

Certified Roundoff Error Bounds Using Semidefinite Programming

Abstract

Roundoff errors cannot be avoided when implementing numerical programs with finite precision. The ability to reason about rounding is especially important if one wants to explore a range of potential representations, for instance in the world of FPGAs. This problem becomes challenging when the program does not employ solely linear operations as nonlinearities are inherent to many interesting computational problems in real-world applications.

Existing solutions to reasoning are limited in presence of nonlinear correlations between variables, leading to either imprecise bounds or high analysis time. Furthermore, while it is easy to implement a straightforward method such as interval arithmetic, sophisticated techniques are less straightforward to implement in a formal setting. Thus there is a need for methods which output certificates that can be formally validated inside a proof assistant.

We present a framework to provide upper bounds of absolute roundoff errors. This framework is based on optimization techniques employing semidefinite programming and sums of squares certificates, which can be formally checked inside the Coq theorem prover. Our tool covers a wide range of nonlinear programs, including polynomials and transcendental operations as well as conditional statements. We illustrate the efficiency and precision of this tool on non-trivial programs coming from biology, optimization and space control.

Categories and Subject Descriptors D.2.4 [Software Engineering]: Program Verification

Keywords hardware precision tuning; roundoff error; numerical accuracy; floating-point arithmetic; fixed-precision arithmetic; semidefinite programming; sums of squares; correlation sparsity pattern; proof assistant; formal verification.

1. Introduction

Constructing numerical programs which perform accurate computation turns out to be difficult, due to finite numerical precision of implementations such as floating-point or fixed-point representations. Finite-precision numbers induce roundoff errors, and knowledge of the range of these roundoff errors is required to fulfill safety criteria of critical programs, as typically arising in modern embedded systems such as aircraft controllers. Such a knowledge can be used in general for developing accurate numerical software, but appears to be particularly relevant while considering algorithms migration onto reconfigurable hardware (e.g. FPGAs). The advantage of architectures based on FPGAs is that they allow more flexible choices, rather than choosing either for IEEE standard single or double precision. Indeed, in this case, we

benefit from a more flexible number representation while ensuring guaranteed bounds on the program output.

To obtain lower bounds over roundoff errors, one can rely on testing approaches, such as meta-heuristic search [16] or under-approximation tools (e.g. `s3fp` [20]). Here, we are interested in handling efficiently the complementary over-approximation problem, namely to obtain precise upper bounds over the error. This problem boils down to finding tight abstractions of linearities or non-linearities while being able to bound the resulting approximations in an efficient way. For computer programs consisting of linear operations, automatic error analysis can be obtained with well-studied optimization techniques based on SAT/SMT solvers [34], affine arithmetic [26]. However, non-linear operations are key to many interesting computational problems arising in physics, biology, controller implementations and global optimization. Recently, two promising frameworks have been designed to provide upper bounds for roundoff errors of nonlinear programs. The corresponding algorithms rely on Taylor-interval methods [62], implemented in the `FPTaylor` tool, and on combining SMT with affine arithmetic [23], implemented in the `Rosa` real compiler.

The common drawback of these two frameworks is that they do not fully take into account the correlations between program variables. Thus they may output coarse error bounds or perform analysis within a large amount of time.

While the `Rosa` tool is based on theoretical results that should provide sounds over-approximations of error bounds, the tool does not produce formal proof certificates to allow independent soundness checking. The complexity of the mathematics underlying techniques for nonlinear reasoning, and the intricacies associated with constructing an efficient implementation, are such that a means for independent formal validation of results is particularly desirable. To the best of our knowledge, the `FPTaylor` software is the only academic tool which can produce formal proof certificates. This is based on the framework developed in [61] to verify nonlinear inequalities in `HOL-LIGHT` [36] using Taylor-interval methods. However, most of computation performed in the informal optimization procedure end up being redone inside the `HOL-LIGHT` proof assistant, yielding a formal verification which is computationally demanding.

