Generating Non-linear Interpolants by Semidefinite Programming*

Liyun Dai[1,3], Bican Xia[1], and Naijun Zhan[2]

[1] LMAM & School of Mathematical Sciences, Peking University
[2] State Key Laboratory of Computer Science, Institute of Software, CAS
[3] Beijing International Center for Mathematical Research, Peking University
dailiyun@pku.edu.cn, xbc@math.pku.edu.cn, znj@ios.ac.cn

Abstract. Interpolation-based techniques have been widely and successfully applied in the verification of hardware and software, e.g., in bounded-model checking, CEGAR, SMT, etc., in which the hardest part is how to synthesize interpolants. Various work for discovering interpolants for propositional logic, quantifier-free fragments of first-order theories and their combinations have been proposed. However, little work focuses on discovering polynomial interpolants in the literature. In this paper, we provide an approach for constructing non-linear interpolants based on semidefinite programming, and show how to apply such results to the verification of programs by examples.

Keywords: Craig interpolant, Positivstellensatz Theorem, semidefinite programming, program verification.

1 Introduction

It becomes a grand challenge to guarantee the correctness of software, as our modern life depends more and more on computerized systems. There are lots of verification techniques based either on model-checking [1], theorem proving [2,3], abstract interpretation [4] or their combination, which have been invented for the verification of hardware and software, like bounded model-checking [5], CEGAR [6], satisfiability modulo theories (SMT) [7], etc. Scalability is a bottleneck of these techniques, as many of real softwares are very complex with different features like complicated data structures, concurrency, distributed, real-time and hybrid etc. Interpolation-based techniques provide a powerful mechanism for local and modular reasoning, which indeed improves the scalability of these techniques, in which the notion of Craig interpolants plays a key role.

Interpolation-based local and modular reasoning was first applied in theorem proving by Nelson and Oppen [8], called Nelson-Oppen method. The basic idea of Nelson-Oppen method is to reduce the satisfiability (validity) of a composite theory into the ones of its component theories whose satisfiability (validity) have been obtained. The hardest part of the method, which also determines the efficiency of the method, is to construct a formula using the common part of the component theories for a given formula

* The first two authors are funded by NSFC-11271034, NSFC-11290141 and SYSKF1207, and the third author is funded by NSFC-91118007, NSFC-60970031 and 2012ZX03039-004.

of the composite theory with Craig's Interpolation Theorem [9]. In the past decade, the Nelson-Oppen method was further extended to SMT which is based on DPLL [10] and Craig's Interpolation Theorem for combining different decision procedures in order to verify a property of programs with complicated data structures. For instance, Z3 [11] integrates more than 10 different decision procedures up to now, including propositional logic, equality logic with uninterpreted functions, Presburger arithmetic, array logic, difference arithmetic, bit vector logic etc.

In recent years, it is noted that interpolation based local and modular reasoning is quite helpful to improve the scalability of model-checking, in particular for bounded model-checking of systems with finite or infinite states [5,12,13], CEGAR [14,15], etc. McMillan first considered how to combine Craig interpolants with bounded model-checking to verify infinite state systems [12]. The basic idea of his approach is to generate invariants using Craig interpolants, so that it can be claimed that an infinite state system satisfies a property after k steps in model-checking whenever an invariant, which is strong enough to guarantee the property, is obtained. In [14,15,16], how to apply the local property of Craig interpolants generated from a counter-example to refine the abstract model in order to exclude the spurious counter-example in CEGAR was investigated. Meanwhile, in [17], using interpolation technique to generate a set of atomic predicates as the base of machine-learning based verification technique was investigated by Wang et al.

Obviously, synthesizing Craig interpolants is the cornerstone of interpolation based techniques. In fact, many approaches have been proposed in the literature. In [13], McMillan presented a method for deriving Craig interpolants from proofs in the quantifier-free theory of linear inequality and uninterpreted function symbols, and based on which an interpolating theorem prover was provided. In [15], Henzinger et al. proposed a method to synthesize Craig interpolants for a theory with arithmetic and pointer expressions, as well as call-by-value functions. In [18], Yorsh and Musuvathi presented a combination method to generate Craig interpolants for a class of first-order theories. In [19], Kapur et al presented different efficient procedures to construct interpolants for the theories of arrays, sets and multisets using the reduction approach. Rybalchenko and Sofronie-Stokkermans [20] proposed an approach to reducing the synthesis of Craig interpolants of the combined theory of linear arithmetic and uninterpreted function symbols to constraint solving.

However, in the literature, there is little work on how to synthesize non-linear interpolants, except that in [21] Kupferschmid and Becker provided a method to construct non-linear Craig Interpolant using iSAT, which is a variant of SMT solver based on interval arithmetic.

In this paper we investigate how to construct non-linear interpolants. The idea of our approach is as follows: Firstly, we reduce the problem of generating interpolants for two arbitrary polynomial formulas to that of generating interpolants for two semi-algebraic systems (SASs), which is a conjunction of a set of polynomial equations, inequations and inequalities (see the definition later). Then, by **Positivstellensatz Theorem** [22], there exists a witness to indicate that the considered two SASs do not have common real solutions if their conjunction is unsatisfiable. Parrilo in [23,24] gave an approach for constructing the witness by applying semidefinite programming [25]. Our algorithm invokes Parrilo's method as a subroutine. Our purpose is to construct Craig interpolants, so we need to obtain a special witness. In general, we cannot guarantee the existence of the special witness, which means that our approach is sound but incomplete. However, we discuss that if the considered two SASs meet the *Archimedean condition*, (e.g. each variable occurring in the SASs is bounded, which is a reasonable assumption in practice), our approach is not only sound, but also complete. We demonstrate our approach by some examples, in particular, we show how to apply the results to program verification by examples.

The complexity of our approach is polynomial in $ud\binom{n+d/2}{n}\binom{n+d}{n}$, where u is the number of polynomial constraints in the considered problem, n is the number of variables, and d is the highest degree of polynomials and interpolants. So, the complexity of our approach is polynomial in d for a given problem in which n and u are fixed.

Structure of the Paper: The rest of the paper is organized as follows. By a running example, we sketch our approach and show how to apply it to program verification in Section 2. Some necessary preliminaries are introduced in Section 3. A sound but incomplete algorithm for synthesizing non-linear interpolants in general case is described in Section 4. Section 5 provides a practical algorithm for systems containing only non-strict inequalities and satisfying the Archimedean condition. Section 6 focuses on the correctness and complexity analysis of our approach. Our implementation and experimental results are briefly reported in Section 7. Section 8 summarizes the paper and discusses future work.

2 An Overview of Our Approach

In this section, we sketch our approach and show how to apply our results to program verification by an example.

```
1  IF ($x*x+y*y<1$)                       $g_1 = 1 - x^2 - y^2 > 0$
2  {   /* initial values */
3    WHILE ($x*x+y*y<3$)                  $g_2 = 3 - x^2 - y^2 > 0$
4    {   $x := x*x+y-1$;                  $f_1 = x^2 + y - 1 - x' = 0$
5        $y := y+x*y+1$;                  $f_2 = y + x'y + 1 - y' = 0$
6        IF ($x*x-2*y*y-4>0$)             $g_3 = x'^2 - 2y'^2 - 4 > 0$
7            /* unsafe area */
8            error();    } }
```

