Validating Numerical Semidefinite Programming Solvers for Polynomial Invariants

Pierre Roux, Yuen-Lam Voronin, Sriram Sankaranarayanan

Validating Numerical Semidefinite Programming Solvers for Polynomial Invariants.

Pierre Roux[1], Yuen-Lam Voronin[2], and Sriram Sankaranarayanan[2]

[1] ONERA – The French Aerospace Lab, Toulouse, FRANCE
[2] University of Colorado, Boulder, CO, USA

Abstract. Semidefinite programming (SDP) solvers are increasingly used as primitives in many program verification tasks to synthesize and verify polynomial invariants for a variety of systems including programs, hybrid systems and stochastic models. On one hand, they provide a tractable alternative to reasoning about semi-algebraic constraints. However, the results are often unreliable due to "numerical issues" that include a large number of reasons such as floating-point errors, ill-conditioned problems, failure of strict feasibility, and more generally, the specifics of the algorithms used to solve SDPs. These issues influence whether the final numerical results are trustworthy or not. In this paper, we briefly survey the emerging use of SDP solvers in the static analysis community. We report on the perils of using SDP solvers for common invariant synthesis tasks, characterizing the common failures that can lead to unreliable answers. Next, we demonstrate existing tools for guaranteed semidefinite programming that often prove inadequate to our needs. Finally, we present a solution for verified semidefinite programming that can be used to check the reliability of the solution output by the solver and a padding procedure that can check the presence of a feasible nearby solution to the one output by the solver. We report on some successful preliminary experiments involving our padding procedure.

1 Introduction

Program analysis techniques using abstract interpretation, especially numerical domain program analysis, rely fundamentally on the ability to reason about constraints expressed in a suitable logic that stems from the abstract domain. Typical *reasoning tasks* include the problem of checking satisfiability of an assertion in the logic used in emptiness and inclusion checks, and characterizing elements of the cone of consequences of an assertion used to compute the transfer function and join operations [16]. The process of using basic solver primitives has led to many constraint-based approaches to synthesizing and verifying invariants for programs [2,15,25,26,27,52]. Initial approaches that focused on linear systems [25,26,52] have been generalized to address nonlinear (polynomial) systems [2,4,28,42]. Other extensions to hybrid systems and stochastic systems have also been proposed [12,17,47].

However, extensions to polynomial systems necessarily face the challenge of reasoning about polynomial inequality constraints. While the problem of checking satisfiability of these constraints is well-studied, precise solutions to this problem are as yet intractable for large problems. Likewise, computing the cone of consequences precisely is also

prohibitively expensive in practice, requiring quantifier elimination. A more tractable alternative uses a convex relaxation from the given polynomial system to a *semidefinite programming problem* (SDP) using the *sum of squares* (SOS) relaxation [36,43,55]. Such a relaxation guarantees soundness when used in invariant checking/synthesis tasks, and has been shown to have nice theoretical guarantees. However, in practice, the approach requires us to use SDP solvers. It is well-known that precise solutions of SDPs is a hard and open research question. For instance, there are SDPs which have a feasible solution but no rational feasible solutions. Therefore, most solvers seek an approximate solution. At the same time, it is well known (but not well documented) that numerical SDP solvers are also hard to use in practice. The presence of "numerical issues" leads to unreliable answers from the SDP solvers, that in turn lead to unsound results when employed in program analysis tasks.

In this paper, we characterize numerical issues into many types. At one end of the spectrum, we have issues that arise from floating-point errors and approximate answers, since numerical solvers seldom reach a true optimal solution. At the other end, certain problems are not well posed, depending on the nature of the solution technique used. One common reason involves the failure of strict feasibility. Using actual examples from the literature, we show how the answers from popular numerical SDP solvers can be wrong and potentially mislead even a careful user who pays due attention to the various errors reported by the SDP solver.

Finally, we address some of the numerical problems raised. We first present a sound verification procedure that can check the answer from the solver and help us decide whether the answer is *qualitatively* correct. Next, we provide a padding procedure that helps reformulate a given problem into a stricter version so that if an approximate, floating-point solver can find a reliable answer to the stricter version, then we conclude feasibility of the original version. We integrate our framework into a polynomial invariant synthesis/verification task, showing how our ideas can successfully address numerical issues arising from the solver.

2 Motivating Examples

In this section, we illustrate through two examples the scenario where numerical SDPs give seemingly sensible solutions to simple invariant generation problems, and yet the generated invariants are not sound.

Consider the program in Fig. 1. Does there exist an inductive invariant[3] in the form $\left\{(x_1, x_2) \in \mathbb{R}^2 \;\middle|\; p(x_1, x_2) \geq 0\right\}$ for some polynomial p? A *tractable* sufficient condition that guarantees this can be formulated using the SOS optimization approach (see Section 3), resulting in an SDP instance that can be solved by numerical solvers. The widely used SDPT3 [59] solver reports a solution. Although all the DIMACS errors [53] are less than 10^{-8}, not raising any suspicion, we found traces of the program that violate this purported invariant (see Fig. 1).

As another example, we consider a program from ADJÉ et al. [1] and the "invariant" they offer, generated with numerical solvers (Fig. 2). Note that the purported invariant is

[3] In the remainder of this paper, the word "invariant" is used for inductive invariant.

```
(x1, x2) ∈ {x1, x2 | x1^2 + x2^2 ≤ 1.5^2}
while (1) {  // Find Inv. p(x1, x2) ≥ 0
  x1 = x1 * x2;
  x2 = -x1;
}
```

$$p(x_1, x_2) : \begin{pmatrix} 1 + 2.46x_1^2 + 2.46x_2^2 - 5 \times 10^{-7} x_1^4 \\ -2.46x_1^2 x_2^2 - 5 \times 10^{-7} x_2^4 \end{pmatrix}$$

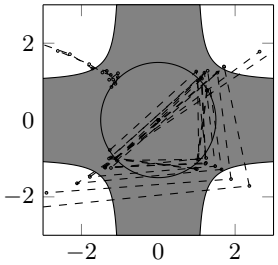

Fig. 1: (**Left**) An example program, and "loop invariant" $p(x_1, x_2) \geq 0$ synthesized using numerical solvers. (**Right**) The claimed "invariant" and dashed lines showing violations.

```
(x1, x2) ∈ [0.9, 1.1] × [0, 0.2]
while (1) {
  pre_x1 = x1; pre_x2 = x2;
  if (x1^2 + x2^2 <= 1) {
    x1 = pre_x1^2 + pre_x2^3;
    x2 = pre_x1^3 + pre_x2^2;
  } else {
    x1 = 0.5 * pre_x1^3
             + 0.4 * pre_x2^2;
    x2 = -0.6 * pre_x1^2
             + 0.3 * pre_x2^2;
  }
}
```

$$\begin{aligned} &2.510902467 + 0.0050x_1 + 0.0148x_2 - 3.0998x_1^2 \\ &+ 0.8037x_2^3 + 3.0297x_1^3 - 2.5924x_2^2 \\ &- 1.5266x_1x_2 + 1.9133x_1^2x_2 + 1.8122x_1x_2^2 - 1.6042x_1^4 \\ &- 0.0512x_1^3x_2 + 4.4430x_1^2x_2^2 + 1.8926x_1x_2^3 - 0.5464x_2^4 \\ &+ 0.2084x_1^5 - 0.5866x_1^4x_2 - 2.2410x_1^3x_2^2 - 1.5714x_1^2x_2^3 \\ &+ 0.0890x_1x_2^4 + 0.9656x_2^5 - 0.0098x_1^6 + 0.0320x_1^5x_2 \\ &+ 0.0232x_1^4x_2^2 - 0.2660x_1^3x_2^3 - 0.7746x_1^2x_2^4 \\ &- 0.9200x_1x_2^5 - 0.6411x_2^6 \geq 0 \end{aligned}$$

Fig. 2: (**Left**) An example program taken from from ADJÉ et al. [1] (Example 4). (**Right**) Purported invariant at loop head synthesized using SDP solvers [1].

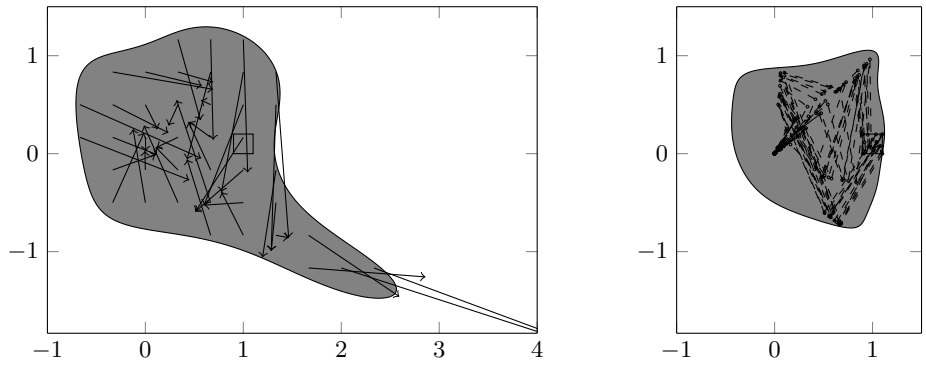

Fig. 3: (**Left**) The candidate invariant from Fig. 2 with arrows showing concrete transitions. The arrows leaving it are counterexamples to its inductiveness. (**Right**) The invariant of degree 8 whose soundness is proved using the approach in this paper.

indeed not inductive: one can find points in it whose image after one iteration of the loop body exits the invariant (Fig. 3). Fig. 3 also depicts an actual invariant, proved using the method in this paper.

