Example 2. In order to demonstrate the applicability of our method to formulas beyond polynomial ones, we consider nonlinear formulas:

$$\phi(x, y) : f_1(x, y) \le 0 \wedge f_2(x, y) \le 0$$
$$\psi(x, y) : (g_{1,1}(x, y) \le 0 \wedge g_{1,2}(x, y) \ge 0) \vee (g_{2,1}(x, y) \le 0 \wedge g_{2,2}(x, y) \le 0)$$

where $f_1(x, y) = y^2 + x^2 - 4$, $f_2(x, y) = y^8 - 2y^4x^2 + 6y^4 + y^2 + \sin(x)^4 - 5x^2 + 5$, $g_{1,1}(x, y) = y^2 + x^2 - 4, g_{1,2}(x, y) = y^2 - x^2 - 1, g_{2,1}(x, y) = y^2 + x^2 - 4$ and $g_{2,2}(x, y) = 2y^2 - 2yx^2 - 2y + x^4 + 3x^2 - 3$.

Via solving (14), we obtain a $\texttt{CI}(0.001, 0.001)$:

$$\begin{aligned} h(x, y) = &-9.99999999941 + 0.0280147362282x \\ &- 0.312255569319y + 10x^2 - 0.0107265793759xy - 6.16535622183y^2. \end{aligned} \tag{17}$$

That is, with at least 99.9% confidence, the probability that $h(x, y) > 0$ is an interpolant for ϕ and ψ is larger than or equal to 0.999.

Since the formula ϕ is non-polynomial, we use the satisfiability checker iSAT3 [10] to check that $h(x, y)$ in (17) is a true (not just probably approximately correct) interpolant.

The plots of $\{(x, y) \mid \psi(x, y)\}$, $\{(x, y) \mid \phi(x, y)\}$ and $\{(x, y) \mid h(x, y) > 0\}$ are presented in Fig. 4.

Example 3. To demonstrate the applicability of our approach to formulas with high-dimensional variables, we consider a scalable example with variables of high dimension of 100.

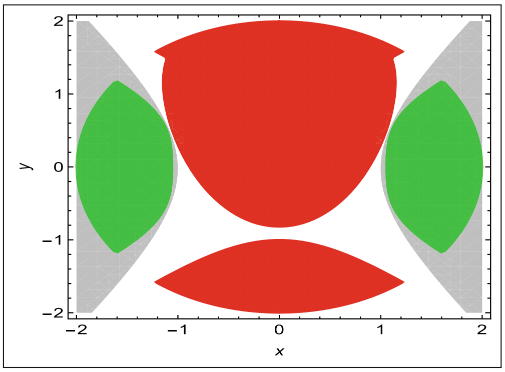

Fig. 4. An illustration of $\mathtt{CI}(0.001, 0.001)$ for Example 2. Red, green and gray regions are respectively the set $\{(x, y) \mid \psi(x, y)\}$, $\{(x, y) \mid \phi(x, y)\}$ and $\{(x, y) \mid h(x, y) > 0\}$. (Color figure online)

$$\begin{aligned}
&\phi(\boldsymbol{x}):\\
&\wedge_{i=1}^{100} [x_i \le 0 \wedge x_i + 0.5 \ge 0].\\
&\psi(\boldsymbol{x}):\\
&\wedge_{i=1}^{100} [-x_i - 0.9 \le 0 \wedge x_i + 0.4 \le 0]\\
&\wedge \sum_{i=1}^{100} x_i - e^{x_{10}} + \cos(x_{50}) \sin(x_{50}) + 0.9 \le 0.
\end{aligned}$$

The interpolant template is $c_0 + \sum_{i=1}^{100} c_i x_i$. Algorithm 1 returns a $\mathtt{CI}(0.01, 10^{-12})$. The satisfiability checker iSAT3 [10] fails to check whether the computed $\mathtt{CI}(0.01, 10^{-12})$ is a true interpolant. However, according to (6), we have that the probability of states in $\{\boldsymbol{x} \mid \phi(\boldsymbol{x})\}$ such that such that $\phi(\boldsymbol{x}) \wedge \psi(\boldsymbol{x}) \models \perp$ is larger than $\max\{0, 1 - (0.01 + 0.01\frac{0.5^{100}}{0.5^{100}})\} \ge 0.98$, with confidence at least $1 - 10^{-12}$.

The dimensionality of this example demonstrates that our approach opens up a promising prospect for interpolant synthesis of formulas with high-dimensional variables by selecting appropriate ϵ, β and interpolant templates.

5 Conclusion

In this paper we investigated the generation of interpolants for two Boolean formulas in the framework of PAC learning, attempting to fight against the "curse of dimensionality" suffered by traditional formal methods. A new concept, called PAC Craig interpolants, was introduced to characterize the relationship between two formulas using violation levels and confidence levels. Based on scenario approaches, we could construct a linear programming formulation of the instantiation problem for an interpolant template and compute a PAC interpolant.

Using the computed PAC interpolant, we could characterize how inconsistent the two given formulas are. Besides, one important consequence of our approach is that in contrast to traditional methods for computing interpolants, our method scales well to the high-dimensional formulas. Three examples demonstrated the performance of our approach.