Code 1.1

Consider the program in Code 1.1 (left part). This program tests the initial value of x and y at line 1, afterwards executes the *while loop* with $x^2+y^2<3$ as the loop condition. The body of the while loop contains two assignments and an **if** statement in sequence. The property we wish to check is that **error()** procedure will never be executed. Suppose there is an execution $1 \to 3 \to 4 \to 5 \to 6 \to 8$. We can encode such an execution by the formulas in Code 1.1 (right part). Note that in these formulas we use unprimed and primed versions of each variable to represent the values of the variable before and after updating respectively. Obviously, the execution is infeasible iff the conjunction of these formulas is unsatisfiable. Let $\phi \triangleq g_1 > 0 \wedge f_1 = 0 \wedge f_2 = 0$[1] and $\psi \triangleq g_3 > 0$. To show $\phi \wedge \psi$ is unsatisfiable, we need to construct an *interpolant* θ for ϕ and ψ, i.e., $\phi \Rightarrow \theta$ and $\theta \Rightarrow \neg\psi$. If there exist $\delta_1, \delta_2, \delta_3, h_1, h_2$ such that

$$g_1\delta_1 + f_1h_1 + f_2h_2 + g_3\delta_2 + \delta_3 = -1,$$

where $\delta_1, \delta_2, \delta_3 \in \mathbb{R}[x, y, x', y']$ are sums of squares and $h_1, h_2 \in \mathbb{R}[x, y, x', y']$, then $\theta \triangleq g_3\delta_2 + \frac{1}{2} \leq 0$ is such an interpolant for ϕ and ψ. In this example, applying our tool `AiSat`, we obtain in 0.025 seconds that

$$\begin{aligned}
h_1 &= -290.17 - 56.86y' + 1109.95x' + 37.59y - 32.20yy' + 386.77yx' + 203.88y^2 + 107.91x^2,\\
h_2 &= -65.71 + 0.39y' + 244.14x' + 274.80y + 69.33yy' - 193.42yx' - 88.18y^2 - 105.63x^2,\\
\delta_1 &= 797.74 - 31.38y' + 466.12y'^2 + 506.26x' + 79.87x'y' + 402.44x'^2 + 104.43y\\
&\quad +41.09yy' - 70.14yx' + 451.64y^2 + 578.94x^2\\
\delta_2 &= 436.45,\\
\delta_3 &= 722.62 - 91.59y' + 407.17y'^2 + 69.39x' + 107.41x'y' + 271.06x'^2 + 14.23y + 188.65yy'\\
&\quad +69.33yy'^2 - 600.47yx' - 226.01yx'y' + 142.62yx'^2 + 325.78y^2 - 156.69y^2y' + 466.12y^2y'^2\\
&\quad +10.54y^2x'y' + 595.87y^2x'^2 - 11.26y^3 + 41.09y^3y' + 18.04y^3x' + 451.64y^4 + 722.52x^2\\
&\quad -80.15x^2y' + 466.12x^2y'^2 - 495.78x^2x' + 79.87x^2x'y' + 402.44x^2x'^2 + 64.57x^2y\\
&\quad +241.99y^2x' + 73.29x^2yy' - 351.27x^2yx' + 826.70x^2y^2 + 471.03x^4.
\end{aligned}$$

Note that δ_1 can be represented as $923.42(0.90 + 0.7y - 0.1y' + 0.43x')^2 + 252.84(0.42 - 0.28y + 0.21y' - 0.84x')^2 + 461.69(-0.1 - 0.83y + 0.44y' + 0.34x')^2 + 478(-0.06 + 0.48y + 0.87y' + 0.03x')^2 + 578.94(x)^2$. Similarly, δ_2 and δ_3 can be represented as sums of squares also.

Moreover, using the approach in [26], we can prove θ is an inductive invariant of the loop, therefore, **error()** will never be executed.

3 Theoretical Foundations

In this section, for self-containedness, we briefly introduce some basic notions and mathematical theories, based on which our approach is developed.

3.1 Problem Description

Definition 1 (Interpolants). *A theory $\mathcal{T}$ has interpolant if for all formulae ϕ and ψ in the signature of $\mathcal{T}$, if $\models_{\mathcal{T}} (\phi \wedge \psi) \Rightarrow \bot$, then there exists a formula Θ that contains only symbols that ϕ and ψ share such that $\phi \models_{\mathcal{T}} \Theta$ and $\models_{\mathcal{T}} (\psi \wedge \Theta) \Rightarrow \bot$.*

In what follows, we denote by $\mathbf{x}$ a variable vector[2]$(x_1, \cdots, x_n)$ in $\mathbb{R}^n$, and by $\mathbb{R}[\mathbf{x}]$ the polynomial ring with real coefficients in variables $\mathbf{x}$.

A semi-algebraic system (SAS) $\mathcal{T}(\mathbf{x})$ is of the form $\bigwedge_{j=0}^{k} f_j(\mathbf{x}) \rhd_j 0$, where f_j are polynomials in $\mathbb{R}[\mathbf{x}]$ and $\rhd_j \in \{=, \neq, \geq\}$. Clearly, any polynomial formula ϕ can be represented as the disjunction of several SASs. Let $\phi_1 = \bigvee_{t=1}^{m} \mathcal{T}_{1t}(\mathbf{x}), \phi_2 = \bigvee_{l=1}^{n} \mathcal{T}_{2l}(\mathbf{x})$ be

[1] As $g_1 > 0 \Rightarrow g_2 > 0$, we ignore $g_2 > 0$ in ϕ.

[2] In the following, we also abuse $\mathbf{x}$ to denote the set of variables $\{x_1, \ldots, x_n\}$.

two polynomial formulas in $\mathbb{R}[\mathbf{x}]$, and $\phi_1 \wedge \phi_2 \models \bot$, i.e., ϕ_1 and ϕ_2 do not share any real solutions. Then, the problem to be considered in this paper is how to find another polynomial formula I such that $\phi_1 \models I$ and $I \wedge \phi_2 \models \bot$.

It is easy to show that if, for each t and l, there is an interpolant I_{tl} for SASs $\mathcal{T}_{1t}(\mathbf{x})$ and $\mathcal{T}_{2l}(\mathbf{x})$, then $I = \bigvee_{t=1}^{m} \bigwedge_{l=1}^{n} I_{tl}$ is an interpolant of ϕ_1 and ϕ_2. Thus, we only need to consider how to construct interpolants for two SASs in the rest of this paper.

Discussions on Common Variables: In reality, it is more likely that ϕ_1 and ϕ_2 in the above problem description are over different sets of variables, say over $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively, and $\mathbf{x}_1 \neq \mathbf{x}_2$. So, we need to reduce the interpolant generation of ϕ_1 and ϕ_2 to that of another two formulas that share the common variables first. A simple way to achieve this is through introducing $\exists \mathbf{x}_1 - \mathbf{x}_2$ over $\phi_1(\mathbf{x}_1)$ and $\exists \mathbf{x}_2 - \mathbf{x}_1$ over $\phi_2(\mathbf{x}_2)$, then apply quantifier elimination to $\exists \mathbf{x}_1 - \mathbf{x}_2 . \phi_1(\mathbf{x}_1)$ and $\exists \mathbf{x}_2 - \mathbf{x}_1 . \phi_2(\mathbf{x}_2)$, and obtain two formulas on the common variables $\mathbf{x}_1 \cap \mathbf{x}_2$. Obviously, the interpolant generation problem of ϕ_1 and ϕ_2 is reduced to that of the two resulted formulas.

But in practice, as the cost of quantifier elimination is very high, we can adopt the following more efficient way. For each $x \in \mathbf{x}_1 - \mathbf{x}_2$, if x is a local variable introduced in the respective program, we always have an equation $x = h$ corresponding to the assignment to x (possibly the composition of a sequence of assignments to x); otherwise, x is a global variable, but only occurring in ϕ_1. For this case, we introduce an equation $x = x$ to ϕ_2; Symmetrically, each $x \in \mathbf{x}_2 - \mathbf{x}_1$ can be coped with similarly. The detailed discussion can be found in the full version of this paper [27].

So, in what follows, we assume any two SASs share the common variables if no otherwise stated.

3.2 Real Algebraic Geometry

In this subsection, we introduce some basic notions and results on real algebraic geometry, that will be used later.