3 Sum of Squares (SOS) and Semidefinite Programs (SDP)

In this section, we provide the background for sum of squares (SOS) optimization from the perspective of proving entailments for program analysis (invariant synthesis/verification) tasks. We then trace the steps along which SOS optimization problems are relaxed to semidefinite programs (SDP).

Let $\mathbb{R}$ denote the set of real numbers and $\mathbf{x} : (x_1, \ldots, x_n)$ denote a vector of real-valued variables. The ring of multivariate real polynomials over $\mathbf{x}$ is denoted by $\mathbb{R}[\mathbf{x}]$. The degree of any polynomial $p(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ is denoted by $\deg(p)$. A *template polynomial* is of the form $p(\mathbf{c}, \mathbf{x}) : \sum_{j=1}^{s} c_j\, p_j(\mathbf{x})$, where $p_1, \ldots, p_s$ are *basis polynomials* and $\mathbf{c} : (c_1, \ldots, c_s)$ is a placeholder for parameters serving as scalar multiples of the basis polynomials. A *generic template polynomial* of degree $d > 0$ is formed by choosing all monomials of degree up to d as the basis polynomials, and has $s = \binom{n+d}{d}$ parameters.

3.1 Semi-algebraic Assertions and Entailment Problems

A *semi-algebraic* assertion φ is a finite conjunction of polynomial inequalities:

$$\varphi : \; p_1(\mathbf{x}) \geq 0 \; \wedge \; \cdots \; \wedge \; p_m(\mathbf{x}) \geq 0.$$

It denotes a corresponding semi-algebraic set $[\![\varphi]\!] : \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \models \varphi\}$. As such, semi-algebraic assertions subsume useful abstract domains such as polyhedra and ellipsoids. They also represent a rich class of constraints with a decidable entailment checking problem [7]. We define two classes of problems involving semi-algebraic sets that are commonly used as primitives.

Definition 1 (Entailment Checking). *Given two semi-algebraic assertions φ and ψ over $\mathbf{x}$, check if $\varphi \models \psi$, i.e., for all $\mathbf{x} \in \mathbb{R}^n$, if $\mathbf{x} \models \varphi$, then $\mathbf{x} \models \psi$.*

Definition 2 (Parametric Entailment). *Let $\mathbf{c} : (c_1, \ldots, c_s)$ represent parameters. The input to a* parametric entailment problem *consists of k pairs $(\varphi_i, p_i)_{i=1}^{k}$, wherein φ_i is a semi-algebraic assertion and $p_i(\mathbf{c}, \mathbf{x})$ is a template polynomial. The goal is to compute a value $\mathbf{c}$ such that all the entailments hold:*

$$(\varphi_1 \; \models \; p_1(\mathbf{c}, \mathbf{x}) \geq 0) \; \wedge \; \cdots \; \wedge \; (\varphi_k \; \models \; p_k(\mathbf{c}, \mathbf{x}) \geq 0)\,.$$

The entailment checking and its analog of parametric entailment checking are fundamental primitives that we will use for synthesizing and checking invariants of programs. The example below illustrates the application of these primitives.

```
(x1, x2) ∈ I : {(x1, x2) | x1^2 ≤ 1 ∧ x2^2 ≤ 1}
while (1) {
  pre_x1 = x1; pre_x2 = x2;
  if (x1 >= x2) {
    x1 = 0.687 * pre_x1 + 0.558 * pre_x2
         - 0.0001 * pre_x1 * pre_x2;
    x2 = -0.292 * pre_x1 + 0.773 * pre_x2;
  } else {
    x1 = 0.369 * pre_x1 + 0.532 * pre_x2
         - 0.0001 * pre_x1^2;
    x2 = -1.27 * pre_x1 + 0.12 * pre_x2
         - 0.0001 * prex_x1 * pre_x2;
  }
}
```

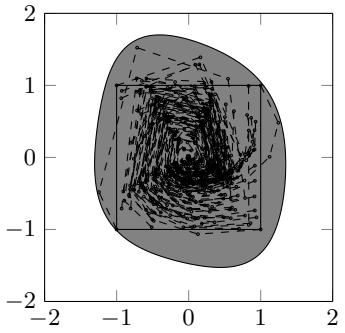

Fig. 4: (**Left**) An example program. (**Right**) Invariant $[\![p(x_1, x_2) \geq 0]\!]$, along with executions that start inside the initial set I (square).

Invariant checking: Consider the program in Fig. 4. We wish to prove that all executions remain inside a safe set $S : \{(x_1, x_2) \mid |x_1| \leq 2 \ \wedge \ |x_2| \leq 2\}$. To prove this, we consider an inductive invariant $\{(x_1, x_2) \mid p(x_1, x_2) \geq 0\}$, where

$$p(x_1, x_2) : 37 - x_2^2 + x_1^3 - 2x_1^2x_2 + 2x_2^3 - 12x_1^4 - 10x_1^2x_2^2 - 6x_1x_2^3 - 6x_2^4. \quad (1)$$

Fig. 4 shows the invariant region $p(x_1, x_2) \geq 0$. To show that it is indeed an invariant that establishes S as a safe set, we check that the following conditions hold:

(a) *Initial condition:* $1 - x_1^2 \geq 0 \ \wedge \ 1 - x_2^2 \geq 0 \ \models \ p(x_1, x_2) \geq 0$,
(b) *Consecution (loop) conditions:* Let $\tau_1(x_1, x_2) : (0.687x_1 + 0.558x_2 - 0.0001x_1x_2, \ -0.292x_1 + 0.773x_2)$ denote the transition enabled by the condition $x_1 \geq x_2$, and $\tau_2(x_1, x_2) : (0.369x_1 + 0.532x_2 - 0.0001x_1^2, \ -1.27x_1 + 0.12x_2 - 0.0001x_1x_2)$ denote the transition enabled by the condition $x_1 < x_2$. We require two conditions, corresponding to the two transitions in the loop:
   (i) $x_1 - x_2 \geq 0 \ \wedge \ p(x_1, x_2) \geq 0 \ \models \ \hat{p_1} \geq 0$, where $\hat{p_1} = p \circ \tau_1$, and
   (ii) $x_2 - x_1 \geq 0 \ \wedge \ p(x_1, x_2) \geq 0 \ \models \ \hat{p_2} \geq 0$, where $\hat{p_2} = p \circ \tau_2$.
(c) *Safety conditions:* $p(x_1, x_2) \geq 0 \ \models \ -2 \leq x_1 \leq 2 \ \wedge \ -2 \leq x_2 \leq 2$.

The invariant checking problem is then a series of polynomial entailment checking.

Invariant synthesis: The invariant synthesis problem requires us to synthesize polynomials that satisfy some entailment conditions, such as $p(x_1, x_2)$ in (1) satisfying the initial, loop and safety conditions. To do so, we parameterize a *template* polynomial as follows:

$$p(\mathbf{c}, \mathbf{x}) : \begin{pmatrix} c_1 + c_2x_1 + c_3x_2 + c_4x_1^2 + c_5x_1x_2 + c_6x_2^2 + c_7x_1^3 + c_8x_1x_2^2 + \\ c_9x_1^2x_2 + c_{10}x_2^3 + c_{11}x_1^4 + c_{12}x_1^3x_2 + c_{13}x_1^2x_2^2 + c_{14}x_1x_2^3 + c_{15}x_2^4 \end{pmatrix}. \quad (2)$$

We then search for values of $\mathbf{c} = (c_1, \ldots, c_{15})$ such that the following entailments hold:

(a) *Initial condition:* $1 - x_1^2 \geq 0 \ \wedge \ 1 - x_2^2 \geq 0 \ \models \ p(\mathbf{c}, \mathbf{x}) \geq 0$,

(b) *Loop conditions:* The loop condition for the transition τ_1 is $(x_1 - x_2 \geq 0 \wedge p(\mathbf{c}, \mathbf{x}) \geq 0 \models \hat{p_1}(\mathbf{c}, \mathbf{x}) \geq 0)$, where $\hat{p_1} = p \circ \tau_1$. However, we note that in this condition, a parametric polynomial inequality appears on the antecedent side. This leads to hard *bilinear optimization* problems that are beyond the scope of this paper (see [51] for a discussion of this issue). We impose a stronger condition on p that states that p must be non-decreasing for each loop iteration[4]:
   (i) $x_1 - x_2 \geq 0 \models \hat{p_1}(\mathbf{c}, \mathbf{x}) \geq p(\mathbf{c}, \mathbf{x})$, and
   (ii) $x_2 - x_1 \geq 0 \models \hat{p_2}(\mathbf{c}, \mathbf{x}) \geq p(\mathbf{c}, \mathbf{x})$, where $\hat{p_2} = p \circ \tau_2$.

The invariant synthesis problem is thus reduced to a *parametric* entailment problem.