In our future work we would extend our method to safety verification of hybrid systems, as well as the generation of PAC interpolants for Boolean formulas with uncommon variables.

References

1. Andersen, M., Dahl, J., Liu, Z., Vandenberghe, L.: Interior-point methods for large-scale cone programming. In: Optimization for Machine Learning, pp. 55–83 (2011)
2. Benhamou, F., Granvilliers, L.: Continuous and interval constraints. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence, vol. 2, pp. 571–603 (2006)
3. Calafiore, G.C., Campi, M.C.: The scenario approach to robust control design. IEEE Trans. Autom. Control **51**(5), 742–753 (2006)
4. Campi, M.C., Garatti, S., Prandini, M.: The scenario approach for systems and control design. Ann. Rev. Control **33**(2), 149–157 (2009)
5. Chen, M., Wang, J., An, J., Zhan, B., Kapur, D., Zhan, N.: NIL: learning nonlinear interpolants. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 178–196. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_11
6. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient interpolant generation in satisfiability modulo theories. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 397–412. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_30
7. Craig, W.: Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symb. Logic **22**(3), 250–268 (1957)
8. Dai, L., Xia, B., Zhan, N.: Generating non-linear interpolants by semidefinite programming. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 364–380. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_25
9. Fitch, J.: Solving algebraic problems with REDUCE. J. Symb. Comput. **1**(2), 211–227 (1985)
10. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. J. Satisf. Boolean Model. Comput. **1**, 209–236 (2007)
11. Gan, T., Dai, L., Xia, B., Zhan, N., Kapur, D., Chen, M.: Interpolation synthesis for quadratic polynomial inequalities and combination with EUF. In: IJCAR 2016, pp. 195–212 (2016)
12. Gan, T., Xia, B., Xue, B., Zhan, N., Dai, L.: Nonlinear Craig interpolant generation. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 415–438. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_20
13. Gao, S., Kong, S., Clarke, E.M.: Proof generation from delta-decisions. In: SYNASC 2014, pp. 156–163 (2014)
14. Gao, S., Zufferey, D.: Interpolants in nonlinear theories over the reals. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 625–641. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_41

15. Gearhart, J.L., Adair, K.L., Detry, R.J., Durfee, J.D., Jones, K.A., Martin, N.: Comparison of open-source linear programming solvers. Technical report SAND2013-8847 (2013)
16. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63166-6_10
17. Haussler, D.: Probably approximately correct learning. University of California, Santa Cruz, Computer Research Laboratory (1990)
18. Henzinger, T., Jhala, R., Majumdar, R., McMillan, K.: Abstractions from proofs. In POPL **2004**, 232–244 (2004)
19. Kapur, D., Majumdar, R., Zarba, C.: Interpolation for data structures. In: FSE 2006, pp. 105–116 (2006)
20. Kovács, L., Voronkov, A.: Interpolation and symbol elimination. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 199–213. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_17
21. Krajíček, J.: Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. J. Symb. Logic **62**(2), 457–486 (1997)
22. Kupferschmid, S., Becker, B.: Craig interpolation in the presence of non-linear constraints. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 240–255. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24310-3_17
23. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_1
24. McMillan, K.: An interpolating theorem prover. Theor. Comput. Sci. **345**(1), 101–121 (2005)
25. McMillan, K.L.: Quantified invariant generation using an interpolating saturation prover. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 413–427. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_31
26. Pudlăk, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. J. Symb. Logic **62**(3), 981–998 (1997)
27. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. J. Symb. Comput. **45**(11), 1212–1233 (2010)
28. Sharma, R., Nori, A.V., Aiken, A.: Interpolants as classifiers. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 71–87. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_11
29. Steyvers, M.: Computational statistics with MATLAB (2011)
30. Törnblom, J., Nadjm-Tehrani, S.: Formal verification of random forests in safety-critical applications. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2018. CCIS, vol. 1008, pp. 55–71. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-12988-0_4
31. Xue, B., Easwaran, A., Cho, N.-J., Fränzle, M.: Reach-avoid verification for non-linear systems based on boundary analysis. IEEE Trans. Autom. Control **62**(7), 3518–3523 (2016)
32. Xue, B., Fränzle, M., Zhao, H., Zhan, N., Easwaran, A.: Probably approximate safety verification of hybrid dynamical systems. In: Ait-Ameur, Y., Qin, S. (eds.) ICFEM 2019. LNCS, vol. 11852, pp. 236–252. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32409-4_15
33. Xue, B., Liu, Y., Ma, L., Zhang, X., Sun, M., Xie, X.: Safe inputs approximation for black-box systems. In: ICECCS 2019, pp. 180–189. IEEE (2019)

34. Xue, B., Zhang, M., Easwaran, A., Li, Q.: PAC model checking of black-box continuous-time dynamical systems. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (IEEE TCAD) (2020, to appear)
35. Yorsh, G., Musuvathi, M.: A combination method for generating interpolants. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 353–368. Springer, Heidelberg (2005). https://doi.org/10.1007/11532231_26
36. Zhan, N., Wang, S., Zhao, H.: Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-47016-0
37. Zhao, H., Zhan, N., Kapur, D., Larsen, K.G.: A "hybrid" approach for synthesizing optimal controllers of hybrid systems: a case study of the oil pump industrial example. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 471–485. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32759-9_38