# A reflexive tactic for polynomial positivity using numerical solvers and floating-point computations

Érik Martin-Dorel, Pierre Roux

# A Reflexive Tactic for Polynomial Positivity using Numerical Solvers and Floating-Point Computations *

Érik Martin-Dorel
IRIT, Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 9, France
erik.martin-dorel@irit.fr

Pierre Roux
ONERA
2 avenue Édouard Belin
31055 Toulouse Cedex 4, France
pierre.roux@onera.fr

## Abstract

Polynomial positivity over the real field is known to be decidable but even the best algorithms remain costly. An incomplete but often efficient alternative consists in looking for positivity witnesses as sum of squares decompositions. Such decompositions can in practice be obtained through convex optimization. Unfortunately, these methods only yield approximate solutions. Hence the need for formal verification of such witnesses. State of the art methods rely on heuristic roundings to exact solutions in the rational field. These solutions are then easy to verify in a proof assistant. However, this verification often turns out to be very costly, as rational coefficients may blow up during computations.

Nevertheless, overapproximations with floating-point arithmetic can be enough to obtain proofs at a much lower cost. Such overapproximations being non trivial, it is mandatory to formally prove that rounding errors are correctly taken into account. We develop a reflexive tactic for the Coq proof assistant allowing one to automatically discharge polynomial positivity proofs. The tactic relies on heavy computation involving multivariate polynomials, matrices and floating-point arithmetic. Benchmarks indicate that we are able to formally address positivity problems that would otherwise be untractable with other state of the art methods.

***Categories and Subject Descriptors*** D.2.4 [*Software Engineering*]: Software/Program Verification—Formal methods; G.1.6 [*Numerical analysis*]: Optimization

***Keywords*** Coq formal proof, reflexive tactic, multivariate polynomials, witness verification, SDP solvers, floating-point, data refinement, Cholesky decomposition

* This work has been partially supported by the FAGames project of LabEx CIMI, the French ANR project ANR-12-INSE-0007 Cafein and the project SEFA IKKY.



## 1. Introduction and Motivation

Satisfiability of conjunctions of polynomial inequalities in the real field is known to be decidable for more than half a century (Tarski 1951). Although decision algorithms have been improved since then, their complexity remains daunting, even for the best ones such as the cylindrical algebraic decomposition (CAD) (Collins 1975). A practical semi-decision procedure is offered by relaxing polynomial positivity to sum of squares (SOS). These SOS constraints can then be encoded as semidefinite programming (SDP) problems for which efficient optimization algorithms are available (Lasserre 2001; Parrilo 2003). Unfortunately, these algorithms only compute approximate solutions which becomes a major obstacle when it comes to use them as rigorous proof witnesses.

Most state of the art solutions resort to some kind of rounding to exact rational solutions. Although this can be a simple way to obtain formal proofs out of SOS witnesses, those proofs involve heavy exact computations in the rational field. This can be dramatically expensive, as the denominators of the rational numbers may blow up during these computations. An efficient alternative consists in keeping the approximate solutions and using floating-point computations while carefully taking the induced overapproximation into account. This involves computations with polynomials, matrices and floating-point arithmetic.

The main contribution of this paper is to demonstrate that such rigorous proofs can be mechanized in a proof assistant with nice computation capabilities such as Coq (Coq 2016). We also identify which mechanisms could be added to Coq in order to make this kind of rigorous proofs based on floating-point computations much more efficient.

After the next section describing related work, Section 2 gives a presentation of the main algorithm and some proof schemes of its correctness. Section 3 presents the data refinements that were needed in order to obtain an executable

version of the previous algorithm. Then, Section 4 details our tactic that is able to automatically discharge polynomial inequality goals thanks to the previous programs. Finally, Section 5 comments the results on some benchmarks and compares our tactic to other tools while Section 6 concludes.

### 1.1 Related Work

There have been several works for developing (semi)decision procedures for nonlinear inequalities in formal proof assistants. Some of them follow the so-called *autarkic* approach and perform all the required computations within the prover itself. Others follow the so-called *skeptical* approach and delegate most of the computations to some external, possibly unsound yet efficient oracle. In this latter case the prover only needs to verify the witnesses generated by the oracle, by using a dedicated algorithm that has been formally verified.

First, let us focus on the HOL Light proof assistant. The REAL_SOS decision procedure (Harrison 2007) relies on the CSDP library (Borchers 1999) for semidefinite programming. It generates a semidefinite programming problem from the user's goal, calls CSDP and tries to infer an "exact" solution. This is done by rounding the approximation solution returned by the SDP solver to "rational numbers with moderate coefficients".

Another decision procedure for the HOL Light proof assistant has been developed as part of the Flyspeck project[1]: verify_ineq (Solovyev and Hales 2013). It does not involve SDP computations but relies on the computation of Taylor polynomials, using interval arithmetic and bisection. To be more specific, it uses an external search procedure to precompute a "solution certificate" that is useful to speed-up the *a posteriori* verification. These certificates notably describe how the input domain should be split. Then, the verification procedure computes order-1 Taylor-Lagrange enclosures (i.e., with quadratic reminders) using interval arithmetic and floating-point numbers. Contrarily to REAL_SOS, verify_ineq supports transcendental functions such as $\cos$ and $\arctan$.

Focusing on the Coq proof assistant, L. Théry has isolated the HOL-independent code of REAL_SOS that F. Besson has then integrated in several decision procedures within the Micromega Coq library (Besson 2006). The approach of Micromega is the same as that of REAL_SOS: it depends on CSDP and relies on the Positivstellensatz.

Another decision procedure that is amenable to Coq formal proof has been developed with a special focus on certifying SDP problems with empty interior (Monniaux and Corbineau 2011). This package has been written in Sage and relies on DSDP (Benson and Ye 2008) and fpLLL[2]: it can be viewed as an enhancement of (Harrison 2007)'s approach, with a better rounding heuristic that takes advantage of the LLL algorithm (Lenstra et al. 1982).

[1] `https://github.com/flyspeck/flyspeck`

[2] `https://github.com/fplll/fplll`

The Coq.Interval library provides a decision procedure for nonlinear inequalities with transcendental functions. It has been developed with a special focus on verifying approximation errors for univariate expressions, but is also able to solve quasi-multivariate problems (Martin-Dorel and Melquiond 2016). It follows the *autarkic* approach and involves floating-point and interval arithmetic, bisection, and computation of univariate order-$n$ Taylor polynomials.