3.2 Solving Entailment Problems

There are numerous approaches to solving semi-algebraic entailment checking and parametric entailment problems. We classify these into four broad classes: (a) quantifier elimination for the theory of polynomial inequalities, (b) interval arithmetic with branch-and-bound, (c) linear programming relaxations, and (d) sum of squares relaxations that will be the focus of our exposition.

Exact Approaches: It is well-known that the logical theory of polynomial inequalities admits effective decision procedures and a quantifier elimination procedure, originally discovered by TARSKI and further developed by COLLINS, HONG, WEISPFENNING and others [7,13,14,57,61]. These procedures attempt to solve the entailment problem $\varphi \models \psi$ by checking the unsatisfiability of the assertion $\varphi(\mathbf{x}) \ \wedge \ (\neg\psi(\mathbf{x}))$. This problem is known to be NP-hard in theory, and hard to solve, in practice. Typical sizes of problems that can be tackled involve polynomials with ~ 5 variables, and degrees ~ 3 [19].

Likewise, an exact approach for the parametric entailment problem requires to perform a quantifier elimination of the form:

$$(\forall\, \mathbf{x}) \ \left(\varphi_1(\mathbf{x}) \ \Rightarrow \ p_1(\mathbf{c}, \mathbf{x}) \geq 0 \ \wedge \ \cdots \wedge \ \varphi_k(\mathbf{x}) \ \Rightarrow \ p_k(\mathbf{c}, \mathbf{x}) \geq 0 \right) .$$

Doing so leads to an assertion that can be expressed purely in terms of $\mathbf{c}$. If this assertion is satisfiable, a solution $\mathbf{c} = \mathbf{c}^*$ can be extracted.

Branch-and-Bound (BnB) Approaches: They work over states $\mathbf{x}$ that are a priori restricted to a compact set X. They proceed by subdividing X into finitely many interval (hyper-rectangular) cells. Inside each interval, the entailment is evaluated using interval arithmetic [23,24] or a branch-and-bound scheme using linear programming relaxation of the constraints [8]. BnB approaches can be used to check whether an entailment $\varphi \models \psi$ holds by checking the unsatisfiability of the assertion $\varphi(\mathbf{x}) \ \wedge \ (\neg\psi(\mathbf{x}))$. They can conclude soundly that the entailment holds or even find a *witness* $\mathbf{x}$ such that $\varphi(\mathbf{x}) \ \wedge \ (\neg\psi(\mathbf{x}))$ is satisfied. Unfortunately, due to computational limitations, these techniques may also terminate without an answer. Recent work on delta-satisfiability procedures have carefully analyzed this condition to conclude that a "nearby" formula is

[4] Control theorists call (opposite of) such functions *Lyapunov functions* [22].

satisfiable [24]. BnB approaches can extend beyond polynomial programs and invariants. Currently, BnB approaches are restricted to solving entailment problems. Their application to solving parametric entailment problems remains an open challenge. Part of the challenge involves the optimal subdivision of X to search for solutions $\mathbf{c}$.

Linear Programming (LP) Relaxations: Linear programming relaxations have been considered for checking polynomial entailment and solving parametric entailment problems (see BEN SASSI et al. for details and further references [8]). LP approaches are primarily based on so-called *Handelman relaxation* and *reformulation linearization*. Given an entailment problem with $p_1, \ldots, p_k$ as antecedents. We generate "valid inequalities" that are consequences of the original antecedents. This is achieved by simply multiplying the antecedents together, enriching the set of possible antecedents. This step is inspired by the Handelman positivstellensatz [29]. Next, we introduce fresh variables corresponding to each monomial term and turn our polynomial entailment problem into a *linear entailment* problem that can be checked using solvers. This step is called *reformulation linearization technique* (RLT) [54]. BEN SASSI et al. show that the generation of linear constraints can be performed in the Bernstein polynomial basis [9,21], rather than the monomial basis to obtain a larger set of valid inequalities. The LP approach has the main advantage that Simplex solvers can be used with exact arithmetic to completely avoid numerical issues. The recent work of MARÉCHAL et al. use this approach and generate machine checkable proofs of polynomial entailments [38]. However, LP relaxations yield a "weak" proof system that requires higher degree terms or a BnB decomposition of the domain, to prove "simple consequences" such as $-1 \leq x \leq 1 \wedge -1 \leq y \leq 1 \models (x^2+y^2 \geq 0)$ [8]. Interestingly, the Handelman relaxation and RLT are implicit in the polyhedral abstract domain for computing semi-algebraic invariants proposed by BAGNARA et al. [6]. A related approach of *diagonal SOS* (DSOS) has been proposed by ALI AHMADI and MAJUMDAR [3]. Their approach is based on the SOS relaxation wherein instead of reducing to a SDP, they reduce to an LP by imposing the stronger condition of *diagonal dominance* on the associated matrix rather than the semi-definiteness condition that will be described subsequently in this section. A full comparison of DSOS with SOS relaxations for program analysis problems is currently open.

Positivstellensatz/Sum of Squares (SOS) Relaxations: The SOS relaxation [36,43] is an incomplete but efficient way to numerically solve polynomial entailment problems.

Definition 3 (SOS Polynomial). *A polynomial $p \in \mathbb{R}[\mathbf{x}]$ is said to be* SOS *if there exist polynomials $h_i \in \mathbb{R}[\mathbf{x}]$ such that for all $\mathbf{x}$, $p(\mathbf{x}) = \sum_i h_i^2(\mathbf{x})$.*

Although not all nonnegative polynomials are SOS, being SOS is a sufficient condition to be nonnegative.

Example 1. Consider $p : 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$. Since $p = h_1^2 + h_2^2$, where $h_1 : \frac{1}{\sqrt{2}}\left(2x_1^2 + x_1x_2 - 3x_2^2\right)$ and $h_2 : \frac{1}{\sqrt{2}}\left(3x_1x_2 + x_2^2\right)$, the polynomial p is nonnegative, i.e., $p(x_1, x_2) \geq 0$ holds for all $x_1, x_2 \in \mathbb{R}^2$.

Now consider a polynomial entailment problem of the form:

$$\underbrace{p_1(\mathbf{x}) \geq 0 \ \wedge \ \cdots \ \wedge \ p_k(\mathbf{x}) \geq 0}_{\varphi} \ \models \ p(\mathbf{x}) \geq 0. \tag{3}$$

Our goal is to write p as a *combination* of $p_1, \ldots, p_k$ in the following form:

$$p = \sigma_0 + \sigma_1 \, p_1 + \cdots + \sigma_k \, p_k \tag{4}$$

such that $\sigma_0, \ldots, \sigma_k$ are SOS polynomials over $\mathbf{x}$. Let K denote the semi-algebraic set $\left[\!\left[\bigwedge_{j=1}^{k} p_j \geq 0\right]\!\right]$, let $R(K)$ denote the cone of consequences of K, i.e., $R(K) = \{p(\mathbf{x}) \mid \varphi \models p(\mathbf{x}) \geq 0\}$ and $M(K)$ denote all polynomials expressible in the form (4):

$$M(K) = \left\{ p(\mathbf{x}) \;\middle|\; p = \sigma_0 + \textstyle\sum_{j=1}^{k} \sigma_j \, p_j, \ \sigma_i \text{ SOS} \right\}.$$

Theorem 1 (Putinar's Positivstellensatz). *For all K, $M(K) \subseteq R(K)$.*
Conversely, if K is compact and $M(K)$ contains a polynomial of the form $s(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 - L$ for some constant $L > 0$, then $M(K) = R(K)$.

Proof. We will prove the "easy" direction that $M(K) \subseteq R(K)$. Let $p \in M(K)$. There exist SOS polynomials $\sigma_0, \ldots, \sigma_k$ such that $p = \sigma_0 + \sum_{i=1}^{k} \sigma_i p_i$. Let $\mathbf{x}$ be such that $p_i(\mathbf{x}) \geq 0$ for all $i \in [1, k]$. We have that $\sigma_i(\mathbf{x}) \geq 0$ since each σ_i is a SOS polynomial. Therefore, we conclude that $p(\mathbf{x}) = \sigma_0(\mathbf{x}) + \sum_{i=1}^{k} \sigma_i(\mathbf{x}) p_i(\mathbf{x}) \geq 0$.
For the converse, we refer the reader to Putinar's work [48]. □

Thus, a polynomial entailment problem of the form (3) is relaxed to an SOS problem:

$$\begin{aligned} \mathsf{find}: \ & \text{polynomials } \sigma_0, \ldots, \sigma_k \in \mathbb{R}_d[\mathbf{x}] \\ \mathsf{s.t.} \ \ & p = \sigma_0 + \textstyle\sum_{i=1}^{k} \sigma_i p_i, \\ & \sigma_0, \ldots, \sigma_k \text{ are SOS.} \end{aligned} \tag{5}$$

First, we choose a degree limit $d > 0$ (d must be an even number because all positive polynomials have even maximum degree), and select *templates* $\sigma_0(\mathbf{c}^{(0)}, \mathbf{x}), \ldots, \sigma_k(\mathbf{c}^{(k)}, \mathbf{x})$ with unknowns $\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)}$. We then require that p be equal to a polynomial combination of $p_1, \ldots, p_k$ with "multipliers" $\sigma_0, \ldots, \sigma_k$ as shown above. This yields a set of linear equations involving $\mathbf{c}^{(0)}, \ldots, \mathbf{c}^{(k)}$ and the coefficients of p, obtained by comparing both sides monomial by monomial and setting their coefficients to be the same. Finally, we require $\sigma_0, \ldots, \sigma_k$ to be SOS. This will be tackled through a reduction to a *semidefinite programming* (SDP) problem, as will be explained subsequently.