Definition 2 (ideal). *Let $\mathcal{I}$ be an ideal in $\mathbb{R}[\mathbf{x}]$, that is, $\mathcal{I}$ is an additive subgroup of $\mathbb{R}[\mathbf{x}]$ satisfying $fg \in \mathcal{I}$ whenever $f \in \mathcal{I}$ and $g \in \mathbb{R}[\mathbf{x}]$. Given $h_1, \dots, h_m \in \mathbb{R}[\mathbf{x}]$, $\langle h_1, \dots, h_m \rangle = \left\{ \sum_{j=1}^{m} u_j h_j \mid u_1, \dots, u_m \in \mathbb{R}[\mathbf{x}] \right\}$ denotes the ideal generated by $h_1, \dots, h_m$.*

Definition 3 (multiplicative monoid). *Given a polynomial set P, let Mult(P) be the* multiplicative monoid *generated by P, i.e., the set of finite products of the elements of P (including the empty product which is defined to be 1).*

Definition 4 (Cone). *A* cone *$\mathcal{C}$ of $\mathbb{R}[\mathbf{x}]$ is a subset of $\mathbb{R}[\mathbf{x}]$ satisfying the following conditions: **(i)** $p_1, p_2 \in \mathcal{C} \Rightarrow p_1 + p_2 \in \mathcal{C}$; **(ii)** $p_1, p_2 \in \mathcal{C} \Rightarrow p_1 p_2 \in \mathcal{C}$; **(iii)** $p \in \mathbb{R}[\mathbf{x}] \Rightarrow p^2 \in \mathcal{C}$.*

Given a set $P \subseteq \mathbb{R}[\mathbf{x}]$, let $\mathcal{C}(P)$ be the smallest cone of $\mathbb{R}[\mathbf{x}]$ that contains P. It is easy to see that $\mathcal{C}(\emptyset)$ corresponds to the polynomials that can be represented as a sum of squares, and is the smallest cone in $\mathbb{R}[\mathbf{x}]$, i.e., $\{ \sum_{i=1}^{s} p_i^2 \mid p_1, \dots, p_s \in \mathbb{R}[\mathbf{x}] \}$, denoted by **SOS**. For a finite set $P \subseteq \mathbb{R}[\mathbf{x}]$, $\mathcal{C}(P)$ can be represented as:

$$\mathcal{C}(P) = \{ \sum_{i=1}^{r} q_i p_i \mid q_1, \dots, q_r \in \mathcal{C}(\emptyset), p_1, \dots, p_r \in \mathit{Mult}(P) \}.$$

Positivstellensatz Theorem, due to Stengle [22], is an important theorem in real algebraic geometry. It states that, for a given SAS, either the system has real solution(s), or there exists a polynomial to indicate that the system has no solution.

Theorem 1 (Positivestellensatz Theorem, [22]). *Let $(f_j)_{j=1}^s$, $(g_k)_{k=1}^t$, $(h_l)_{l=1}^u$ be finite families of polynomials in $\mathbb{R}[\mathbf{x}]$. Denote by $\mathcal{C}$ the cone generated by $(f_j)_{j=1}^s$, Mult the multiplicative monoid generated by $(g_k)_{k=1}^t$, and $\mathcal{I}$ the ideal generated by $(h_l)_{l=1}^u$. Then the following two statements are equivalent:*

1. *the SAS* $\begin{cases} f_1(\mathbf{x}) \geq 0, & \cdots, & f_s(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, & \cdots, & g_t(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, & \cdots, & h_u(\mathbf{x}) = 0 \end{cases}$ *has no real solutions;*
2. *there exist $f \in \mathcal{C}$, $g \in$ Mult, $h \in \mathcal{I}$ such that $f + g^2 + h \equiv 0$.*

3.3 Semidefinite Programming

In [22], Stengle did not provide a constructive proof for Theorem 1. However, Parrilo in [23,24] provided a constructive way to obtain the witness, which is based on semidefinite programming. Parrilo's result will be the starting point of our method, so we briefly review semidefinite programming below. We use Sym_n to denote the set of $n \times n$ real symmetric matrices, and $\deg(f)$ the highest total degree of f for a given polynomial f in the sequel.

Definition 5 (Positive semidefinite matrix). *A matrix $M \in Sym_n$ is called* positive semidefinite*, denoted by $M \succeq 0$, if $\mathbf{x}^T M \mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$.*

Definition 6. *The* inner product *of two matrices $A = (a_{ij}), B = (b_{ij}) \in \mathbb{R}^{n \times n}$, denoted by $\langle A, B \rangle$, is defined by $Tr(A^T B) = \sum_{i,j=1}^n a_{ij} b_{ij}$.*

Definition 7 (Semidefinite programming (SDP)). *The standard (primal) and dual forms of a* **SDP** *are respectively given in the following:*

$$p^* = \inf_{X \in Sym_n} \langle C, X \rangle \text{ s.t. } X \succeq 0,\ \langle A_j, X \rangle = b_j\ (j = 1, \ldots, m) \tag{1}$$

$$d^* = \sup_{y \in \mathbb{R}^m} \mathbf{b}^T \mathbf{y} \text{ s.t. } \sum_{j=1}^m y_j A_j + S = C,\ S \succeq 0, \tag{2}$$

where $C, A_1, \ldots, A_m, S \in Sym_n$ and $\mathbf{b} \in \mathbb{R}^m$.

There are many efficient algorithms to solve **SDP** such as interior-point method. We present a basic path-following algorithm to solve (1) in the following.

Definition 8 (Interior point for SDP).

$$\begin{aligned} intF_p &= \{X : \langle A_i, X \rangle = b_i\ (i = 1, \ldots, m),\ X \succ 0\}, \\ intF_d &= \left\{ (\mathbf{y}, S) : S = C - \sum_{i=1}^m A_i y_i \succ 0 \right\}, \\ intF &= intF_p \times intF_d. \end{aligned}$$

Obviously, $\langle C, X \rangle - \mathbf{b}^T \mathbf{y} = \langle X, S \rangle > 0$ for all $(X, \mathbf{y}, S) \in intF$. Especially, we have $d^* \leq p^*$. So the soul of interior-point method to compute p^* is to reduce $\langle X, S \rangle$ incessantly and meanwhile guarantee $(X, \mathbf{y}, S) \in intF$.

Algorithm 1. `Interior_Point_Method`

```
input : $C, A_j, b_j$ $(j = 1, \ldots, m)$ as in (1) and a threshold $c$
output: $p^*$
1 Given a $(X, \mathbf{y}, S) \in intF$ and $XS = \mu I$;
  /* $\mu$ is a positive constant and $I$ is the identity matrix. */
2 while $\mu > c$ do
3     $\mu = \gamma\mu$;
      /* $\gamma$ is a fixed positive constant less than one */
4     use Newton iteration to solve $(X, \mathbf{y}, S) \in intF$ with $XS = \mu I$;
5 end
```

3.4 Constructive Proof of Theorem 1 Using SDP

Given a polynomial $f(\mathbf{x})$ of degree no more than $2d$, f can be rewritten as $f = Z^TQZ$ where Z is a vector consists of all monomials of degrees no more than d, e.g., $Z = \left[1, x_1, x_2, \ldots, x_n, x_1x_2, x_2x_3, \ldots, x_n^d\right]^T$, and $Q = \begin{pmatrix} a_1 & \frac{a_{x_1}}{2} & \cdots & \frac{a_{x_n}}{2} \\ \frac{a_{x_1}}{2} & a_{x_1^2} & \cdots & \frac{a_{x_1x_n}}{2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{x_n}}{2} & \frac{a_{x_1x_n}}{2} & \cdots & a_{x_n^d} \end{pmatrix}$ is a symmetric matrix. Note that here Q is not unique in general. Moreover, $f \in \mathcal{C}(\emptyset)$ iff there is a positive semidefinite constant matrix Q such that $f(\mathbf{x}) = Z^TQZ$. The following lemma is an obvious fact on how to use the above notations to express the polynomial multiplication.