Within the Flyspeck project, a decision procedure named NLCertify has been devised for the Coq proof assistant (Magron 2014). It relies on the SDPA solver, on an OCaml backend that generates positivity certificates, and on a Coq verification procedure that mainly relies on the `ring` tactic. NLCertify supports nonlinear multivariate inequalities with transcendental functions over an hyperbox, but only the polynomial goals are formally certified currently.

Two decision procedures are available for the PVS proof assistant: `interval` (Narkawicz and Muñoz 2013) and `bernstein` (Muñoz and Narkawicz 2013). Both rely on a branch-and-bound algorithm and follow the *autarkic* approach. The `interval` PVS strategy supports transcendental functions and uses interval arithmetic with rational bounds. The `bernstein` PVS strategy relies on a representation of multivariate polynomials in Bernstein form to easily infer bounds on them.

It is worth noting that all the above work performing proofs in exact rational arithmetic may be able to prove "sharp" inequalities. In contrast, the tactic presented in this paper will usually only be able to prove inequalities satisfied within some margin (i.e., inequalities $p < q$ such that $p + \varepsilon < q$ holds for some $\varepsilon > 0$). This is inherent in the floating-point arithmetic roundings.

## 2. SOS and Cholesky Decomposition

### 2.1 Sum of Squares (SOS) Programming

The sum of squares relaxation (Lasserre 2001; Parrilo 2003) is an incomplete but efficient way to numerically solve polynomial problems. This section aims at recalling its main ideas which are required to understand the main contribution of the paper.

A multivariate polynomial $p \in \mathbb{R}[x]$ is said to be a sum of squares when there exist polynomials $h_i \in \mathbb{R}[x]$ such that, for all $x \in \mathbb{R}^n$,

$$p(x) = \sum_i h_i^2(x).$$

Although not all non negative polynomials are sum of squares, being a sum of squares is a sufficient condition to be non negative.

**Example 1.** *Considering* $p(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$*, there exist* $h_1(x_1, x_2) = \frac{1}{\sqrt{2}}\left(2x_1^2 - 3x_2^2 + x_1x_2\right)$ *and* $h_2(x_1, x_2) = \frac{1}{\sqrt{2}}\left(x_2^2 + 3x_1x_2\right)$ *such that* $p = h_1^2 + h_2^2$*. This proves that for all* $x_1, x_2 \in \mathbb{R}$*,* $p(x_1, x_2) \geq 0$*.*

Any polynomial $p$ of degree $2d$ (a non negative polynomial is necessarily of even degree) can be written as a quadratic form in the vector of all monomials of degree less or equal $d$:

$$p(x) = z^T Q\, z \tag{1}$$

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

**Remark 1.** *If the polynomial $p$ is* homogeneous*, that is if all its monomials have the same degree $2d$ (as in Example 1), then only monomials of degree* exactly *$d$ are required in the vector $z$.*

**Example 2.** *For $p(x_1, x_2) = 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4$, according to the above Remark, we can use the vector of monomials $z = \left[x_1^2, x_2^2, x_1x_2\right]^T$. We then have*

$$\begin{aligned} p(x_1, x_2) &= 2x_1^4 + 2x_1^3x_2 - x_1^2x_2^2 + 5x_2^4 \\ &= \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix}^T \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{12} & q_{22} & q_{23} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \end{bmatrix} \\ &= q_{11}x_1^4 + 2q_{13}x_1^3x_2 + (q_{33} + 2q_{12})x_1^2x_2^2 \\ &\qquad + 2q_{23}x_1x_2^3 + q_{22}x_2^4. \end{aligned}$$

*Thus $q_{11} = 2$, $2q_{13} = 2$, $q_{33} + 2q_{12} = -1$, $2q_{23} = 0$ and $q_{22} = 5$. 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 a sum of squares if and only if there exist a positive semidefinite matrix $Q$ satisfying (1). A matrix $Q$ is said positive semidefinite when, for all vectors $x$, $x^T Q\, x \geq 0$. This will be denoted by $Q \succeq 0$.

**Example 3.** *In the previous example, the first matrix $Q$ is not positive semidefinite (for $x = [0, 0, 1]^T$, $x^T Q\, x = -3$). In contrast, 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 $x$, $x^T Q x = (Lx)^T(Lx) = \|Lx\|_2^2 \geq 0$). This gives the sum of squares decomposition of Example 1: $p(x_1, x_2) = \frac{1}{2}(2x_1^2 - 3x_2^2 + x_1x_2)^2 + \frac{1}{2}(x_2^2 + 3x_1x_2)^2$.*

### 2.2 Semidefinite Programming (SDP)

Given symmetric matrices $C, A_1, \ldots, A_m \in \mathbb{R}^{s\times s}$ and scalars $a_1, \ldots, a_m \in \mathbb{R}$, the following optimization problem is called *semidefinite programming*

$$\begin{array}{ll} \text{minimize} & \operatorname{tr}(CQ) \\ \text{subject to} & \operatorname{tr}(A_1Q) = a_1 \\ & \quad \vdots \\ & \operatorname{tr}(A_mQ) = a_m \\ & Q \succeq 0 \end{array} \tag{2}$$

where the symmetric matrix $Q \in \mathbb{R}^{s\times s}$ is the variable and $\operatorname{tr}(M) = \sum_i M_{i,i}$ denotes the trace of the matrix $M$.

**Remark 2.** *Since the matrices are symmetric, $\operatorname{tr}(AQ) = \operatorname{tr}(A^TQ) = \sum_{i,j} A_{i,j}Q_{i,j}$. The constraints $\operatorname{tr}(AQ) = a$ are then affine constraints between the elements of the matrix $Q$, the variable $Q_{i,j}$ being affected the coefficient $A_{i,j}$.*

Semidefinite programming is a convex optimization problem for which there exist efficient numerical solvers (Boyd and Vandenberghe 2004; Vandenberghe and Boyd 1996).

As we have just seen in Section 2.1, existence of a sum of squares decomposition amounts to existence of a positive semidefinite matrix satisfying a set of affine constraints, that is a solution of a semidefinite program. Thus, semidefinite programming provides an efficient way to numerically solve problems involving polynomial inequalities, by relaxing them as sum of squares constraints.

**Example 4.** *The affine constraints computed in Example 2 can be encoded as a semidefinite program with the following constraints:*

$$\operatorname{tr}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} Q\right) = 2, \qquad \operatorname{tr}\left(\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} Q\right) = 2,$$

$$\operatorname{tr}\left(\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} Q\right) = -1, \quad \operatorname{tr}\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} Q\right) = 0,$$

$$\operatorname{tr}\left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} Q\right) = 5.$$

### 2.3 Parameterized Problems

Up to now, we have only seen how to check whether a given *fixed* polynomial $p$ is sum of squares (which implies its non negativeness). One of the great strength of SOS programming is to enable to solve problems with unknown polynomials.