Example 2. Consider the initial condition check for the program in Fig. 4: $p_1(x_1, x_2) \geq 0 \ \wedge \ p_2(x_1, x_2) \geq 0 \ \models \ p(x_1, x_2) \geq 0$ with $p_i(x_1, x_2) = 1 - x_i^2$ and p given in (1). Our goal here is to find polynomials $\sigma_0, \sigma_1, \sigma_2$ such that $p = \sigma_0 + \sigma_1 \, p_1 + \sigma_2 \, p_2$. For simplicity, let us write $\sigma_0 = c_1 + c_2 x_1 + \cdots + c_{15} x_2^4$, $\sigma_1 = d_1 + \cdots + d_{15} x_2^4$ and

$\sigma_2 = e_1 + \cdots + e_{15}x_2^4$. We obtain equality constraints by equating terms corresponding to the same monomial on both sides:

$$c_1 + d_1 + e_1 = 37 \text{ (comparing constant terms)}, \ldots, -e_{15} = 0 \text{ (comparing } x_2^6\text{)}.$$

The SOS problem seeks to satisfy these equalities, and additionally make $\sigma_0, \sigma_1, \sigma_2$ SOS. Solving this as an SDP problem (as will be explained below), we obtain: $\sigma_1 \approx 11 - 0.13x_1 + 1.5x_2 + 24x_1^2 - 3x_1x_2 + 8.2x_2^2$ and $\sigma_2 \approx 8.8 + 0.63x_1 - 1.4x_2 + 6.5x_1^2 + 1.6x_1x_2 + 18x_2^2$.

SOS formulation of parametric entailment problems. Consider now a parametric entailment problem of the form $(\varphi_j \models p_j(\mathbf{c}, \mathbf{x}) \geq 0)$ for $j = 1, \ldots, K$ involving parameters $\mathbf{c}$. Let us write $\varphi_j : \ p_{j_1}(\mathbf{x}) \geq 0 \ \wedge \ \cdots \ \wedge \ p_{j_l}(\mathbf{x}) \geq 0$. This is reduced to a *sum of squares* problem:

$$\begin{array}{ll} \text{find}: & \text{polynomials } \sigma_{j,0}, \ldots, \sigma_{j,j_l} \in \mathbb{R}_d[\mathbf{x}], j \in \{1, \ldots, K\} \\ \text{s.t.} & p_j = \sigma_{j,0} + \sum_{i=1}^{k} \sigma_{j,i} p_{j_i}, \ j \in \{1, \ldots, K\}, \\ & \sigma_{j,0}, \ldots, \sigma_{j,j_l} \text{ are SOS}, j \in \{1, \ldots, K\}. \end{array} \tag{6}$$

The unknowns include the coefficients $\mathbf{c}$ involved in each $p_j(\mathbf{c}, \mathbf{x})$ for the original parametric entailment and the coefficients $\mathbf{c}^{(j,i)}$ corresponding to SOS multipliers $\sigma_{j,i}$.

Next, we provide a reduction from SOS problems to a well known class of optimization problems: *semidefinite programs* (SDPs). Any polynomial p of degree $2d$ (a nonnegative polynomial is necessarily of even degree) can be written as a quadratic form in the vector z of all monomials of degree less or equal d:

$$p(\mathbf{x}) = z^T Q\, z, \tag{7}$$

where $z = \left[1, x_1, \ldots, x_n, x_1x_2, \ldots, x_n^d\right]^T$ and Q is a constant symmetric matrix.

Example 3. Consider $p(x_1, x_2) : 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$. To satisfy the equality

$$\begin{aligned} p(x_1, x_2) &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix} \\ &= q_{11}x_1^4 + 2q_{13}x_1^3x_2 + (q_{33} + 2q_{12})x_1^2x_2^2 + 2q_{23}x_1x_2^3 + q_{22}x_2^4, \end{aligned}$$

the equalities $q_{11} = 2$, $2q_{13} = 2$, $q_{33} + 2q_{12} = -1$, $2q_{23} = 0$ and $q_{22} = 5$ must hold. Two possible examples for the matrix Q are shown below:

$$Q = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 5 & 0 \\ 1 & 0 & -3 \end{bmatrix}, \qquad Q' = \begin{bmatrix} 2 & -3 & 1 \\ -3 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix}.$$

The polynomial p is then SOS if and only if there exists a positive semidefinite matrix Q satisfying (7). A matrix Q is said to be *positive semidefinite*, denoted by $Q \succeq 0$, when for all vectors y, $y^T Q\, y \geq 0$. A matrix Q is said to be *positive definite*, denoted by $Q \succ 0$, when for all nonzero vectors y, $y^T Q\, y > 0$.

Example 4. In Example 3, the first matrix Q is not positive semidefinite (for $y : [0, 0, 1]^T$, $y^T Q\, y = -3$). However, the second matrix Q' is positive semidefinite as it can be written $Q' = L^T L$ with

$$L = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & -3 & 1 \\ 0 & 1 & 3 \end{bmatrix}$$

(then, for all y, $y^T Q y = (Ly)^T (Ly) = \|Ly\|_2^2 \geq 0$). This gives the SOS decomposition of Example 1: $p(x_1, x_2) = \frac{1}{2}(2x_1^2 + x_1 x_2 - 3x_2^2)^2 + \frac{1}{2}(3x_1 x_2 + x_2^2)^2$.

As a result, SOS programming problems can be written as semidefinite optimization problems involving matrices. Let z be a vector of monomials over $\mathbf{x}$ chosen so that we may write each polynomial $\sigma_i(\mathbf{c}^{(i)}, \mathbf{x})$ as a quadratic form $\sigma_i(\mathbf{c}^{(i)}, \mathbf{x}) = z^T C_i\, z$. Thus, the SOS programming problems (5) and (6) can be written down as an SDP problem:

$$\begin{array}{ll} \mathsf{find}: & c, C_0, \ldots, C_k \\ \mathsf{s.t.} & a_i^T c + \sum_{j=0}^{k} \mathrm{tr}(A_{i,j} C_j) = b_i,\ i = 1, \ldots, m, \\ & C_j \succeq 0,\ j = 0, \ldots, k. \end{array} \tag{8}$$

wherein the vector c encodes the parameters $\mathbf{c}^{(i)}$ of (5) (or $\mathbf{c}$ and $\mathbf{c}^{(j,i)}$ of (6)) and the $A_{i,j}$ and C_j are symmetric matrices. Note that the expression $\mathrm{tr}(XY)$ equals $\sum_{i=1}^{n} \sum_{j=1}^{n} (X)_{i,j} (Y)_{i,j}$ for $n \times n$ matrices X, Y when $X^T = X$.

Example 5. We check whether the entailment $(p_1(x, y) : x - y \geq 0 \models p(x, y) : x - y + 2x^2 - 2y^2 + x^3 + x^2 y - xy^2 - y^3 \geq 0)$ is true using an SOS relaxation: we look for degree 2 SOS polynomials $\sigma_0, \sigma_1 \in \mathbb{R}[x, y]$ such that $p = \sigma_0 + \sigma_1\, p_1$. In other words, we seek coefficients $\mathbf{c}^{(0)} : (c_1, \ldots, c_6)$ and $\mathbf{c}^{(1)} : (c_7, \ldots, c_{12})$ such that $\sigma_0(x, y) : c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 xy + c_6 y^2$ and $\sigma_1(x, y) : c_7 + c_8 x + c_9 y + c_{10} x^2 + c_{11} xy + c_{12} y^2$ are SOS and the coefficients of p and $\sigma_0 + \sigma_1\, p_1$ coincide, i.e., with $z : [1, x, y]^T$,

$$\begin{array}{l} \text{comparing coeffs of } \sigma_0 + \sigma_1\, p_1 \text{ and } p \\ \quad : \begin{cases} \text{const. term} & c_1 = 0, \\ \text{coeff. of } x & c_2 + c_7 = 1, \\ \text{coeff. of } y & c_3 - c_7 = -1, \\ \text{coeff. of } x^2 & c_4 + c_8 = 2, \\ \text{coeff. of } xy & c_5 - c_8 + c_9 = 0, \\ \text{coeff. of } y^2 & c_6 - c_9 = -2, \\ \text{coeff. of } x^3 & c_{10} = 1, \\ \text{coeff. of } x^2 y & c_{11} - c_{10} = 1, \\ \text{coeff. of } xy^2 & c_{12} - c_{11} = -1, \\ \text{coeff. of } y^3 & -c_{12} = -1, \end{cases} \end{array} \quad \begin{array}{l} \text{comparing coefficients of } \sigma_0 \text{ and } z^T C_0\, z \\ \quad : \begin{cases} c_1 = (C_0)_{1,1}, \\ c_2 = (C_0)_{1,2} + (C_0)_{2,1}, \\ c_3 = (C_0)_{1,3} + (C_0)_{3,1}, \\ c_4 = (C_0)_{2,2}, \\ c_5 = (C_0)_{2,3} + (C_0)_{3,2}, \\ c_6 = (C_0)_{3,3}, \end{cases} \\ \text{same for } \sigma_1 \text{ and } z^T C_1\, z. \end{array} \tag{9}$$

Each of the $m = 22$ equality constraints in (9) is then encoded as in (8). For instance, the second constraint on σ_0 is encoded by the vector $a_{12} = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$ and the matrices $A_{12,0} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $A_{12,1} = 0$.