Lemma 1. *Given polynomials $f_1, \ldots, f_n, g_1, \ldots, g_n$, assume $\sum_{i=1}^n f_ig_i = \sum_{i=1}^s c_im_i$, where $c_i \in \mathbb{R}$ and m_is are monomials, $g_i = Z^TQ_{2i}Z$, and $Q_2 = diag(Q_{21}, \ldots, Q_{2n})$. Then there exist symmetric matrices $Q_{11}, \ldots, Q_{1s}$ such that $c_i = \langle Q_{1i}, Q_2\rangle$, i.e., $\sum_{i=1}^n f_ig_i = \sum_{i=1}^s \langle Q_{1i}, Q_2\rangle m_i$, in which Q_{1i} can be constructed from the coefficients of $f_1, \ldots, f_n$.*

Example 1. Let $f = a_{20}x_1^2 + a_{11}x_1x_2 + a_{02}x_2^2$ and $g = b_{00} + b_{10}x_1 + b_{01}x_2$. Then, $fg = \langle Q_{11}, Q_2\rangle\, x_1^2 + \langle Q_{12}, Q_2\rangle\, x_1x_2 + \langle Q_{13}, Q_2\rangle\, x_2^2 + \langle Q_{14}, Q_2\rangle\, x_1x_2^2\, \langle Q_{15}, Q_2\rangle\, x_1^2x_2 + \langle Q_{16}, Q_2\rangle x_2^3 + \langle Q_{17}, Q_2\rangle x_1^3$, where

$$Q_2 = \begin{pmatrix} b_{00} & \frac{b_{10}}{2} & \frac{b_{01}}{2} \\ \frac{b_{10}}{2} & 0 & 0 \\ \frac{b_{01}}{2} & 0 & 0 \end{pmatrix},\quad Q_{11} = \begin{pmatrix} a_{20} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},\quad Q_{12} = \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},\quad Q_{13} = \begin{pmatrix} a_{02} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$Q_{14} = \begin{pmatrix} 0 & \frac{a_{02}}{2} & \frac{a_{11}}{2} \\ \frac{a_{02}}{2} & 0 & 0 \\ \frac{a_{11}}{2} & 0 & 0 \end{pmatrix}, Q_{15} = \begin{pmatrix} 0 & \frac{a_{11}}{2} & \frac{a_{20}}{2} \\ \frac{a_{11}}{2} & 0 & 0 \\ \frac{a_{20}}{2} & 0 & 0 \end{pmatrix}, Q_{16} = \begin{pmatrix} 0 & 0 & \frac{a_{02}}{2} \\ 0 & 0 & 0 \\ \frac{a_{02}}{2} & 0 & 0 \end{pmatrix}, Q_{17} = \begin{pmatrix} 0 & \frac{a_{02}}{2} & 0 \\ \frac{a_{02}}{2} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

Back to Theorem 1. We show how to find $f \in \mathcal{C}$, $g \in Mult$, $h \in \mathcal{I}$ such that $f + g^2 + h \equiv 0$ via **SDP** solving. First, since $f \in \mathcal{C}$, f can be written as a sum of the products of some known polynomials and some unknown SOSs. Second, $h \in \mathcal{I}(\{h_1, \ldots, h_u\})$ is equivalent to $h = h_1p_1 + \cdots + h_up_u$, which is further equivalent to $h = h_1(q_{11} - q_{12}) + \cdots + h_u(q_{u1} - q_{u2})$, where $p_i, q_{ij} \in \mathbb{R}[\mathbf{x}]$ and $q_{ij} \in \mathbf{SOS}$[3]. Third, fix an integer $d > 0$, let

[3] For example, let $q_{i1} = (\frac{1}{4}p_i + 1)^2, q_{i2} = (\frac{1}{4}p_i - 1)^2$.

Algorithm 2. `Certificate_Generation`

```
input : ${f_1, ..., f_n}, g, {h_1, ..., h_u}, b$
output: either ${p_0, ..., p_n}$ and ${q_1, ..., q_u}$ such that
        $1 + p_0 + p_1 f_1 + \cdots + p_n f_n + g + q_1 h_1 + \cdots + q_u h_u \equiv 0$, or NULL
1  Let $q_{11}, q_{12}, q_{21}, q_{22}, \ldots, q_{u1}, q_{u2} \in$ SOS with $\deg(q_{i1}) \le b$ and $\deg(q_{i2}) \le b$ be
   undetermined SOS polynomials;
2  Let $p_0, p_1, \ldots, p_n \in$ SOS with $\deg(p_i) \le b$ be undetermined SOS polynomials;
3  Let $f = 1 + p_0 + p_1 f_1 + \cdots + p_n f_n + g + (q_{11} - q_{12})h_1 + \cdots + (q_{u1} - q_{u2})h_u$;
4  for every monomial $m \in f$ do
5      Let $\langle Q_m, Q \rangle = \texttt{coeff}(f, m)$;
       /* Applying Lemma 1                                                    */
       /* coeff$(f, m)$ the coefficient of monomial $m$ in $f$                 */
       /* $Z$ is a monomial vector that contains all monomials
          with coefficient 1 and degree no more than $b/2$                     */
       /* $p_0 = Z^T Q_0 Z, p_1 = Z^T Q_1 Z, \ldots, p_n = Z^T Q_n Z$          */
       /* $q_{i1} = Z^T Q_{i1} Z, q_{i2} = Z^T Q_{i2} Z, i = 1, \ldots, u$     */
       /* $Q = diag(1, Q_0, Q_1, \ldots, Q_n, 1, Q_{11}, Q_{12}, \ldots, Q_{u1}, Q_{u2})$ */
6  end
7  Applying SDP software CSDP to solve whether there exists a semi-definite symmetric
   matrix $Q$ s.t. $\langle Q_m, Q \rangle = 0$ for every monomial $m \in f$
8  if the return of CSDP is feasible then
       /* $q_i = q_{i1} - q_{i2}$                                              */
9      return ${p_0, \ldots, p_n}, {q_1, \ldots, q_u}$
10 else
11     return NULL
12 end
```

$g = (\Pi_{i=1}^{t} g_i)^d$, and then $f + g^2 + h$ can be written as $\sum_{i=1}^{l} f_i' \delta_i$, where l is a constant integer, $f_i' \in \mathbb{R}[\mathbf{x}]$ are known polynomials and $\delta_i \in$ **SOS** are undermined **SOS** polynomials. Therefore, Theorem 1 is reduced to fixing a sufficiently large integer d and finding undetermined **SOS** polynomials δ_i occurring in f, h with degrees no more than $\deg(g^2)$, satisfying $f + g^2 + h \equiv 0$. Based on Lemma 1, this is a **SDP** problem of form (1). The constraints of the **SDP** are of the form $\langle A_j, X \rangle = 0$, where A_j and X correspond to Q_{1j} and Q_2 in Lemma 1, respectively. And Q_2 is a block diag matrix whose blocks correspond to the undetermined **SOS** polynomials in the above discussion.

Theorem 2 ([23]). *Consider a system of polynomial equalities and inequalities of the form in Theorem 1. Then the search for bounded degree Positivstellensatz refutations can be done using semidefinite programming. If the degree bound is chosen to be large enough, then the* **SDP***s will be feasible, and the certificates can be obtained from its solution.*

Algorithm 2 is an implementation of Theorem 2 and we will invoke Algorithm 2 as a subroutine later. Note that Algorithm 2 is a little different from the original one in [24], as here we require that f has 1 as a summand for our specific purpose.