An unknown polynomial $p \in \mathbb{R}[x]$ with $n$ variables and of degree $d$ can be written

$$p = \sum_{\alpha_1+\cdots+\alpha_n \leq d} p_\alpha x_1^{\alpha_1} \ldots x_n^{\alpha_n}$$

where the $p_\alpha$ are scalar parameters. The $p_\alpha$ then just translate to additional variables in the resulting SDP problem.

This enables to relax polynomial problems with polynomial constraints.

**Example 5.** *Given two polynomials $p$ and $q$, to prove that $p(x) \geq 0$ whenever $q(x) \geq 0$, one can exhibit a polynomial $\sigma$ such that*

$$p - \sigma q \geq 0 \wedge \sigma \geq 0.$$

*Indeed, for any $x$, if $q(x) \geq 0$ then $p(x) \geq \sigma(x)\, q(x) \geq 0$.*

The reader interested in more details is referred to the literature (Lasserre 2009; Löfberg 2009).

### 2.4 Numerical Verification of SOS

#### 2.4.1 Approximate Solutions from SDP Solvers

In practice, the matrix $Q$ returned by SDP solvers upon solving an SDP problem (2) does not precisely satisfy the equality constraints. Therefore, although the SDP solver returns a positive answer for a SOS program, this does not translate to a valid proof that a given polynomial is SOS. This section details an incomplete but efficient validation method to address this issue.

Most SDP solvers start from some $Q \succeq 0$ not satisfying the equality constraints (for instance the identity matrix) and iteratively modify it in order to reduce the distance between $\mathrm{tr}(A_i Q)$ and $a_i$ while keeping $Q$ positive semidefinite. This process is stopped when this distance is deemed small enough.

Therefore, we do not obtain a $Q$ satisfying $\mathrm{tr}(A_i Q) = a_i$ but rather $\mathrm{tr}(A_i Q) = a_i + \delta_i$ for some small[3] $\epsilon_i$ such that $|\delta_i| \leq \epsilon$. This has a simple translation in terms of our original SOS problem. The equality constraints $\mathrm{tr}(A_i Q) = a_i$ correspond to equality between corresponding monomials of the polynomials $p$ and $z^T Q\, z$. As a result, there exists a matrix $E$ such that for all $i, j$, $|E_{i,j}| \leq \delta$, with

$$p = z^T (Q + E)\, z. \tag{3}$$

*Proof scheme (`soscheck_correct` in our Coq code).* Just take the matrix $E$ with coefficient

$$E_{i,j} := \frac{(p - z^T Q z)[z_i\, z_j]}{\#\{(i', j') \mid z_{i'}\, z_{j'} = z_i\, z_j\}}$$

where $p[m]$ stands for the coefficient of monomial $m$ in polynomial $p$ and $\#S$ denotes the cardinal of the set $S$. Then, each $(p - z^T Q z)[z_i\, z_j]$ corresponds to some $\delta_i$ and $\#\{(i', j') \mid z_{i'}\, z_{j'} = z_i\, z_j\} \geq 1$. □

It is worth noting that we cannot trust the value of $\delta$ reported by the solver, as it is just computed in floating-point arithmetic. However, it is easy to measure as $\delta := \max_i |\mathrm{tr}(A_i Q) - a_i|$.

#### 2.4.2 A Validation Method

Our goal is now to check that $Q + E \succeq 0$ which would prove, along with (3) that $p$ is SOS (and then non negative). To avoid an exact computation of $E$, we will rather attempt to prove the stronger result that for any matrix $M$ whose elements are bounded by $\delta$, $Q + M \succeq 0$. Figure 1 gives a geometrical intuition of this.

The following property then provides a sufficient condition.

**Property 1** (`posdef_check_itv_correct` in Coq)**.** *For any $Q \in \mathbb{R}^{s \times s}$ and $\delta \in \mathbb{R}$, if*

$$Q - s\delta I \succeq 0,$$

*then for any $E \in \{M \mid \forall i, j, |M_{i,j}| \leq \delta\}$,*

$$Q + E \succeq 0.$$

Thus, we are left having to prove that a given matrix $Q - s\delta I$ is positive semidefinite instead of $Q$ itself. We use a Cholesky decomposition to do so. Given a matrix $M$, if $M \succeq 0$, the Cholesky decomposition algorithm computes a matrix $R$ such that $M = R^T R$ which proves[4] that $M \succeq 0$. If $M$ is not positive semidefinite, the Cholesky decomposition fails by attempting to take the square root of a negative value. The execution of the algorithm requires $\Theta(s^3)$ arithmetic operations.

However, for the sake of efficiency, a floating point Cholesky decomposition is used, which prevents the exact equality $M = R^T R$. The following theorem gives an incomplete but efficient method that allows us to conclude the positive definiteness of a matrix $M$ using an unreliable floating point Cholesky decomposition of a slightly modified floating point matrix $\tilde{M}$.

**Theorem 1.** *(Rump 2006, Corollary 2.4) (`corollary_2_4_with_c_upper_bound` in Coq) Assume a floating-point format $\mathbb{F}$ with relative error $\varepsilon$ and absolute error $\eta$. For all $M \in \mathbb{R}^{s \times s}$, $\tilde{M} \in \mathbb{F}^{s \times s}$ such that $2(s+2)\varepsilon < 1$, If $M^T = M$, for all $i$, $M_{i,i} \geq 0$ and*

$$\forall i < j, \tilde{M}_{i,j} = M_{i,j}$$
$$\forall i, \tilde{M}_{i,i} \leq M_{i,i} - c$$

*wherein*
$c := \frac{(s+1)\varepsilon}{1-(2s+2)\varepsilon}\mathrm{tr}(M) + 4(s+1)\,(2(s+2) + \max_i M_{i,i})\,\eta$*, then if the floating-point Cholesky decomposition of $\tilde{M}$ succeeds we can also conclude that $M \succeq 0$.*

**Remark 3.** *$\varepsilon$ and $\eta$ are constants characterizing the errors due to normalized and denormalized numbers in the floating-point format $\mathbb{F}$. For instance, for the IEEE754 (IEEE Computer Society 2008) binary64 format with rounding to nearest[5], $\varepsilon = 2^{-53}$ ($\simeq 10^{-16}$) and $\eta = 2^{-1075}$ ($\simeq 10^{-323}$). Thus, the hypothesis $2(s+2)\varepsilon < 1$ is always satisfied for practical values of $s$. Moreover, for typical values ($s \leq 1000$ and elements of $M$ of order of magnitude 1), $c \leq 10^{-10}$. This is negligible in front of $s\delta \sim 1000 \times 10^{-8} = 10^{-5}$ which means that the incompleteness of this positive-definiteness check is not an issue in practice.*