Thus, we have eliminated the formal variables $\mathbf{x}$ from the problem and reduced it to finding matrices that satisfy some linear equality constraints, and are positive semidefinite. In fact, moving one step further, we write a single unknown matrix C in the block diagonal form: $C = \mathrm{Diag}(c_1^+, c_1^-, \ldots, c_s^+, c_s^-, C_0, C_1, \ldots, C_k)$, encoding c_i as $c_i^+ - c_i^-$ with $c_i^+, c_i^- \in \mathbb{R}_+$. This allows us to write (8) as:

$$\begin{array}{ll} \mathsf{find}: C & \mathsf{s.t.}\ \ \mathrm{tr}(A_i C) = b_i,\ i = 1, \ldots, m, \\ & C \succeq 0. \end{array} \tag{10}$$

Problems that follow this form, or equivalently (8), are called *semidefinite programming problems* (SDP). They form a well known class of convex optimization problems that generalize linear programs and can be solved numerically even for large[5] problem matrices C. Numerical solvers also allow to optimize a linear objective function of the coefficients of C. Finally, we define the notion of strict feasibility.

Definition 4 (Strict Feasibility). *The SDP in* (10) *is said to be* strictly feasible *when there exists a solution to the problem wherein the matrix C is positive definite.*

If every feasible solution C to the problem (10) is positive semidefinite but not positive definite (in other words, the matrix has zero eigenvalues, or alternatively is rank deficient), the problem is said to *fail strictly feasibility*.

Remark 1. For a strictly feasible problem, there exist solutions C such that any $\widetilde{C}$ in a neighborhood from C and satisfying the equality constraints $\mathrm{tr}(A_i\widetilde{C}) = b_i$ is also a solution. In contrary, problems that are not strictly feasible are also said to have an *empty (relative) interior* because, for any solution C, there exist $\widetilde{C}$ arbitrarily close from C that satisfy the equality constraints but are not solutions. This is illustrated on Fig 5.

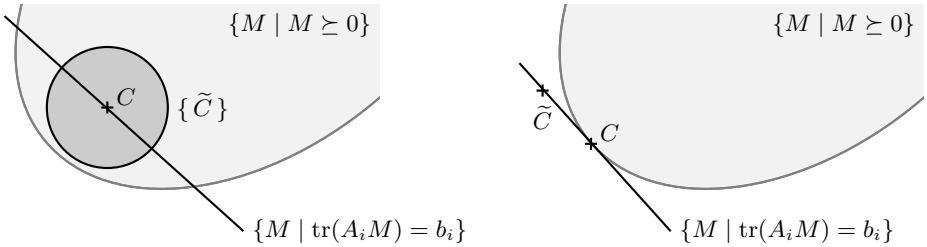

Fig. 5: The line represents the equality constraints $\mathrm{tr}(A_i C) = b_i$ and the shaded area the matrices $C \succeq 0$. The set of solutions is the intersection of the line and the shaded area. (**Left**) A strictly feasible SDP problem. (**Right**) An empty interior problem.

4 Verified SDPs

In Section 3, we laid out a procedure for formulating invariant checking and synthesis as a general SOS feasibility problem, which in turn is an SDP feasibility problem. There are SDPs with rational problem data whose solutions are irrational [58]. However, under some regularity conditions, SDP problems can theoretically be solved *efficiently* up to *arbitrarily small* error tolerance (see e.g., [62, Ch. 8-10]). In practice, many numerical solvers are available to solve SDP instances satisfactorily (see [5, Part III]). Currently, the default choice for solving SDPs are specialized second order methods (i.e., using second order derivatives) called *interior point methods* (IPMs)[6].

[5] Typically, matrices C_j can be of size $n \times n$ for n up to a few hundreds.

[6] There also exist first order methods handling larger problems but with less accurate solutions.

We discuss in Section 4.1 issues leading to inaccuracy or poor solution quality in SDP solving via IPMs. Then in Section 4.2, we consider solutions to guarantee soundness.

4.1 Sources of Solution Inaccuracy in Solving SDPs

Before we discuss ways to ensure soundness of solutions to the invariant checking and synthesis problem generated by SDP solvers, we first focus on a few issues that could possibly make an SDP solution inaccurate, leading to potential unsoundness. We will concentrate on SDP solutions obtained from general IPMs.

How do IPMs Work? The convergence of a general IPM *assumes strict feasibility* (Def. 4). Using positive definite matrices as initial points, a general IPM repeatedly solves a perturbed linearization of the Karush-Kuhn-Tucker optimality conditions for a search direction, and moves along that search direction with a fractional step size that maintains the positive definiteness of the iterates. (See e.g. [58,62].)

In the following, we discuss the four potential issues that can cause solution inaccuracy when a general IPM is used for obtaining SDP solutions: (1) inexact termination, (2) failure of strict feasibility, (3) ill-conditioning and (4) floating-point errors.

Inexact termination. The first source of inaccuracy stems from the fact that IPMs usually do not converge in finitely many iterations. Iterations are then stopped when some *stopping criterion* is met, for instance when the equalities in (10) are ϵ-approximated (i.e., $|\mathrm{tr}(A_i C) - b_i| \leq \epsilon$) or when the number of iterations becomes too big. Thus IPMs only produce approximate solutions. Nonetheless, under strict feasibility assumption, most common IPMs enjoy a convergence result of the following form: for any $\varepsilon \in (0, 1)$, if an appropriate initial point is chosen, then it takes at most a number of steps polynomial in the problem size and $\log(\frac{1}{\varepsilon})$ to obtain an ε-approximate solution. (See e.g. [62, Ch. 10].)

Failure of Strict Feasibility. Strictly feasibility is a desirable property. For instance, as seen above, it guarantees that a SDP can be solved to arbitrary accuracy by an IPM. "Random" SDP problems are strictly feasible with probability one [20, Theorem 3.2]. However, strict feasibility can fail *systematically* for SDP instances arising from applications due to the inherent problem structure. In particular, strict feasibility can fail for entailment problems (Sect. 3.1), as shown in the following example.

Example 6. The SDP feasibility problem in Example 5 fails the strict feasibility. Indeed, for any solution (c, C_0, C_1), the equality constraints imply $(C_0)_{1,1} = 0$, hence[7] $(C_0)_{1,2} = (C_0)_{2,1} = (C_0)_{1,3} = (C_0)_{3,1} = 0$ which means that C_0 is rank deficient.

While the failure of the strict feasibility in small instances such as Example 6 usually does not cause much numerical issues, significant inaccuracy can often be observed as the number of variables and the degrees of the polynomials increase [60]. Facial reduction techniques proposed by Borwein and Wolkowicz can be used for preprocessing SDP instances that are not strictly feasible [11]. A more efficient version using linear programming reduction, called partial facial reduction, was proposed by Permenter and Parrilo [44].

[7] If a matrix is PSD and one of its diagonal entry (e.g. the $(1, 1)$ entry) equals 0, then the entire row and column that contain that diagonal entry (e.g., the first row and column) equal 0.

Ill Conditioning. The coefficients of the polynomials in the entailment problems can influence the condition number of the linear system that is solved in IPMs, and a large condition number can affect the convergence of IPMs. While most SDP solvers use *preconditioning* to enhance the numerical stability, it is important to caution against the possible inaccuracy caused simply by the large input coefficients, which can occur even when preconditioning is used.

Example 7. Consider the entailment checking problem instance:

$$\underbrace{q_1(x,y)}_{x+y} \geq 0 \ \wedge \ \underbrace{q_2(x,y)}_{\gamma\cdot(x^2+y-1)} \geq 0 \ \wedge \ \underbrace{q_3(x,y)}_{x-4y^2} \geq 0 \ \models \ p(x,y) \geq 0, \tag{11}$$

where $p : (x^2+y^2)(q_1(x,y)+q_2(x,y)+q_3(x,y)+8)$ and γ is a user-specified constant. For any $\gamma \in \mathbb{R}$, (11) is true, and the corresponding SOS problem has an obvious solution ($\sigma_i : x^2 + y^2$ for $i = 1, \ldots, 3$) that is independent of γ. Even though theoretically the solution set remains the same for varying γ, we see from Table 1 that a mere change in the value of γ can affect the solution accuracy in some SDP solvers: in this example, SDPT3 appears more robust against ill conditioning than SeDuMi. The large value of γ worsens the conditioning of the linear system solved in each iteration of an IPM and can lead to significant inaccuracy.

Table 1: The relative residual norm of the solutions returned by SDPT3 [59] and SeDuMi [56] for varying values of γ.