4 Synthesizing Non-linear Interpolants in General Case

As discussed before, we only need to consider how to synthesize interpolants for the following two specific SASs

$$\mathcal{T}_1 = \begin{cases} f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0, \\ g_1(\mathbf{x}) \neq 0, \ldots, g_{t_1}(\mathbf{x}) \neq 0, \\ h_1(\mathbf{x}) = 0, \ldots, h_{u_1}(\mathbf{x}) = 0 \end{cases} \quad \mathcal{T}_2 = \begin{cases} f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0, \\ g_{t_1+1}(\mathbf{x}) \neq 0, \ldots, g_t(\mathbf{x}) \neq 0, \\ h_{u_1+l}(\mathbf{x}) = 0, \ldots, h_u(\mathbf{x}) = 0 \end{cases} \tag{3}$$

where $\mathcal{T}_1$ and $\mathcal{T}_2$ do not share any real solutions.

By Theorems 1&2, there exist $f \in \mathcal{C}(\{f_1, \ldots, f_s\})$, $g \in \mathit{Mult}(\{g_1, \ldots, g_t\})$ and $h \in \mathcal{I}(\{h_1, \ldots, h_u\})$ such that $f + g^2 + h \equiv 0$, where

$$\begin{aligned} g &= \Pi_{i=1}^{t} g_i^{2m}, \\ h &= q_1 h_1 + \cdots + q_{u_1} h_{u_1} + \cdots + q_u h_u, \\ f &= p_0 + p_1 f_1 + \cdots + p_s f_s + p_{12} f_1 f_2 + \cdots + p_{1\ldots s} f_1 \ldots f_s. \end{aligned}$$

in which q_i and p_i are in **SOS**.

If f can be represented by three parts: the first part is an **SOS** polynomial that is greater than 0, the second part is from $\mathcal{C}(\{f_1, \ldots, f_{s_1}\})$, and the last part is from $\mathcal{C}(\{f_{s_1+1}, \ldots, f_s\})$, i.e., $f = p_0 + \sum_{v \subseteq \{1,\ldots,s_1\}} p_v(\Pi_{i \in v} f_i) + \sum_{v \subseteq \{s_1+1,\ldots,s\}} p_v(\Pi_{i \in v} f_i)$, where $\forall \mathbf{x} \in \mathbb{R}^n . p_0(\mathbf{x}) > 0$ and $p_v \in \mathbf{SOS}$. Then let

$$\begin{aligned} f_{\mathcal{T}_1} &= \sum_{v \subseteq 1,\ldots,s_1} p_v \Pi_{i \in v} f_i, & h_{\mathcal{T}_1} &= q_1 h_1 + \cdots + q_{u_1} h_{u_1}, \\ f_{\mathcal{T}_2} &= \sum_{v \subseteq s_1+1,\ldots,s} p_v \Pi_{i \in v} f_i, & h_{\mathcal{T}_2} &= h - h_{\mathcal{T}_1}, \end{aligned}$$
$$q = f_{\mathcal{T}_1} + g^2 + h_{\mathcal{T}_1} + \frac{p_0}{2} = -(f_{\mathcal{T}_2} + h_{\mathcal{T}_2}) - \frac{p_0}{2}.$$

Obviously, we have $\forall \mathbf{x} \in \mathcal{T}_1 . q(\mathbf{x}) > 0$ and $\forall \mathbf{x} \in \mathcal{T}_2 . q(\mathbf{x}) < 0$. Thus, let $I = q(\mathbf{x}) > 0$. We have $\mathcal{T}_1 \models I$ and $I \wedge \mathcal{T}_2 \models \perp$.

Notice that because the requirement on f cannot be guaranteed in general, the above approach is not complete generally. We will discuss under which condition the requirement can be guaranteed in the next section. We implement the above method for synthesizing non-linear interpolants in general case by Algorithm 3.

Example 2. Consider

$$\mathcal{T}_1 = \begin{cases} x_1^2 + x_2^2 + x_3^2 - 2 \geq 0, \\ x_1 + x_2 + x_3 \neq 0, \\ 1.2x_1^2 + x_2^2 + x_1 x_3 = 0 \end{cases} \quad \text{and} \quad \mathcal{T}_2 = \begin{cases} -3x_1^2 - 4x_2^3 - 10x_3^2 + 20 \geq 0, \\ 2x_1 + 3x_2 - 4x_3 \neq 0, \\ x_1^2 + x_2^2 - x_3 - 1 = 0 \end{cases}$$

Algorithm 3. SN_Interpolants

```
input : $\mathcal{T}_1$ and $\mathcal{T}_2$ of the form (3), $b$
output: An interpolant $I$ or NULL
1  $g := \Pi_{k=1}^{t} g_k^2$
2  $g := g^{\lfloor \frac{b}{\deg(g)} \rfloor}$
3  $\{f_{t_1}\} := \{\Pi_{i\in v} f_i$ for $v \subseteq \{1,\dots,s_1\}\}$ ;
4  $\{f_{t_2}\} := \{\Pi_{i\in v} f_i$ for $v \subseteq \{s_1+1,\dots,s\}\}$;
5  sdp:=Certificate_Generation($\{f_{t_1}\} \cup \{f_{t_2}\}, g, \{h_1,\dots,h_u\}, b$)
6  if sdp $\equiv$ NULL then
7      return NULL
8  else
9      $I := \frac{1}{2} + \sum_{v\subseteq\{1,\dots,s_1\}} p_v \Pi_{i\in v} f_i + q_1 h_1 + \cdots + q_{u_1} h_{u_1} + g > 0$;
10     return $I$;
11 end
```

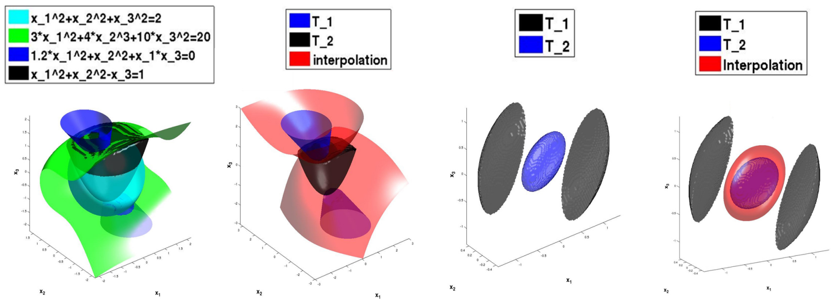

Fig. 1. Examples

Clearly, $\mathcal{T}_1$ and $\mathcal{T}_2$ do not share any real solutions, see Fig. 1 (the first part) [4]. By setting $b = 2$, after calling Certificate_Generation, we obtain an interpolant I with 30 monomials $-14629.26 + 2983.44x_3 + 10972.97x_3^2 + 297.62x_2 + 297.64x_2x_3 + 0.02x_2x_3^2 + 9625.61x_2^2 - 1161.80x_2^2x_3 + 0.01x_2^2x_3^2 + 811.93x_2^3 + 2745.14x_2^4 - 10648.11x_1 + 3101.42x_1x_3 + 8646.17x_1x_3^2 + 511.84x_1x_2 - 1034.31x_1x_2x_3 + 0.02x_1x_2x_3^2 + 9233.66x_1x_2^2 + 1342.55x_1x_2^2x_3 - 138.70x_1x_2^3 + 11476.61x_1^2 - 3737.70x_1^2x_3 + 4071.65x_1^2x_3^2 - 2153.00x_1^2x_2 + 373.14x_1^2x_2x_3 + 7616.18x_1^2x_2^2 + 8950.77x_1^3 + 1937.92x_1^3x_3 - 64.07x_1^3x_2 + 4827.25x_1^4 > 0$, whose figure is depicted in Fig. 1 (the second part). □

5 A Complete Algorithm under Archimedean Condition

Our approach to synthesizing non-linear interpolants presented in Section 4 is incomplete generally as it requires that the polynomial f in $\mathcal{C}(\{f_1,\dots,f_s\})$ produced by Algorithm 2 can be represented by the sum of three polynomials, one of which is positive,

[4] For simplicity, we do not draw $x_1 + x_2 + x_3 \neq 0$, nor $2x_1 + 3x_2 - 4x_3 \neq 0$ in the figure.

the other two polynomials are from $\mathcal{C}(\{f_1, \ldots, f_{s_1}\})$ and $\mathcal{C}(\{f_{s_1+1}, \ldots, f_s\})$ respectively. In this section, we show, under Archimedean condition, the requirement can be indeed guaranteed. Thus, our approach will become complete. In particular, we shall argue Archimedean condition is a necessary and reasonable restriction in practice.