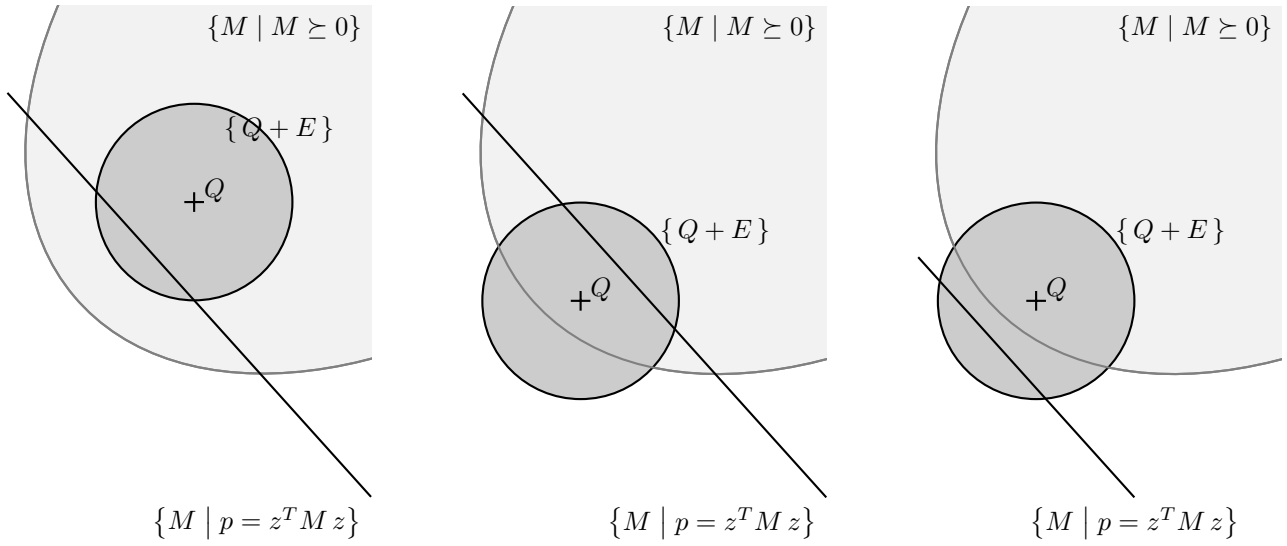
Our matrices $Q - s\delta I$ are symmetric so the hypothesis $M^T = M$ is always satisfied. Similarly, the hypothesis that $M$ has a non negative diagonal is always true in practice, otherwise $M$ would trivially fail to be positive semidefinite. So according to this theorem, the positive definiteness of $Q - s\delta I$ can be established by first checking that its diagonal is non negative, then computing an upper bound[6] of the

[3] Typically, $\delta \sim 10^{-8}$.

[4] For any $x$, $x^T M\, x = (Rx)^T (Rx) = \|Rx\|_2^2 \geq 0$

[5] Type `double` in C.

[6] For instance using floating-point arithmetic with directed rounding.




(a) $p$ is proved SOS. (b) Cannot conclude. (c) Cannot conclude.

**Figure 1.** To prove that $p$ is SOS, we need to prove that there exists a matrix $M \succeq 0$ such that $p = z^T M\, z$, that is to prove that the subspace $\{M \mid p = z^T M\, z\}$ intersects the positive semidefinite cone $\{M \mid M \succeq 0\}$. The SDP solver returns a matrix $Q$ close from the subspace, i.e., such that the ball $\{Q + E \mid \forall i, j, |E_{i,j}| \leq \delta\}$ (denoted $\{\, Q + E \,\}$ on the figure) intersects it. Thus, proving that this ball is included in the cone, as on (a), enables to conclude. The proof can also fail, either because $Q$ is too close from the border of the cone (b) or because $p$ is simply not SOS (c).

constant $c$, subtracting it to the diagonal of our matrix and applying the floating-point Cholesky decomposition.

**Remark 4.** *The proof of Theorem 1 was already formalized in Coq in a previous work (Roux 2016). Unfortunately, this specification was far from an executable program. One contribution of the present paper is to enable this algorithm to actually run inside Coq as detailed in the next sections.*

To sum up, we designed the following verification method to prove that a given polynomial $p$ is SOS. Given the approximate solution $Q$ returned by an SDP solver for the problem $p = z^T Q\, z, Q \succeq 0$:

1. Check that all monomials of the polynomial $p$ are in the monomial base $z^T z$.
2. Bound the difference $\delta$ between the corresponding coefficients of $p$ and $z^T Q\, z$.
3. Check that $Q - s\delta I \succeq 0$.

Step 1 is a purely symbolic computation, step 2 can be achieved using floating-point interval arithmetic[7] in $\Theta(s^2)$ operations (the size of $Q$) and step 3 can be done in $\Theta(s^3)$ floating-point operations thanks to the above theorem. Thus, the whole validation method takes $\Theta(s^3)$ floating-point operations which in practice constitutes a very small overhead compared to the time taken by the SDP solver to compute $Q$.

[7] Although we currently use rational arithmetic for this step in our Coq development, as it appeared cheap enough thanks to the reasonably small amount of computation involved.

**Remark 5** (Incompleteness of the method)**.** *Of course, this method is incomplete (c.f., Figure 1) and may fail to provide a proof even when a polynomial is actually SOS. The reader interested in sources of failure (optimization algorithms used by SDP solvers, failure of strict feasibility, bad conditioning and floating-point computations in the SDP solver) and means to mitigate them is referred to a previous work (Roux et al. 2016).*

### 2.5 Coq Proofs

The Coq proofs of the results presented in this section rely on the Mathcomp library (Gonthier et al. 2008) for matrices, on the Flocq library (Boldo and Melquiond 2011) for the formalization of floating-point arithmetic and on SsrMultinomials (Bernard et al. 2016) for multivariate polynomials.

The proofs related to the Cholesky decomposition (step 3) are borrowed from (Roux 2016) and located in the files `cholesky.v` and `cholesky_infnan.v` whereas the other proofs (steps 1 and 2) are new and can be found in the file `validsdp.v`.