	$\gamma = 1$	$\gamma = 10^3$	$\gamma = 10^6$	$\gamma = 10^9$
SDPT3	2.1×10^{-8}	5.4×10^{-10}	5.1×10^{-9}	2.3×10^{-8}
SeDuMi	5.5×10^{-9}	2.6×10^{-9}	3.3×10^{-5}	0.00023

Floating-Point Errors. For the sake of efficiency, IPMs are implemented using floating-point arithmetic. Thus, the precision of the floating-point format used limits the accuracy of the result. The most commonly used floating-point format offers a precision of about 10^{-16} for arithmetic operations and SDP solvers usually offer accuracies ϵ around 10^{-8} [10,63]. Higher accuracies can be reached using more precise (and expensive) floating-point formats such as done by the SDPA-GMP solver (see [5, Ch. 24] and [41]).

4.2 Proving Soundness

Now we describe several different techniques for proving that SDP feasibility problems (10) arising from the SOS formulation of parametric entailment problems admit solutions. These techniques can be separated in two main approaches:

(a) Techniques that attempt to get an actual solution. They are able to solve some empty interior problems but this is often expensive.
(b) Techniques that prove the existence of an actual solution, nearby to an approximate one. They require strict feasibility but are much cheaper.

After a quick review of the first approach, we detail the second one, since numerical tests in Sect. 5 indicate that it is the most useful one for proving polynomial invariants.

Deriving Exact Solutions As already mentioned in Section 3.2, the problem (10) is decidable. Unfortunately, even the most recent algorithms [32] are not meant to be competitive with numerical solvers. Another approach consists in assuming the existence of a rational solution, using a numerical solver and attempting by various means to project its approximate solution to an exact rational solution [30,35,39,45,46]. These methods are truly impressive as they are able to solve some empty interior problems. It is also worth noting that since they provide an exact SOS decomposition, mechanically checking it with a proof assistant like Isabelle/HOL or Coq is particularly simple [30,39]. Unfortunately they require heavy computations in rational arithmetic, which incurs the risk of an exponential blow-up of the size of the denominators.

Proving Existence of a Nearby Solution We now assume that (10) is strictly feasible, call a numerical solver that returns an approximate solution $\widetilde{C}$ and attempt to derive from it a proof that there exists an actual solution C (without actually computing C), based on the following proposition, whose proof is similar to that of [37, Theorem 4].

Proposition 1. *If* (10) *results from the SOS programming problem* (5) *or* (6)*, and* $\widetilde{C} \in \mathbb{R}^{s\times s}$ *satisfies the inequality* $\left(s \max_{i\in\{1,\dots,m\}} |\mathrm{tr}(A_i\widetilde{C}) - b_i|\right) \leq \lambda_{\min}(\widetilde{C})$ *(the smallest eigenvalue of* $\widetilde{C}$*), then* (10) *admits an actual solution* C*.*

This suggests the following method to prove that a SOS problem is feasible:

Step 1. Obtain an approximate solution $\widetilde{C}$.
Step 2. Compute (an overapproximation of) $\epsilon' := \max_{i\in\{1,\dots,m\}} |\mathrm{tr}(A_i\widetilde{C}) - b_i|$.
Step 3. Check that $\widetilde{C} - s\,\epsilon'\,I \succeq 0$ (which implies $s\,\epsilon' \leq \lambda_{\min}(\widetilde{C})$).

Step 1 is achieved using a numerical solver and Step 2 is performed using floating-point interval arithmetic. The hard step is to provide a sound and efficient way to check $\widetilde{C} - s\,\epsilon'\,I \succeq 0$. We rely on a check suggested by the following theorem. Let $\mathbb{F}$ be a floating-point format with unit roundoff $\mathtt{eps}$ and underflow unit $\mathtt{eta}$. For any symmetric floating-point matrix $M \in \mathbb{F}^{s\times s}$ with $2(s+2)\mathtt{eps} < 1$, define $\alpha : \frac{(s+1)\mathtt{eps}}{1-(2s+2)\mathtt{eps}}\mathrm{tr}(M) + 4(s+1)\left(2(s+2) + \max_i M_{i,i}\right)\mathtt{eta}$.

Theorem 2. ([50, Corollary 2.4]) $M \succeq 0$ *if there exists* $\widetilde{M} \in \mathbb{F}^{s\times s}$ *such that the following conditions hold:*

- $\widetilde{M}_{ij} = M_{ij}$*, for any* $i \neq j$*;*
- $\widetilde{M}_{ii} \leq M_{ii} - \alpha$*, for any* i*; and*
- *the Cholesky algorithm implemented in floating-point arithmetic succeeds on* $\widetilde{M}$*, i.e., "concludes" that* $\widetilde{M}$ *is positive semidefinite,*

Theorem 2 is used to prove that $\widetilde{C} - s\,\epsilon'\,I \succeq 0$, as follows:

- compute $M := \widetilde{C} - s\,\epsilon'\,I$ using floating-point arithmetic with rounding toward $-\infty$. It follows that the error $(\widetilde{C} - s\,\epsilon'\,I) - M$ will be a diagonal matrix with nonnegative entries. Hence, if $M \succeq 0$ then $\widetilde{C} - s\,\epsilon'\,I \succeq 0$, as well.
- check that M is symmetric and that $2(s+2)\texttt{eps} < 1$;
- compute $\widetilde{M} := M - \alpha\,I$ with rounding toward $-\infty$ (the closest $\widetilde{M}$ to $M - \alpha\,I$, the more likely its Cholesky decomposition is to succeed);
- compute the Cholesky decomposition of $\widetilde{M}$.

If the Cholesky decomposition succeeds (which happens when, e.g., $\lambda_{\min}(\widetilde{C}) \geq s\,\epsilon' + 2\alpha$ [18]), then $\widetilde{C} - s\,\epsilon'\,I \succeq 0$.

Remark 2. For the IEEE 754 [33] binary64 format with rounding to nearest[8], $\texttt{eps} = 2^{-53}$ ($\simeq 10^{-16}$) and $\texttt{eta} = 2^{-1075}$ ($\simeq 10^{-323}$). Thus, the hypothesis $2(s+2)\texttt{eps} < 1$ is always satisfied for practical values of s. Moreover, for typical values ($s \leq 1000$ and elements of M of order of magnitude 1), $\alpha \leq 10^{-10}$. This is negligible in front of $s\,\epsilon' \sim 10^{-8}s$ (10^{-8} being the typical default stopping tolerance), which means that the incompleteness of this positive definiteness check is not an issue in practice.

Steps 2 and 3 can be performed in only $\mathrm{O}(s^3)$ floating-point operations (cost of the Cholesky decomposition) so the cost of the whole method is dominated by the call to the numerical SDP solver in Step 1.

Remark 3. For ease of exposition, the above technique was presented on the whole matrix C, although it is preferable to apply it on each block C_j of C.

Padding the SDP Problem. Naturally, all this requires that the least eigenvalue of the solution returned by the numerical solver be larger than $s\,\epsilon'$. It could seem that ϵ' is known only after numerically solving the SDP problem, since it is computed from its result in Step 2. In fact, ϵ' will be less than the stopping criterion ϵ of the solver, which is known in advance. Thus instead of solving (10), we solve the slightly modified problem

$$\begin{array}{ll} \text{find}: & C \ \ \text{s.t.} \ \ \mathrm{tr}(A_i C) = b_i,\ i = 1, \ldots, m, \\ & C - s\,\epsilon\,I \succeq 0, \end{array}$$

which is an SDP (up to the change of variable $C \mapsto C + s\,\epsilon\,I$).

The simple criterion in Proposition 1 assumes SDP problems translated from SOS problems. On the other hand, the tool VSDP [31,34] verifies the solutions of general SDP problems using interval arithmetic results.

Remark 4. Mechanically checking proofs generated by the three step method of this section is an ongoing project. To this end, Theorem 2 has been verified [49] in Coq.

5 Experiments

[8] Type `double` in C.

This section presents an experimental evaluation of the methods described in Sections 3 and 4 on the examples of ADJÉ et al. [1]. We first synthesize polynomial invariants for these programs, following [1], then attempt to formally prove their soundness. As seen in Section 2, these formal proofs are particularly worthwhile as synthesizing incorrect invariants is quite easy.

```
x ∈ { x ∈ ℝ^n | ∧_{j=1}^k i_j(x) ≥ 0 }
while (1)
  if (g(x) <= 0)
    x = τ1(x)
  else
    x = τ2(x)
```

Fig. 6: Benchmarks form.