The aim of this work is to provide a formal framework to perform automated precision analysis of computer programs that manipulate finite-precision data using nonlinear operators. For such programs, guarantees can be provided with certified programming techniques. Semidefinite programming (SDP) is relevant to a wide range of mathematical fields, including combinatorial optimization, control theory and matrix completion. In 2001, Lasserre introduced a hierarchy of SDP relaxations [42] for approximating polynomial infima. Our method to bound the error is a decision

procedure based on an specialized variant of Lasserre hierarchy [43]. The procedure relies on SDP to provide sparse sum-of-squares decompositions of nonnegative polynomials. Our framework handles polynomial program analysis (involving the operations $+$, \times , $-$) as well as extensions to the more general class of semialgebraic and transcendental programs (involving $\sqrt{\cdot}$, \min , \max , \arctan , \exp), following the approximation scheme described in [49].

Overview of our Method We present an overview of our method and of the capabilities of related techniques, using an example. Consider a program implementing the following polynomial expression f :

$$f(\mathbf{x}) := x_2 \times x_5 + x_3 \times x_6 - x_2 \times x_3 - x_5 \times x_6 \\ + x_1 \times (-x_1 + x_2 + x_3 - x_4 + x_5 + x_6),$$

where the six-variable vector $\mathbf{x} := (x_1, x_2, x_3, x_4, x_5, x_6)$ is the input of the program. For this example, the set \mathbf{X} of possible input values is a product of closed intervals: $\mathbf{X} = [4.00, 6.36]^6$. This function f together with the set X appear in many inequalities arising from the the proof of Kepler Conjecture [32], yielding challenging global optimization problems.

The polynomial expression f is obtained by performing 15 basic operations (1 negation, 3 subtractions, 6 additions and 5 multiplications). When executing this program with a set of floating-point numbers $\hat{\mathbf{x}} := (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6) \in \mathbf{X}$, one actually computes a floating-point result \hat{f} , where all operations $+$, $-$, \times are replaced by the respectively associated floating-point operations \oplus , \ominus , \otimes . The results of these operations comply with IEEE 754 standard arithmetic [3] (see relevant background in Section 2.1). For instance, one can write $\hat{x}_2 \otimes \hat{x}_5 = (x_2 \times x_5)(1 + e_1)$, by introducing an error variable e_1 such that $-\epsilon \leq e_1 \leq \epsilon$, where the bound ϵ is the machine precision (e.g. $\epsilon = 2^{-24}$ for single precision). One would like to bound the absolute roundoff error $|r(\mathbf{x}, \mathbf{e})| := |\hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})|$ over all possible input variables $\mathbf{x} \in \mathbf{X}$ and error variables $e_1, \dots, e_{15} \in [-\epsilon, \epsilon]$. Let us define $\mathbf{E} := [-\epsilon, \epsilon]^{15}$ and $\mathbf{K} := \mathbf{X} \times \mathbf{E}$, then our bound problem can be cast as finding the maximum r^* of $|r|$ over \mathbf{K} , yielding the following nonlinear optimization problem:

$$r^* := \max_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} |r(\mathbf{x}, \mathbf{e})| \\ = \max\{-\min_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} r(\mathbf{x}, \mathbf{e}), \max_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} r(\mathbf{x}, \mathbf{e})\}, \quad (1)$$

One can directly try to solve these two polynomial optimization problems using classical SDP relaxations [42]. As in [62], one can also decompose the error term r as the sum of a term $l(\mathbf{x}, \mathbf{e})$, which is affine w.r.t. \mathbf{e} , and a nonlinear term $h(\mathbf{x}, \mathbf{e}) := r(\mathbf{x}, \mathbf{e}) - l(\mathbf{x}, \mathbf{e})$. Then the triangular inequality yields:

$$r^* \leq \max_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} |l(\mathbf{x}, \mathbf{e})| + \max_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} |h(\mathbf{x}, \mathbf{e})|. \quad (2)$$