## 3. Verification of Effective Computation using Data Refinement

The Coq proofs mentioned in the previous section deal with an abstract version of the algorithms, which are not suitable for computation. For instance, Mathcomp's matrices have been specifically designed to ease reasoning on them, but one cannot compute with them as most functions about matrices

do not permit evaluation. So we rely on the CoqEAL library (Cohen et al. 2013) which has been devised to facilitate the design and the proof of effective computation, notably for linear algebra. We thus follow the following methodology that is typical when using CoqEAL:

- We implement the algorithms in a general way (using polymorphic functions and Type Classes).
- We specialize the algorithms with proof-oriented datatypes and prove these algorithms correct (this relates to Section 2.5). This step often requires doing some "program refinement", i.e., proving that the considered algorithm is extensionally equal to a simpler algorithm.
- We specialize the algorithms with effective datatypes and prove them correct with respect to the proof-oriented datatypes. This amounts to doing some "data refinement".

We elaborate on the last step in the rest of this section.

To obtain an effective version of our verification algorithm, we need to compute with rational numbers, floating-point numbers, matrices, vectors of monomials and multivariate polynomials. To this aim, we rely on the following Coq libraries : $(i)$ The BigQ standard library for efficient rational numbers (based on machine integers); $(ii)$ The floating-point kernel of CoqInterval (Martin-Dorel and Melquiond 2016) for emulating Binary64 floating-point numbers[8]; $(iii)$ The CoqEAL library for effective matrices based on lists of lists; $(iv)$ Lists of binary integers (`seq N`) for vectors of monomials; $(v)$ The FMapAVL standard library for efficient maps (based on AVL trees).

Relying on these datatypes, a major contribution of our work was to formalize effective multivariate polynomials using FMaps, and prove them correct with respect to Ssr-Multinomials, by taking advantage of CoqEAL's framework. This formalization can be found in the file `multipoly.v`.

We also needed to refine proof-oriented rational numbers (of type `rat`, which is endowed with a `realFieldType` Mathcomp algebraic structure) to effective rationals, and provide functions to go back and forth between rationals and floating-point numbers. The diagrams in Figure 2 summarize the two main properties that are involved in our formal development. This formalization also led us to develop a theory of the unit-in-the-last-place for integers in radix 2 (i.e., the largest power of 2 that divides a big integer).

# 4. An Automated Tactic for Verifying Positivity Witnesses

## 4.1 Calling SDP Solvers Through an OCaml Module

The main ingredients of the positivity proofs using SOS polynomials, as described in Section 2, are the vector of monomial $z$ and the matrix $Q$ computed by the numerical SDP solvers.

[8] Without overflows, although the proofs of Section 2.5 could handle them.

We developed an OCaml library called OSDP[9] that handles the work of translating SOS problems to SDP, calling an SDP solver[10], retrieving the result and performing the verification method of Section 2. The library is composed of 6.2 kloc of OCaml and 1.2 kloc of C code. A Coq module written in OCaml then uses this library. This module[11] only requires 0.3 kloc of OCaml to read a polynomial from Coq, call OSDP and translate the resulting $z$ and $Q$ back to Coq terms. It is worth noting that all this OCaml code and the underlying SDP solvers are only used as untrusted oracles, just making the main proof ingredients available to the formally verified Coq tactic discussed in the remainder of this section. The main advantage of this skeptical approach is to enable the use of any off-the-shelf solver and easy implementation of arbitrary optimizations in the SOS to SDP translation, without any risk of jeopardizing the eventual proof soundness.

## 4.2 Verification of the Witness

The verification of witnesses is performed by a computational Boolean function `soscheck_eff_wrapup` whose type is as follows:

```
seq R → p_abstr_poly →
seq(seq N) * seq(seq(s_float bigZ bigZ)) → bool
```

The first two arguments (the list of real variables involved in the user's goal and the abstract syntax tree corresponding to the polynomial under study) will be obtained after the reification of the user's goal (c.f., the upcoming section). The third argument is the witness `(z, Q)` obtained after calling the external SDP solver.

It can be noted that we relied on CoqEAL's approach for data refinement (see Section 3) on every building block involved in the definition of `soscheck_eff_wrapup`.

We then prove the main correctness claim associated to this function by relying on the previously presented material:

```
Theorem soscheck_eff_wrapup_correct :
∀(vm : seq R) (pap : p_abstr_poly)
(zQ : seq(seq N) * seq(seq(s_float bigZ bigZ))),
  soscheck_eff_wrapup vm pap zQ = true →
  (0 ≤ interp_p_abstr_poly vm pap)%R.
```

## 4.3 Overview of the Tactic

Finally, we developed some dedicated Ltac code to reify the user's goal and obtain an element of the `p_abstr_poly` abstract syntax tree. This syntax involves rational constants (defined as a separated inductive `p_real_cst` built from non negative integers, opposites and divisions thereof), real variables, and the usual polynomial operations including exponentiation (with positive, constant exponents). As usual with a Coq reflexive tactic, the Ltac code then applies the

[9] Available at `http://cavale.enseeiht.fr/osdp/`.

[10] The library interfaces the SDP solvers CSDP (Borchers 1999), SDPA, SDPA-GMP and SDPA-DD (Yamashita et al. 2010) and MOSEK (MOS 2015).

[11] File `soswitness.ml4` in our development.