5.1 Archimedean Condition

To the end, we need more knowledge of real algebraic geometry.

Definition 9 (quadratic module). *For $g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$, the set $\mathcal{M}(g_1, \ldots, g_m) = \{\delta_0 + \sum_{j=1}^{m} \delta_j g_j \mid \delta_0, \delta_j \in \mathcal{C}(\emptyset)\}$ is called the* quadratic module *generated by $g_1, \ldots, g_m$. A quadratic module $\mathcal{M}$ is called* proper *if $-1 \notin \mathcal{M}$ (i.e. $\mathcal{M} \neq \mathbb{R}[\mathbf{x}]$). A quadratic module $\mathcal{M}$ is* maximal *if for any $p \in \mathbb{R}[\mathbf{x}] \cap \overline{\mathcal{M}}$, $\mathcal{M} \cup \{p\}$ is not a quadratic module.*

In the sequel, we use $-\mathcal{M}$ to denote $\{-p \mid p \in \mathcal{M}\}$ for a given quadratic module $\mathcal{M}$.

The following results are adapted from [22,28] and will be used later, whose proofs can be found in [22,28].

Lemma 2 ([22,28]).

1) *If $\mathcal{M} \subseteq \mathbb{R}[\mathbf{x}]$ is a quadratic module, then $I = \mathcal{M} \cap -\mathcal{M}$ is an ideal.*
2) *If $\mathcal{M} \subseteq \mathbb{R}[\mathbf{x}]$ is a maximal proper quadratic module, then $\mathcal{M} \cup -\mathcal{M} = \mathbb{R}[\mathbf{x}]$.*
3) *$\{\mathbf{x} \in \mathbb{R}^n \mid f(\mathbf{x}) \geq 0\}$ is a compact set[5] for some $f \in \mathcal{M}(\{f_1, \ldots, f_s\})$ iff*

$$\forall p \in \mathbb{R}[\mathbf{x}], \exists n \in \mathbb{N}. n \pm p \in \mathcal{M}(f_1, \ldots, f_s). \tag{4}$$

Definition 10 (Archimedean). *For $g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$, the quadratic module $\mathcal{M}(g_1, \ldots, g_m)$ is said to be* Archimedean *if the condition (4) holds.*

$$\text{Let } \mathcal{T}_1 = f_1(\mathbf{x}) \geq 0, \ldots, f_{s_1}(\mathbf{x}) \geq 0 \text{ and } \mathcal{T}_2 = f_{s_1+1}(\mathbf{x}) \geq 0, \ldots, f_s(\mathbf{x}) \geq 0 \tag{5}$$

be two SASs, which contains constraints $c_l \leq x_i \leq c_r$ for every $x_i \in \mathbf{x}$, where c_l and c_r are reals, and $\mathcal{T}_1$ and $\mathcal{T}_2$ do not share real solutions.

Proposition 1. *Suppose $\{f_1(\mathbf{x}), \ldots, f_s(\mathbf{x})\}$ is given in (5), which contains constraints $c_l \leq x_i \leq c_r$ for every $x_i \in \mathbf{x}$. We can alway obtain a system $\{f_1(\mathbf{x}), \ldots, f_{s'}(\mathbf{x})\}$ such that $\mathcal{M}(f_1, \ldots, f_{s'})$ is Archimedean and $f_1 \geq 0 \wedge \cdots \wedge f_s \geq 0 \Leftrightarrow f_1 \geq 0 \wedge \cdots \wedge f_{s'} \geq 0$.*

Proof. For $\{f_1, \ldots, f_s\}$ in (5), as any variable is bounded, $N - \sum_{i=1}^{n} x_i^2 \geq 0$ is valid for some constant N under (5). We denote $N - \sum_{i=1}^{n} x_i^2$ by f_{s+1}. If $f_{s+1} \in \{f_1, \ldots, f_s\}$, then $\{f_1, \ldots, f_s\}$ is *Archimedean*. Otherwise, $\mathcal{M}(f_1, \ldots, f_s, f_{s+1})$ is *Archimedean*. □

Lemma 3. *[22,28] Let $\mathcal{M} \subseteq \mathbb{R}[\mathbf{x}]$ be a maximal and proper quadratic module, which is* Archimedean, *$I = \mathcal{M} \cap -\mathcal{M}$, and $f \in \mathbb{R}[\mathbf{x}]$, then there exists $a \in \mathbb{R}$ such that $f - a \in I$.*

Lemma 4. *If I is an ideal and there exists $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$ such that $x_i - a_i \in I$ for $i = 1, \ldots, n$, then for any $f \in \mathbb{R}[\mathbf{x}], f - f(\mathbf{a}) \in I$.*

[5] S is a compact set in $\mathbb{R}^n$ iff S is a bounded closed set.

Proof. Because $x_i - a_i \in I$ for $i = 1, \dots, n$, $\langle x_1 - a_1, \dots, x_n - a_n \rangle \subseteq I$. For any $f \in \mathbb{R}[\mathbf{x}]$, $\langle x_1 - a_1, \dots, x_n - a_n \rangle$ is a radical ideal[6] and $(f - f(\mathbf{a}))(\mathbf{a}) = 0$, so $f - f(\mathbf{a}) \in \langle x_1 - a_1, \dots, x_n - a_n \rangle \subseteq I$. □

Theorem 3. *Suppose* $\{f_1(\mathbf{x}), \dots, f_{s'}(\mathbf{x})\}$ *is given in Proposition (1). If* $\bigwedge_{i=1}^{s'}(f_i \geq 0)$ *is unsatisfiable i.e.* $\bigwedge_{i=1}^{s}(f_i \geq 0)$ *is unsatisfiable, then* $-1 \in \mathcal{M}(f_1, \dots, f_{s'})$.

Proof. By Proposition 1, we only need to prove that the quadratic module $\mathcal{M}(f_1, \dots, f_{s'})$ is not proper.

Assume $\mathcal{M}(f_1, \dots, f_{s'})$ is proper. By Zorn's lemma, we can extend $\mathcal{M}(f_1, \dots, f_{s'})$ to a maximal proper quadratic module $\mathcal{M} \supseteq \mathcal{M}(f_1, \dots, f_{s'})$. Since $\mathcal{M}(f_1, \dots, f_{s'})$ is Archimedean, $\mathcal{M}$ is also Archimedean. By Lemma 3, there exists $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{R}^n$ such that $x_i - a_i \in I = \mathcal{M} \cap -\mathcal{M}$ for all $i \in \{1, \dots, n\}$. From Lemma 4, $f - f(\mathbf{a}) \in I$ for any $f \in \mathbb{R}[\mathbf{x}]$. In particular, for $f = f_j$, we have $f_j(\mathbf{a}) = f_j - (f_j - f_j(\mathbf{a})) \in \mathcal{M}$, as $f_j \in \mathcal{M}(f_1, \dots, f_{s'}) \subseteq \mathcal{M}$ and $-(f_j - f_j(\mathbf{a})) \in \mathcal{M}$, for $j = 1, \dots, s'$. Suppose $f_j(\mathbf{a}) < 0$, then there exists $y \in \mathbb{R}$ such that $y^2 f_j \mathbf{a} = -1 \in \mathcal{M}$, which contradicts to the assumption, so $f_j(\mathbf{a}) \geq 0$. This contradicts to the unsatisfiability of $\bigwedge_{i=1}^{s'}(f_i \geq 0)$. □