Considered programs are of the form in Fig. 6. An invariant $p(\mathbf{c}, \mathbf{x}) \geq 0$ can be provided by any solution[9] of the parametric entailment problem:

$$\begin{cases} i_1(\mathbf{x}) \geq 0 \ \wedge \ \cdots \ \wedge \ i_k(\mathbf{x}) \geq 0 \models p(\mathbf{c}, \mathbf{x}) \geq 0 \\ g(\mathbf{x}) \leq 0 \models (p \circ \tau_1)(\mathbf{c}, \mathbf{x}) \geq p(\mathbf{c}, \mathbf{x}) \\ g(\mathbf{x}) \geq 0 \models (p \circ \tau_2)(\mathbf{c}, \mathbf{x}) \geq p(\mathbf{c}, \mathbf{x}). \end{cases}$$

(See Section 3.) Thus, any solution of the following SOS problem gives an invariant:

$$\begin{aligned} \text{find :} \ & \text{polynomials } \sigma_j \in \mathbb{R}_{d-d_{i_j}}[\mathbf{x}] \ (j \in \{1, \ldots, k\}), \sigma_{k+1}, \sigma_{k+2} \in \mathbb{R}_{d-d_g}[\mathbf{x}] \\ \text{s.t.} \ & p - \textstyle\sum_{j=1}^k \sigma_j i_j \text{ is SOS}, \\ & (p \circ \tau_1) - p + \sigma_{k+1}\, g \text{ is SOS}, \\ & (p \circ \tau_2) - p - \sigma_{k+2}\, g \text{ is SOS}, \\ & \sigma_1, \ldots, \sigma_{k+2} \text{ are SOS}, \end{aligned} \tag{12}$$

where d is the degree of p and $d_{i_1}, \ldots, d_{i_k}$ and d_g are the degrees of $i_1, \ldots, i_k$ and g respectively (all assumed to be less than d).

Table 2 gives the time needed to synthesize candidate invariants of degree d equal to 4, 6, 8 and 10 by solving the above SOS problem. "Example 4" in this table corresponds to the program of Fig. 2. The candidate invariant obtained for degree $d = 6$ is given in Fig. 2 and displayed in Fig. 3. The one obtained for $d = 8$ is also displayed in Fig. 3. "Example 8" corresponds to Fig. 4.

Unfortunately, the problem (12) usually has an empty interior[10]. This means that the candidate invariant obtained from numerical solvers does not precisely satisfy (12). In fact, there often exist values $\mathbf{x}_0$ such that $i_1(\mathbf{x}_0) \geq 0, \ldots, i_k(\mathbf{x}_0) \geq 0$ and $p(\mathbf{x}_0)$ is a tiny negative value. To fix that, we look for a small[11] $c \in \mathbb{R}$ such that $p + c - \sum_{j=1}^k \sigma_j i_j$ is SOS for SOS polynomials σ_j. This is done using the padding technique of Section 4. Times in table 2 include this fixing step.

We now attempt to prove that the fixed candidate invariants p are correct by considering the following entailment checking problem

$$\begin{aligned} & i_1(\mathbf{x}) \geq 0 \ \wedge \ \cdots \ \wedge \ i_k(\mathbf{x}) \geq 0 \models p(\mathbf{x}) \geq 0 \\ & g(\mathbf{x}) \leq 0 \ \wedge \ p(\mathbf{x}) \geq 0 \models (p \circ \tau_1)(\mathbf{x}) \geq 0 \\ & g(\mathbf{x}) \geq 0 \ \wedge \ p(\mathbf{x}) \geq 0 \models (p \circ \tau_2)(\mathbf{x}) \geq 0. \end{aligned} \tag{13}$$

[9] To get a "small" invariant, one minimizes the radius of the ball enclosing it [1].

[10] Assignments τ often admit a fixpoint $\mathbf{x}_0 = \tau(\mathbf{x}_0)$ meaning that the condition $(p \circ \tau) - p + \sigma\, g \geq 0$ boils down in $\mathbf{x}_0$ to $\sigma(\mathbf{x}_0)\, g(\mathbf{x}_0) \geq 0$ implying $\sigma(\mathbf{x}_0) = 0$ when $g(\mathbf{x}_0) < 0$.

[11] In practice, $c < 10^{-3}$ when coefficients of p are of order of magnitude 1.

We first evaluated methods looking for exact solutions with the implementation of MONNIAUX and CORBINEAU [39]. Table 3 gives the results. The checking process is split in two parts: init for the initialization property (first entailment of (13)) and ind. for the inductiveness property (remaining entailments). As seen in the table, most of the initialization properties are indeed proved but proofs of the inductiveness property fail for all but the smallest example. This can be explained by the size of the corresponding SDP problems. For the initialization property, the largest block is a matrix of size $\binom{n+\frac{d}{2}}{n} \times \binom{n+\frac{d}{2}}{n}$ whereas for inductiveness it is of size $\binom{n+\frac{d\,d_\tau}{2}}{n} \times \binom{n+\frac{d\,d_\tau}{2}}{n}$. This is too much[12] to perform heavy computations with exact rational arithmetic.

Although (12) usually has an empty interior, it is worth noting that this unfortunate property is due to the relaxation and is not intrinsic to the problem. Indeed, the loop body τ of the considered programs are usually strictly contractive, i.e., the image of the invariant $\{\mathbf{x} \mid p(\mathbf{x}) \geq 0\}$ by τ is included in its interior. When τ is continuous, this means that any polynomial $\tilde{p}$ close enough from p also defines an invariant $\{\mathbf{x} \mid \tilde{p}(\mathbf{x}) \geq 0\}$. In fact, the entailment checking problem (13) commonly leads to strictly feasible SDP problems. Thus, the method presented in Section 4.2 can be used to efficiently prove the soundness of a large part of the candidate invariants, as seen in Table 4. The time needed to compute the proofs (Table 4) is comparable to the time needed to synthesize the invariants (Table 2). Indeed, most of this time is spent running SDP solvers.

These results are confirmed by VSDP [31,34] when we provide it the SDP problems corresponding to (13) and the strictly feasible solutions we computed using SDP solvers. This again indicates that these numerical verification methods only induce a very small overhead compared to the time required to run SDP solvers.

Implementation. The SOS to SDP translation described in Section 3, as well as the validation method described in Section 4.2 have been implemented in our OCaml library OSDP. It offers an interface to the SDP solvers Csdp [10], Mosek [40], SDPA [63] and SDPA-GMP [41] and is available at `http://cavale.enseeiht.fr/osdp/`. Results from Tables 2 and 4 have been obtained thanks to a small static analyzer relying on the library and available, along with all benchmarks, at `http://cavale.enseeiht.fr/validatingSDP2016/`. All computations were performed with the Mosek solver on a Xeon @ 2.67GHz.

6 Conclusion

Thus far, we have reviewed the use of SOS relaxations and numerical SDP solvers to solve polynomial problems arising in static analysis of programs. We presented some examples and experiments showing that, although erroneous results are often obtained from numerical solvers, rigorous proofs of soundness are possible. Moving forward, we wish to examine the application of our approach inside theorem provers and applications to hybrid systems, as well.

Acknowledgments: The authors would like to thank Didier Henrion, Pierre-Loïc Garoche and Assalé Adjé for interesting discussions on this subject.

[12] For $n = 2$, $d_\tau = 3$ and $d = 8$, $\binom{n+\frac{d}{2}}{n} = \binom{6}{2} = 15$ whereas $\binom{n+\frac{d\,d_\tau}{2}}{n} = \binom{14}{2} = 91$.

Table 2: Time to synthesize candidate invariants for benchmarks [1]. n is the number of variables and d_τ the degree of the polynomial assignments. All times are in seconds, TO means timeout (900s) and MO out of memory (4GB).

	$d=4$	$d=6$	$d=8$	$d=10$
Example 4 ($n=2$, $d_\tau=3$)	0.25	0.95	3.31	8.85
Example 5 ($n=3$, $d_\tau=2$)	0.48	2.83	22.75	112.37
Example 6 ($n=4$, $d_\tau=2$)	2.12	64.07	TO	MO
Example 7 ($n=2$, $d_\tau=3$)	0.25	0.96	3.15	10.45
Example 8 ($n=2$, $d_\tau=2$)	0.17	0.34	0.74	1.93

Table 3: Checking the candidate invariants with the implementation of MONNIAUX and CORBINEAU [39]. All times are in seconds, NS means that no proof is found, TO means timeout (900s) and MO out of memory (4GB).

	$d=4$		$d=6$		$d=8$		$d=10$	
	init	ind.	init	ind.	init	ind.	init	ind.
Example 4 ($n=2$, $d_\tau=3$)	1.43	NS	3.35	TO	19.80	MO	142.33	MO
Example 5 ($n=3$, $d_\tau=2$)	3.82	TO	142.49	MO	TO	MO	TO	MO
Example 6 ($n=4$, $d_\tau=2$)	32.20	TO	TO	MO	—	—	—	—
Example 7 ($n=2$, $d_\tau=3$)	1.48	NS	3.36	TO	18.36	MO	120.40	MO
Example 8 ($n=2$, $d_\tau=2$)	1.93	12.81	3.78	NS	26.29	TO	193.79	TO

Table 4: Checking the candidate invariants with the method of Section 4 (computing strictly feasible SDP solutions and verifying them). All times are in seconds. As seen in Section 2, counter-examples are easily found for Ex. 4, $d=4$ and 6 and Ex. 7, $d=4$. No such counter-examples were found for the other unproved cases and it remains unknown whether they are actually inductive or not.

	$d=4$		$d=6$		$d=8$		$d=10$	
	init	ind.	init	ind.	init	ind.	init	ind.
Example 4 ($n=2$, $d_\tau=3$)	0.05	NS	0.07	NS	0.19	3.03	0.17	NS
Example 5 ($n=3$, $d_\tau=2$)	0.08	0.33	0.23	2.20	0.74	14.55	2.50	92.15
Example 6 ($n=4$, $d_\tau=2$)	0.22	1.52	1.26	38.94	—	—	—	—
Example 7 ($n=2$, $d_\tau=3$)	0.05	NS	0.07	0.85	0.19	3.32	0.17	NS
Example 8 ($n=2$, $d_\tau=2$)	0.05	0.13	0.07	NS	0.09	NS	0.15	NS

Table 5: Rechecking the proofs of Table 4 with VSDP [31,34] (verifying given strictly feasible SDP solutions). All times are in seconds.