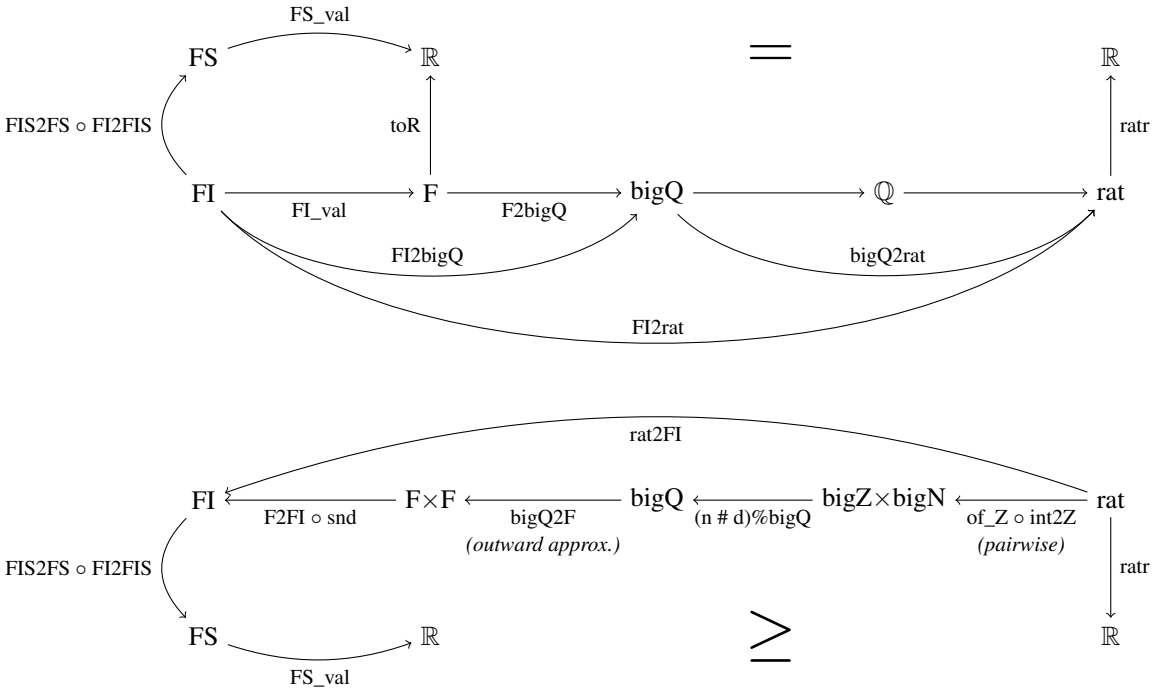



**Figure 2.** Functions relating rationals, floating-point and real numbers. The type rat is defined in MathComp and represents rational numbers using Peano integers. The type $\mathbb{Q}$ is defined in the module QArith of Coq's standard library and represents rational numbers using binary integers from ZArith. The types bigN, bigZ and bigQ are defined in the Coq standard library for representing natural/integer/rational numbers with an efficient representation (based on machine integers). The type F is defined in CoqInterval and represents floating-point number with unbounded mantissa and exponent. The type FI is a record containing an element of F and a proof that it is representable in 53 bits. The type FS is a record gathering a real number and a proof that it it is representable in 53 bits (relying on the radix-2 precision-53 FLX format of Flocq).

correctness claim `soscheck_eff_wrapup_correct` to the user's goal and discharges the obtained subgoals by computation.

As a result, our `validsdp` tactic recognizes goals that are inequalities between polynomials expressions, involving atomic real variables as well as rational constants.

The complete development is available at `https://sourcesup.renater.fr/validsdp/`. It amounts to 13.6 kloc, that can be decomposed as 5.1 kloc for the proof of the Cholesky decomposition algorithm (borrowed from (Roux 2016)), 3.0 kloc for the refinement of the algorithm to an executable program (relying on an arithmetic parameterized with Type Classes), 1.3 kloc for the refinement of CoqInterval floating-point arithmetic, 2.2 kloc for the refinement of multivariate polynomials and finally 2.0 kloc for the main tactic and the related correctness proofs.

## 5. Benchmarks

We evaluated the performances of our tactic with respect to the related works mentioned in Section 1.1 on a set of benchmarks.

These benchmarks are composed of three subsets:

- A set of global optimization problems consisting in proving bounds for polynomials on given hyperboxes. These problems are taken from (Muñoz and Narkawicz 2013).
- A set of problems on 6 variable polynomials coming from some inequalities of the Flyspeck project. Most of them require to prove that a given polynomial is positive on a given hyperbox. Some require to prove that at least one of two or three polynomials is positive on any point of a given hyperbox (two polynomials in `fs868`, `fs884` and `fs890`, three polynomials in `fs260` and `fs491`).
- A set of problems on loop invariants of programs (Adjé et al. 2015; Roux et al. 2016). They require to prove that a given polynomial $p$ is non negative when some initial conditions are met and that whenever $p$ is non negative, then $p \circ f$ also is, for a given polynomial $f$ (representing the loop body assignments).

Note that some benchmarks are known to be false[12] (viz. `ex4_d4`, `ex4_d6` and `ex7_d4`), so the absence of proof is the expected result. Others are known to be correct[13] (`fs884`,

[12] Counter examples can easily be found.

[13] Another SDP solver and/or some symbolic preprocessing enables the proof to go through with OSDP.

| Problem | $n$ | $d$ | OSDP (not verified) | ValidSDP | PVS/Bernstein | Monniaux and Corbineau 11 (not verified) | NLCertify (not verified) | NLCertify | HOL Light/ Taylor$^{(\star)}$ |
|---|---|---|---|---|---|---|---|---|---|
| adaptativeLV | 4 | 4 | 0.70 | 8.73 | 14.93 | 2.67 | 1.12 | 2.61 | 12.31 |
| butcher | 6 | 4 | 0.84 | 21.14 | 48.44 | — | 1.05 | 8.36 | 15.62 |
| caprasse | 4 | 4 | 0.47 | 8.91 | 25.89 | 1.82 | 0.88 | 2.63 | 17.68 |
| heart | 8 | 4 | 0.99 | 37.99 | 131.13 | 268.75 | — | — | 26.15 |
| magnetism | 7 | 2 | 0.24 | 4.68 | 245.52 | 2.04 | 1.64 | 14.50 | 16.07 |
| reaction | 3 | 2 | 1.02 | 4.05 | 11.48 | 1.56 | 0.24 | 1.96 | 12.41 |
| schwefel | 3 | 4 | 0.88 | 5.67 | 14.72 | 2.45 | 2.76 | 56.13 | 17.46 |
| fs260 | 6 | 4 | 0.26 | 12.32 | — | — | — | — | — |
| fs461 | 6 | 4 | 0.26 | 11.24 | 621.06 | 11.18 | 0.87 | 7.46 | 22.70 |
| fs491 | 6 | 4 | 1.02 | 14.93 | — | 21.81 | — | — | — |
| fs745 | 6 | 4 | 0.29 | 12.20 | 623.17 | 11.74 | 0.94 | 6.90 | 22.48 |
| fs752 | 6 | 2 | 0.43 | 3.88 | 54.52 | 1.81 | 0.90 | 7.88 | 13.34 |
| fs859 | 6 | 8 | — | — | — | — | — | — | — |
| fs860 | 6 | 4 | 0.97 | 11.09 | 73.65 | 10.53 | 1.11 | 7.34 | 14.28 |
| fs861 | 6 | 4 | 0.25 | 11.13 | 69.74 | 10.48 | 1.20 | 7.87 | 14.28 |
| fs862 | 6 | 4 | 0.79 | 10.81 | 73.54 | 79.25 | 1.25 | 7.58 | 14.14 |
| fs863 | 6 | 2 | — | — | — | 1.50 | — | — | 13.85 |
| fs864 | 6 | 2 | 0.97 | 4.50 | — | 2.05 | — | — | 13.28 |
| fs865 | 6 | 2 | 0.93 | 4.56 | — | 2.11 | — | — | 13.76 |
| fs867 | 6 | 2 | 0.47 | 3.90 | — | 2.09 | 1.74 | 8.04 | — |
| fs868 | 6 | 4 | 0.72 | 12.65 | — | — | — | — | — |
| fs884 | 6 | 4 | — | — | — | — | — | — | — |
| fs890 | 6 | 4 | — | — | — | 7.78 | — | — | — |
| fs8 | 6 | 2 | 0.65 | 4.58 | 52.63 | 1.53 | 1.48 | 6.62 | 13.40 |
| ex4_d4 | 2 | 12 | — | — | — | — | — | — | — |
| ex4_d6 | 2 | 18 | — | — | — | — | — | — | — |
| ex4_d8 | 2 | 24 | — | — | — | — | — | — | — |
| ex4_d10 | 2 | 30 | — | — | — | — | — | — | — |
| ex5_d4 | 3 | 8 | 0.73 | 22.67 | — | — | — | — | — |
| ex5_d6 | 3 | 12 | 3.41 | 85.34 | — | — | — | — | — |
| ex5_d8 | 3 | 16 | 20.54 | 324.29 | — | — | — | — | — |
| ex5_d10 | 3 | 20 | 150.86 | — | — | — | — | — | — |
| ex6_d4 | 4 | 8 | 2.44 | 57.97 | — | — | — | — | — |
| ex6_d6 | 4 | 12 | 56.19 | 502.17 | — | — | — | — | — |
| ex7_d4 | 2 | 12 | — | — | — | — | — | — | — |
| ex7_d6 | 2 | 18 | 0.84 | 43.87 | — | — | — | — | — |
| ex7_d8 | 2 | 24 | — | — | — | — | — | — | — |
| ex7_d10 | 2 | 30 | — | — | — | — | — | — | — |
| ex8_d4 | 2 | 8 | 0.13 | 10.53 | — | 15.72 | — | — | — |
| ex8_d6 | 2 | 12 | — | — | — | — | — | — | — |
| ex8_d8 | 2 | 16 | — | — | — | — | — | — | — |
| ex8_d10 | 2 | 20 | — | — | — | — | — | — | — |