By Theorem 3 we have $-1 \in \mathcal{M}(f_1, \dots, f_{s'})$. So, there exist $\sigma_0, \dots, \sigma_{s'} \in \mathcal{C}(\emptyset)$ such that $-1 = \sigma_0 + \sigma_1 f_1 + \cdots + \sigma_{s_1} f_{s_1} + \sigma_{s_1+1} f_{s_1+1} + \cdots + f_{s'} \sigma_{s'}$. It follows

$$-\left(\frac{1}{2} + \sigma_{s_1+1} f_{s_1+1} + \cdots + \sigma_{s'} f_{s'}\right) = \frac{1}{2} + \sigma_0 + \sigma_1 f_1 + \cdots + \sigma_{s_1} f_{s_1}. \qquad (6)$$

Let $q(\mathbf{x}) = \frac{1}{2} + \sigma_0 + \sigma_1 f_1 + \cdots + \sigma_{s_1} f_{s_1}$, we hence have $\forall \mathbf{x} \in \mathcal{T}_1 . q(\mathbf{x}) > 0$ and $\forall \mathbf{x} \in \mathcal{T}_2 . f_{s'}(\mathbf{x}) \geq 0 \wedge q(\mathbf{x}) < 0$. So, let $I = q(\mathbf{x}) > 0$. According to Definition 1, I is an interpolant of $\mathcal{T}_1$ and $\mathcal{T}_2$. So, under Archimedean condition, we can revise Algorithm 3 as Algorithm 4.

Algorithm 4. `RSN_Interpolants`

```
input : $\mathcal{T}_1$ and $T_2$ as in (5)
output: $I$
1  b=2;
2  while true do
3      sdp=Certificate_Generation($\{f_1, \dots, f_s\}$,0,{},$b$);
4      if sdp $\neq$ NULL then
5          $I = \{\frac{1}{2} + \sum_{i=1}^{s_1} p_i f_i > 0\}$;
6          return $I$;
7      else
8          b=b+2;
9      end
10 end
```

[6] Ideal I is a radical ideal if $I = \sqrt{I} = \{f \mid f^k \in I$ for some integer $k > 0\}$.

Example 3. Let $\Psi = \bigwedge_{i=1}^{3} x_i \geq -2 \wedge -x_i \geq -2$, $f_1 = -x_1^2 - 4x_2^2 - x_3^2 + 2$, $f_2 = x_1^2 - x_2^2 - x_1x_3 - 1$, $f_3 = -x_1^2 - 4x_2^2 - x_3^2 + 3x_1x_2 + 0.2$, $f_4 = -x_1^2 + x_2^2 + x_1x_3 + 1$. Consider $\mathcal{T}_1 = \Psi \wedge f_1 \geq 0 \wedge f_2 \geq 0$ and $\mathcal{T}_2 = \Psi \wedge f_3 \geq 0 \wedge f_4 \geq 0$. Obviously, $\mathcal{T}_1 \wedge \mathcal{T}_2$ is unsatisfiable, see Fig. 1 (the third part).

By applying `RSN_Interpolants`, we can get an interpolant as $-33.7255x_1^4 + 61.1309x_1^3x_2 + 4.6818x_1^3x_3 - 57.927x_1^2x_2^2 + 13.4887x_1^2x_2x_3 - 48.9983x_1^2x_3^2 - 8.144x_1^2 - 48.1049x_1x_2^3 - 6.7143x_1x_2^2x_3 + 29.8951x_1x_2x_3^2 + 61.5932x_1x_2 + 0.051659x_1x_3^3 - 0.88593x_1x_3 - 34.7211x_2^4 - 7.8128x_2^3x_3 - 71.9085x_2^2x_3^2 - 60.5361x_2^2 - 1.6845x_2x_3^3 - 0.5856x_2x_3 - 15.2929x_3^4 - 9.7563x_3^2 + 6.7326 > 0$, which is depicted in Fig 1 (the fourth part). In this example, the final value of b is 2. □

5.2 Discussions

1. Reasonability of the Archimedean Condition: Only bounded numbers can be represented in computer, so it is reasonable to constraint each variable with upper and lower bounds in practice. Not allowing strict inequalities indeed reduces the expressiveness from a theoretical point of view. However, as only numbers with finite precision can be represented in computer, we can always relax a strict inequality to an equivalent non-strict inequality in practice too. In a word, we believe the *Archimedean condition* is reasonable in practice.

2. Necessity of the Archimedean Condition: In Theorem 3, the *Archimedean condition* is necessary. For example, let $\mathcal{T}_1 = \{x_1 \geq 0, x_2 \geq 0\}$ and $\mathcal{T}_2 = \{-x_1x_2 - 1 \geq 0\}$. So, $\mathcal{T}_1 \wedge \mathcal{T}_2 = \emptyset$ is not *Archimedean* and unsatisfiable, but $-1 \notin \mathcal{M}(x_1, x_2, -x_1x_2 - 1)$ (refer to the full version [27] for the proof).

6 Correctness and Complexity Analysis

The correctness of the algorithm `SN_Interpolants` is obvious according to Theorem 2 and the discussion of Section 4. Its complexity just corresponds to one iteration of the algorithm `RSN_Interpolants`. The correctness of the algorithm `RSN_Interpolants` is guaranteed by Theorem 2 and Theorem 3. The cost of each iteration of `RSN_Interpolants` depends on the number of the variables n, the number of polynomial constraints u, and the current highest degree d. The size of X in (1) is $u\binom{n+d/2}{n}$ and the m in (1) is $\binom{n+d}{n}$. So, the complexity of applying interior point method to solve the **SDP** is polynomial in $u\binom{n+d/2}{n}\binom{n+d}{n}$. Hence, the cost of each iteration of `RSN_Interpolants` is $u\binom{n+d/2}{n}\binom{n+d}{n}$. Therefore, the total cost of `RSN_Interpolants` is polynomial in $du\binom{n+d/2}{n}\binom{n+d}{n}$ for a given upper bound of degree d. For a given problem in which n, u are fixed, the complexity of the algorithm becomes polynomial in d. As indicated in [24], the theoretical bound of d is at least triply exponential. So our method can not solve every possible instances in polynomial time. The complexity of Algorithm `SN_Interpolants` is the same as above discussions, except that the number of polynomial constraints is about $2^{s_1} + 2^{s-s_1}$.

7 Implementation and Experimental Results

We have implemented a prototypical tool of the algorithms described in this paper, called `AiSat`, which contains 6000 lines of `C++` codes. `AiSat` calls `Singular` [29] to deal with polynomial input and `CSDP` to solve **SDP**s. In `AiSat`, we design a specific algorithm to transform polynomial constraints to matrix constraints, which indeed improves the efficiency of our tool very much, indicated by the comparison with `SOSTOOLS` [30] (see the table below). As a future work, we plan to implement a new **SDP** solver with more stability and convergence efficiency for solving **SDP**s.

In the following, we report the experimental results by applying `AiSat` to some benchmarks.

The first example is from [31], see the source code in Code 1.2. We show its correctness by applying `AiSat` to the following two possible executions.