	$d=4$		$d=6$		$d=8$		$d=10$	
	init	ind.	init	ind.	init	ind.	init	ind.
Example 4 ($n=2$, $d_\tau=3$)	0.04	NS	0.06	NS	0.06	0.30	0.07	NS
Example 5 ($n=3$, $d_\tau=2$)	0.06	0.18	0.09	0.26	0.16	0.80	0.27	2.52
Example 6 ($n=4$, $d_\tau=2$)	0.10	0.30	0.27	1.11	—	—	—	—
Example 7 ($n=2$, $d_\tau=3$)	0.05	NS	0.05	0.15	0.06	0.25	0.07	NS
Example 8 ($n=2$, $d_\tau=2$)	0.04	0.07	0.03	NS	0.04	NS	0.05	NS

References

1. Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. Property-based Polynomial Invariant Generation Using Sums-of-Squares Optimization. In *SAS*, 2015.
2. Assalé Adjé, Stéphane Gaubert, and Éric Goubault. Coupling Policy Iteration with Semi-definite Relaxation to Compute Accurate Numerical Invariants in Static Analysis. In *ESOP*, 2010.
3. Amir Ali Ahmadi and Anirudha Majumdar. DSOS and SDSOS optimization: LP and SOCP-based alternatives to sum of squares optimization. In *Annual Conference on Information Sciences and Systems (CISS)*, 2014.
4. Xavier Allamigeon, Stéphane Gaubert, Eric Goubault, Sylvie Putot, and Nikolas Stott. A scalable algebraic method to infer quadratic invariants of switched systems. In *EMSOFT*, 2015.
5. Miguel F. Anjos and Jean Bernard Lasserre. Introduction to semidefinite, conic and polynomial optimization. In *Handbook on semidefinite, conic and polynomial optimization*. Springer, 2012.
6. Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. Generation of Basic Semi-algebraic Invariants Using Convex Polyhedra. In *SAS*, 2005.
7. Sugata Basu, Richard Pollock, and Marie-Francoise Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2006.
8. Mohamed Amin Ben Sassi, Sriram Sankaranarayanan, Xin Chen, and Erika Abraham. Linear Relaxations of Polynomial Positivity for Polynomial Lyapunov Function Synthesis . *IMA Journal of Mathematical Control and Information*, 2015.
9. Sergei Nanatovich Bernstein. Démonstration du théoréme de Weierstrass fondée sur le calcul des probabilités. *Communcations de la Société Mathématique de Kharkov 2*, 1912.
10. Brian Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 1999.
11. Jon M. Borwein and Henry Wolkowicz. Facial reduction for a cone-convex programming problem. *J. Austral. Math. Soc. Ser. A*, 1980/81.
12. Aleksandar Chakarov, Yuen-Lam (Vris) Voronin, and Sriram Sankaranarayanan. Deductive Proofs of Almost Sure Persistence and Recurrence Properties . In *TACAS*, 2016.
13. George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, 1975.
14. George E. Collins and Hoon Hong. Partial Cylindrical Algebraic Decomposition for Quantifier Elimination. *Journal of Symbolic Computation*, 1991.
15. Patrick Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *VMCAI*, 2005.
16. Patrick Cousot and Radhia Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *POPL*, 1977.
17. Thao Dang and Thomas Martin Gawlitza. Template-Based Unbounded Time Verification of Affine Hybrid Automata. In *APLAS*, 2011.
18. James Demmel. On floating point errors in Cholesky. Lapack working note, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1989.
19. Andreas Dolzmann and Thomas Sturm. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bull.*, 1997.
20. Mirjam Dür, Bolor Jargalsaikhan, and Georg Still. The Slater condition is generic in linear conic programming, 2012.
21. Rida T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 2012.
22. Éric Féron. From control systems to control software. *Control Systems, IEEE*, 2010.

23. Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation, Special Issue on SAT/CP Integration*, 2007.
24. Sicun Gao, Soonho Kong, and Edmund M. Clarke. dreal: An SMT solver for nonlinear theories over the reals. In *Intl. Conference on Automated Deduction (CADE)*, 2013.
25. Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou. Static analysis by policy iteration on relational domains. In *ESOP*, 2007.
26. Thomas Gawlitza and Helmut Seidl. Precise fixpoint computation through strategy iteration. In *ESOP*, 2007.
27. Thomas Martin Gawlitza and David Monniaux. Improving strategies via SMT solving. In *ESOP*, 2011.
28. Thomas Martin Gawlitza and Helmut Seidl. Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In *SAS*, 2010.
29. David Handelman. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math*, 1988.
30. John Harrison. Verifying nonlinear real formulas via sums of squares. In *TPHOL*, 2007.
31. Viktor Härter, Christian Jansson, and Marko Lange. VSDP: verified semidefinite programming. http://www.ti3.tuhh.de/jansson/vsdp/. Accessed on March 28, 2016.
32. Didier Henrion, Simone Naldi, and Mohab Safey El Din. Exact algorithms for linear matrix inequalities. *arXiv preprint arXiv:1508.03715*, 2015.
33. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*, 2008.
34. Christian Jansson, Denis Chaykin, and Christian Keil. Rigorous error bounds for the optimal value in semidefinite programming. *SIAM J. Numerical Analysis*, 2007.
35. Erich Kaltofen, Bin Li, Zhengfeng Yang, and Lihong Zhi. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. *J. Symb. Comput.*, 2012.
36. Jean Bernard Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 2001.
37. Johan Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 2009.
38. Alexandre Maréchal, Alexis Fouilhé, Tim King, David Monniaux, and Michaël Périn. Polyhedral approximation of multivariate polynomials using handelman's theorem. In *VMCAI*, 2016.
39. David Monniaux and Pierre Corbineau. On the generation of positivstellensatz witnesses in degenerate cases. In *ITP*, 2011.
40. MOSEK ApS. *The MOSEK C optimizer API manual Version 7.1 (Revision 40)*, 2015.
41. Maho Nakata. A numerical evaluation of highly accurate multiple-precision arithmetic version of semidefinite programming solver: SDPA-GMP, -QD and -DD. In *Computer-Aided Control System Design*, 2010.
42. Mendes Oulamara and Arnaud J. Venet. Abstract interpretation with higher-dimensional ellipsoids and conic extrapolation. In *CAV*, 2015.
43. Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, 2003.
44. Frank Permenter and Pablo Parrilo. Partial facial reduction: simplified, equivalent sdps via approximations of the psd cone. *arXiv preprint arXiv:1408.4685*, 2014.
45. Helfried Peyrl and Pablo A. Parrilo. Computing sum of squares decompositions with rational coefficients. *Theor. Comput. Sci.*, 2008.
46. André Platzer, Jan-David Quesel, and Philipp Rümmer. Real world verification. In *CADE*, 2009.

47. Stephen Prajna and Ali Jadbabaie. Safety verification using barrier certificates. In *HSCC*, 2004.
48. Mihai Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana Univ. Math. Journal*, 1993.
49. Pierre Roux. Formal proofs of rounding error bounds. *Journal of Automated Reasoning*, 2015.
50. Siegfried M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 2006.
51. Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 2008.
52. Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, 2005.
53. Stefan H. Schmieta and Gabor Pataki. Reporting solution quality for the DIMACS library of mixed semidefinite-quadratic-linear programs. http://dimacs.rutgers.edu/Challenges/Seventh/Instances/error_report.html. [Online; accessed March 23, 2016].
54. Hanif D. Sherali and Cihan H. C.H. Tuncbilek. A global optimization algorithm for polynomial programming using a reformulation-linearization technique. *Journal of Global Optim.*, 1991.
55. Naum Z. Shor. Class of global minimum bounds on polynomial functions. *Cybernetics*, 1987. Originally in Russian: Kibernetika, 1987.
56. Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 1999.
57. Alfred Tarski. A decision method for elementary algebra and geometry. Technical report, Univ. of California Press, Berkeley, 1951.
58. Levent Tuncel. *Polyhedral and semidefinite programming methods in combinatorial optimization*. American Mathematical Society, 2010.
59. Reha H Tütüncü, Kim C Toh, and Michael J Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical programming*, 2003.
60. Hayato Waki, Maho Nakata, and Masakazu Muramatsu. Strange behaviors of interior-point methods for solving semidefinite programming problems in polynomial optimization. *Computational Optimization and Applications*, 2011.
61. Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. In *Applied Algebra and Error-Correcting Codes (AAECC)*, 1997.
62. Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming*. Kluwer Academic Publishers, 2000.
63. Makoto Yamashita, Katsuki Fujisawa, Kazuhide Nakata, Maho Nakata, Mituhiro Fukuda, Kazuhiro Kobayashi, and Kazushige Goto. A high-performance software package for semidefinite programs: SDPA 7. Technical Report B-460, Tokyo Institute of Technology, 2010.