**Table 1.** Running time (elapsed real time, in s) for various tools on a set of benchmarks. "—" indicates either that a tool is not applicable or that it failed to produce a proof within the time limit (900 s). $n$ is the number of variables of the polynomials and $d$ is their degree. "OSDP" is our OCaml implementation of our proof procedure and "ValidSDP" our Coq tactic, "PVS/Bernstein" corresponds to (Muñoz and Narkawicz 2013), "Monniaux and Corbineau 11" corresponds to (Monniaux and Corbineau 2011), "NLCertify" corresponds to (Magron 2014), and "HOL Light/Taylor" corresponds to (Solovyev and Hales 2013).
$^{(\star)}$Remark: it should be noted that each running time in the last column includes the time (around 11s) for loading the image of the HOL Light libraries, checkpointed beforehand using DMTCP.

ex4_d8 and ex7_d8). Finally, the status of the remaining unproved benchmarks is unknown (fs859, ex4_d10, ex7_d10, ex8_6, ex8_d8 and ex8_d10).

Table 1 presents the results on a Core i5-4460S CPU clocked at 2.9 GHz. Unfortunately, some of the tools (PVS/Bernstein, NLCertify and HOL Light/Taylor) were not applicable on part of the benchmarks as they require an hyperbox. It is also worth noting that for methods based on SDP solvers (i.e., all methods but PVS/Bernstein and HOL Light/Taylor), its choice can have a slight influence on the results. As a matter of fact, NLCertify uses SDPA. Monniaux and Corbineau 2011 can use either CSDP, DSDP or SDPA and we chose CSDP as it gave the best results. Finally, OSDP (hence ValidSDP) can use either CSDP, Mosek or SDPA. Although Mosek tends to give the best results on large problems and is about ten times faster, we chose SDPA for the sake of fairness of the comparison.

These results indicate that while our tactic, based on floating-point arithmetic, is competitive with other tools on the simplest benchmarks it, more noticeably, enables to perform proofs that seem out of reach for other methods, mostly based on rational arithmetic.

The OSDP column in the table is included to give some insight on what is lost, in terms of performance, by running the verification procedure inside the Coq virtual machine (ValidSDP column). Indeed, most of the time in the OSDP column is spent running SDP solvers. The table then indicates that our Coq tactic commonly incurs a factor ten to fifty overhead. This is somewhat disappointing as the verification of results from SDP solvers should ideally imply a negligible overhead. However, most of this overhead comes from the emulation of floating-point arithmetic that is typically three orders of magnitude slower than native operations performed by the hardware floating-point unit. This means that a native implementation of floating-point arithmetic in the Coq kernel, as done with 31 bit integers (Armand et al. 2010), could yield a speedup of our tactic of one or two orders of magnitude. It is also worth noting that the memory footprint of the Coq implementation appears to be about ten times the memory footprint of the OCaml implementation.

Finally, one can notice that all computations in our tactic are performed with the vm_compute instead of the more recent native_compute mechanism (Boespflug et al. 2011). We did experiment with this mechanism[14]. Unfortunately, the large terms, corresponding to the $Q$ matrices, produced by our OCaml module led to huge OCaml source codes[15] whose compilation time made native_compute a few times slower than vm_compute. This illustrates that the Coq system may lack a way to load (untrusted) proof elements other than hard coding them in the sources.

[14] https://coq.inria.fr/bugs/show_bug.cgi?id=4714

[15] For instance, 10MB, 100 kloc OCaml source code for medium size benchmarks.

## 6. Conclusion and Future Work

We developed a reflexive Coq tactic for proving multivariate polynomials positivity. This tactic, relying on intensive floating-point computations, demonstrates that performing rigorous proofs inside a proof assistant using floating-point implementations of non trivial algorithms is tractable. The fact that our tactic is able to discharge proof obligations that seem untractable with other state of the art methods even indicates that such proof methods can be profitable. This, more generally, opens to proof assistants a wide range of rigorous proofs based on floating-point computations (Rump 2010).

Our experiments also indicate that this kind of proof methodology would greatly benefit from having native floating-point operations available in the proof assistant, rather than having to emulate them.

## References


A. Adjé, P. Garoche, and V. Magron. Property-based polynomial invariant generation using sums-of-squares optimization. In S. Blazy and T. Jensen, editors, *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, volume 9291 of *LNCS*, pages 235–251. Springer, 2015. ISBN 978-3-662-48287-2. doi: 10.1007/978-3-662-48288-9_14. URL http://dx.doi.org/10.1007/978-3-662-48288-9_14.

M. Armand, B. Grégoire, A. Spiwack, and L. Théry. Extending Coq with imperative features and its application to SAT verification. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2010. ISBN 978-3-642-14051-8. doi: 10.1007/978-3-642-14052-5_8. URL http://dx.doi.org/10.1007/978-3-642-14052-5_8.