- Subproblem 1: Suppose there is an execution starting from a state satisfying the assertion at line 13 (obviously, the initial state satisfies the assertion), after $\rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 11 \rightarrow 12 \rightarrow 13$, ending at a state that does not satisfy the assertion.
  Then the interpolant synthesized by our approach is $716.77+1326.74ya+1.33ya^2+433.90ya^3+668.16xa-155.86xa\cdot ya+317.29xa\cdot ya^2+222.00xa^2+592.39xa^2\cdot ya+271.11xa^3>0$, which guarantees that this execution is infeasible.
- Subproblem 2 : Assume there is an execution starting from a state satisfying the assertion at line 13, after $\rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 13$, ending at a state that does not satisfy the assertion.
  The interpolant generated by our approach is $716.95+1330.91ya+67.78ya^2+551.51ya^3+660.66xa-255.52xa\cdot ya+199.84xa\cdot ya^2+155.63xa^2+386.87xa^2\cdot ya+212.41xa^3>0$, which guarantees this execution is infeasible either.

```
1  int main ()
2  {    int x, y;
3       int xa := 0;
4       int ya := 0;
5       while (nondet())
6       {    x := xa + 2 * ya;
7            y := -2 * xa + ya;
8            x + +;
9            if (nondet()) y = y + x;
10           else y := y - x;
11           xa := x - 2 * y;
12           ya := 2 * x + y;    }
13      assert (xa + 2 * ya >= 0);
14      return 0;    }
```

Code 1.2:ex1

```
vc := 0;
   /* the initial veclocity */
fr := 1000;
   /* the initial force */
ac := 0.0005 * fr;
   /* the initial acceleration */
while ( 1 )
{    fa := 0.5418 * vc * vc;
        /* the force control */
   fr := 1000 - fa;
   ac := 0.0005 * fr;
   vc := vc + ac;
     assert(vc < 49.61);
        /* the safety velocity */ }
```

Code 1.3: An accelerating car

The second example `accelerate` (see Code 1.3) is from [21]. Taking the air resistance into account, the relation between the car's velocity and the physical drag contains quadratic functions. Due to air resistance the velocity of the car cannot surpass

$49.61m/s$, which is a safety property. Assume that there is an execution $(vc < 49.61) \rightarrow 8 \rightarrow 10 \rightarrow 11 \rightarrow 12 \rightarrow 13(vc \geq 49.61)$. By applying `AiSat`, we can obtain an interpolant $-1.3983vc + 69.358 > 0$, which guarantees $vc < 49.61$. So, `accelerate` is correct.

The last example `logistic` is also from [21]. Mathematically, the logistic loop is written as $x_{n+1} = rx_n(1 - x_n)$, where $0 \leq x_n \leq 1$. When $r = 3.2$, the logistic loop oscillates between two values. The verification obligation is to guarantee that it is within the safe region $(0.79 \leq x \leq 0.81) \vee (0.49 \leq x \leq 0.51)$. By applying `AiSat` to the following four possible executions, the correctness is obtained.

- Subproblem 1: $\{x \geq 0.79 \wedge x \leq 0.81\}$ `logistic` $\{x > 0.51\}$ is invalidated by the synthesized interpolant $108.92 - 214.56x > 0$.
- Subproblem 2: $\{x \geq 0.79 \wedge x \leq 0.81\}$ `logistic` $\{x < 0.49\}$ is outlawed by the synthesized interpolant $-349.86 + 712.97x > 0$.
- Subproblem 3: $\{x \geq 0.49 \wedge x \leq 0.51\}$ `logistic` $\{x > 0.81\}$ is excluded by the generated interpolant $177.21 - 219.40x > 0$.
- Subproblem 4: $\{x \geq 0.49 \wedge x \leq 0.51\}$ `logistic` $\{x < 0.79\}$ is denied by the generated interpolant $-244.85 + 309.31x > 0$.

The experimental results of applying `AiSat` to the above three examples on a desktop (64-bit Intel(R) Core(TM) i5 CPU 650 @ 3.20GHz, 4GB RAM memory and Ubuntu 12.04 GNU/Linux) are listed in the table below. Meanwhile, as a comparison, we apply the SOSTOOLS to the three examples with the same computer.

Benchmark	#Subporblems	`AiSat` (milliseconds)	`SOSTOOLS` (milliseconds)
`ex1`	2	60	3229
`accelerate`	1	940	879
`logistic`	4	20	761

8 Conclusion

The main contributions of the paper include:

- We give a sound but inomplete algorithm `SN_Interpolants` for the generation of interpolants for non-linear arithmetic in general.
- If the two systems satisfy *Archimedean condition*, we provide a more practical algorithm `RSN_Interpolants`, which is not only sound but also complete, for generating Craig interpolants.
- We implement the above algorithms as a protypical tool `AiSat`, and demonstrate our approach by applying the tool to some benchmarks.

In the future, we will focus how to relax the Archimedean condition and how to combine non-linear arithmetic with other well-established decidable first order theories. In particular, we believe that we can use the method of [32,21] to extend our algorithm to uninterpreted functions. To investigate errors caused by numerical computation in **SDP** is quite interesting. In addition, it deserves to investigate the possibility to apply our results to verify hybrid systems.

References

1. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logic of Programs, vol. 131. pp. 52–71 (1981)
2. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: a proof assistant for higher-order logic. Springer, Heidelberg (2002)
3. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)
4. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL 1977, pp. 238–252 (1977)
5. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999)
6. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 154–169. Springer, Heidelberg (2000)
7. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract DPLL procedure to DPLL(T). J. ACM 53(6), 937–977
8. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. 1(2), 245–257 (1979)
9. Craig, W.: Linear reasoning: A new form of the Herbrand-Gentzen theorem. J. Symb. Log. 22(3), 250–268 (1957)
10. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
11. de Moura, L., Bjørner, N.S.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
12. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
13. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. 345(1), 101–121
14. 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)
15. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: POPL 2004, pp. 232–244 (2004)
16. McMillan, K.L.: Lazy abstraction with interpolants. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 123–136. Springer, Heidelberg (2006)
17. Jung, Y., Lee, W., Wang, B.-Y., Yi, K.: Predicate generation for learning-based quantifier-free loop invariant inference. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 205–219. Springer, Heidelberg (2011)
18. 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)
19. Kapur, D., Majumdar, R., Zarba, C.: Interpolation for data structures. In: FSE 2006, pp. 105–116 (2006)
20. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. J. Symb. Comput. 45(11), 1212–1233 (2010)
21. 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)
22. Bochnak, J., Coste, M., Roy, M.F.: Real Algebraic Geometry. Springer (1998)

23. Parrilo, P.A.: Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. PhD thesis, California Inst. of Tech. (2000)
24. Parrilo, P.A.: Semidefinite programming relaxations for semialgebraic problems. Mathematical Programming 96, 293–320 (2003)
25. Vandenberghe, L., Boyd, S.: Semidefinite programming. SIAM Review 38(1), 49–95 (1996)
26. Chen, Y., Xia, B., Yang, L., Zhan, N.: Generating polynomial invariants with DISCOVERER and QEPCAD. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) Formal Methods and Hybrid Real-Time Systems. LNCS, vol. 4700, pp. 67–82. Springer, Heidelberg (2007)
27. Dai, L., Xia, B., Zhan, N.: Generating non-linear interpolants by semidefinite programming. CoRR abs/1302.4739 (2013)
28. Laurent, M.: Sums of squares, moment matrices and optimization over polynomials. In: Emerging Applications of Algebraic Geometry. The IMA Volumes in Mathematics and its Applications, vol. 149, pp. 157–270 (2009)
29. Greuel, G.M., Pfister, G., Schönemann, H.: Singular: a computer algebra system for polynomial computations. ACM Commun. Comput. Algebra 42(3), 180–181 (2009)
30. Prajna, S., Papachristodoulou, A., Seiler, P., Parrilo, P.A.: SOSTOOLS: Sum of squares optimization toolbox for MATLAB (2004)
31. Gulavani, B., Chakraborty, S., Nori, A., Rajamani, S.: Automatically refining abstract interpretations. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 443–458. Springer, Heidelberg (2008)
32. Sofronie-Stokkermans, V.: Interpolation in local theory extensions. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 235–250. Springer, Heidelberg (2006)