S. J. Benson and Y. Ye. Algorithm 875: DSDP5 - software for semidefinite programming. *ACM Trans. Math. Softw.*, 34(3), 2008. URL http://doi.acm.org/10.1145/1356052.1356057.

S. Bernard, Y. Bertot, L. Rideau, and P. Strub. Formal proofs of transcendence for e and pi as an application of multivariate and symmetric polynomials. In J. Avigad and A. Chlipala, editors, *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*, pages 76–87. ACM, 2016. ISBN 978-1-4503-4127-1. doi: 10.1145/2854065.2854072. URL http://doi.acm.org/10.1145/2854065.2854072.

F. Besson. Fast reflexive arithmetic tactics the linear case and beyond. In T. Altenkirch and C. McBride, editors, *Types for Proofs and Programs – TYPES 2006, Nottingham, UK, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2006. ISBN 978-3-540-74463-4. URL http://dx.doi.org/10.1007/978-3-540-74464-1_4.

M. Boespflug, M. Dénès, and B. Grégoire. Full reduction at full throttle. In J. Jouannaud and Z. Shao, editors, *Certified Programs and Proofs - First International Conference, CPP*

*2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, volume 7086 of *Lecture Notes in Computer Science*, pages 362–377. Springer, 2011. ISBN 978-3-642-25378-2. doi: 10.1007/978-3-642-25379-9_26. URL http://dx.doi.org/10.1007/978-3-642-25379-9_26.

S. Boldo and G. Melquiond. Flocq: A Unified Library for Proving Floating-point Algorithms in Coq. In *Proceedings of the 20th IEEE Symposium on Computer Arithmetic*, pages 243–252, Tübingen, Germany, July 2011.

B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1-4), 1999. URL http://dx.doi.org/10.1080/10556789908805765.

S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

C. Cohen, M. Dénès, and A. Mörtberg. Refinements for free! In G. Gonthier and M. Norrish, editors, *Certified Programs and Proofs*, volume 8307 of *LNCS*, pages 147–162. Springer, 2013. ISBN 978-3-319-03544-4. URL http://dx.doi.org/10.1007/978-3-319-03545-1_10.

G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, 1975.

*The Coq proof assistant reference manual*. The Coq development team, 2016. URL https://coq.inria.fr. Version 8.5.

G. Gonthier, A. Mahboubi, and E. Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA, 2008. URL http://hal.inria.fr/inria-00258384.

J. Harrison. Verifying nonlinear real formulas via sums of squares. In *TPHOLs 2007*, volume 4732 of *LNCS*, pages 102–118. Springer, 2007.

IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*, 2008.

J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001. doi: 10.1137/S1052623400366802. URL http://dx.doi.org/10.1137/S1052623400366802.

J.-B. Lasserre. *Moments, Positive Polynomials, and Their Applications. 2009*. Imperial College Press, 2009. ISBN 978-1-84816-445-1.

A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. ISSN 1432-1807. doi: 10.1007/BF01457454. URL http://dx.doi.org/10.1007/BF01457454.

J. Löfberg. Pre- and post-processing sum-of-squares programs in practice. *IEEE Transactions on Automatic Control*, 54(5):1007–1011, 2009. URL http://ieeexplore.ieee.org/iel5/9/4919221/04908937.pdf?arnumber=4908937.

V. Magron. NLCertify: A Tool for Formal Nonlinear Optimization. In *Mathematical Software – ICMS 2014*, volume 8592 of *LNCS*, pages 315–320. Springer, 2014. URL http://dx.doi.org/10.1007/978-3-662-44199-2_49.

E. Martin-Dorel and G. Melquiond. Proving tight bounds on univariate expressions with elementary functions in Coq. *Journal of Automated Reasoning*, 57(3):187–217, Oct. 2016. URL http://dx.doi.org/10.1007/s10817-015-9350-4.

D. Monniaux and P. Corbineau. On the Generation of Positivstellensatz Witnesses in Degenerate Cases. In *Interactive Theorem Proving – ITP 2011, Berg en Dal, The Netherlands*, pages 249–264, 2011. URL http://dx.doi.org/10.1007/978-3-642-22863-6_19.

*The MOSEK C optimizer API manual Version 7.1 (Revision 40)*. MOSEK ApS, 2015. URL http://docs.mosek.com/7.1/capi/index.html.

C. Muñoz and A. Narkawicz. Formalization of Bernstein polynomials and applications to global optimization. *J. Autom. Reasoning*, 51(2):151–196, 2013. URL http://dx.doi.org/10.1007/s10817-012-9256-3.

A. Narkawicz and C. Muñoz. A formally verified generic branching algorithm for global optimization. In *International Conference on Verified Software: Theories, Tools, Experiments*, volume 8164 of *LNCS*, pages 326–343, 2013. URL http://dx.doi.org/10.1007/978-3-642-54108-7_17.

P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Math. Program.*, 96(2):293–320, 2003. doi: 10.1007/s10107-003-0387-5. URL http://dx.doi.org/10.1007/s10107-003-0387-5.

P. Roux. Formal proofs of rounding error bounds - with application to an automatic positive definiteness check. *J. Autom. Reasoning*, 57(2):135–156, 2016. doi: 10.1007/s10817-015-9339-z. URL http://dx.doi.org/10.1007/s10817-015-9339-z.

P. Roux, Y. Voronin, and S. Sankaranarayanan. Validating numerical semidefinite programming solvers for polynomial invariants. In X. Rival, editor, *Static Analysis - 23rd International Symposium, SAS 2016, Edinburgh, UK, September 8-10, 2016, Proceedings*, volume 9837 of *LNCS*, pages 424–446. Springer, 2016. ISBN 978-3-662-53412-0. doi: 10.1007/978-3-662-53413-7_21. URL http://dx.doi.org/10.1007/978-3-662-53413-7_21.

S. M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 46, 2006.

S. M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19, 2010.

A. Solovyev and T. C. Hales. Formal verification of nonlinear inequalities with Taylor interval approximations. In *NASA Formal Methods*, volume 7871 of *LNCS*, pages 383–397, 2013. URL http://dx.doi.org/10.1007/978-3-642-38088-4_26.

A. Tarski. A decision method for elementary algebra and geometry. Technical report, Univ. of California Press, Berkeley, 1951.

L. Vandenberghe and S. P. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996. doi: 10.1137/1038003. URL http://epubs.siam.org/doi/abs/10.1137/1038003.

M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs: Sdpa 7. Technical Report B-460, Tokyo Institute of Technology, Tokyo, 2010.