It follows for this example that $l(\mathbf{x}, \mathbf{e}) = x_2x_5e_1 + x_3x_6e_2 + (x_2x_5 + x_3x_6)e_3 + \dots + f(\mathbf{x})e_{15} = \sum_{i=1}^{15} s_i(\mathbf{x})e_i$. The *Symbolic Taylor expansions* method [62] consists of using Taylor-interval optimization to compute a rigorous interval enclosure of each polynomial s_i , $i = 1, \dots, 15$, over \mathbf{X} and finally obtain an upper bound of $|l| + |h|$ over \mathbf{K} . Our method uses sparse semidefinite relaxations for polynomial optimization (derived from [43]) to bound l as well as basic interval arithmetic to bound h . The following results have been obtained on an Intel Core i5 CPU (2.40 GHz). All execution times have been computed by averaging over five runs.

- A direct attempt to solve the two polynomial problems occurring in Equation (1) fails as the SDP solver (in our case SDPA [67]) runs out of memory.
- Using our method implemented in the `Real2Float` tool, one obtains an upper bound of 788ϵ for $|l| + |h|$ over \mathbf{K} in less than one second. This bound is provided together with a certificate which can be formally checked inside the COQ proof assistant in 0.34 seconds.
- Using basic interval arithmetic, one obtains 17.1 times more quickly a coarser bound of 2023ϵ .
- Symbolic Taylor expansions implemented in `FPTaylor` [62] provide an intermediate bound of 937ϵ but 16.3 times slower than with our implementation. Formal verification of this bound inside the HOL-LIGHT proof assistant takes 73.8 seconds and is 217 times slower than proof checking with `Real2Float` inside COQ.
- Finally, our bound is also obtained with the `Rosa` real compiler [23] but 4.6 times slower than with our implementation.

Contributions Our key contributions can be summarized as follows:

- We present an optimization algorithm providing certified over-approximations for roundoff errors of nonlinear programs. This algorithm is based on sparse sums of squares programming [43]. By comparison with other methods, our algorithm allows us to obtain tighter upper bounds, while overcoming scalability and numerical issues inherent in SDP solvers [64]. Our algorithm can currently handle programs implementing polynomial functions, but also involving non-polynomial components, including either semialgebraic or transcendental operations (e.g. $/$, $\sqrt{\cdot}$, \arctan , \exp), as well as conditional statements. Programs containing iterative or while loops are not currently supported.
- Our framework is fully implemented in the `Real2Float` tool. Among several features, the tool can optionally perform formal verification of roundoff error bounds for polynomial programs, inside the COQ proof assistant [21]. The last software release of `Real2Float` provides OCAML [2] and COQ libraries and is freely available¹. Our implementation tool is built in top of the `NLCertify` verification system [48]. Precision and efficiency of the tool are evaluated on several benchmarks coming from the existing literature. Numerical experiments demonstrate that our method competes well with recent approaches relying on Taylor-interval approximations [62] or combining SMT solvers with affine arithmetic [23].

The paper is organized as follows. In Section 2, we recall mandatory background on roundoff errors due to finite precision arithmetic before describing our nonlinear program semantics (Section 2.1). Then we remind how to perform certified polynomial optimization based on semidefinite programming (Section 2.2) and how to obtain formal bounds while checking the certificates inside the COQ proof assistant (Section 2.3). Section 3 contains the main contribution of the paper, namely how to compute tight over-approximations for roundoff errors of nonlinear programs with sparse semidefinite relaxations. Finally, Section 4 is devoted to the evaluation of our nonlinear verification tool `Real2Float` on benchmarks arising from control systems, optimization, physics and biology.

¹forge.ocamlcore.org/frs/?group_id=351 (not anonymized)

2. Preliminaries

2.1 Program Semantics and Floating-point Numbers

We adopt the standard practice [38] to approximate a real number x with its closest floating-point representation $\hat{x} = x(1 + e)$, with $|e|$ is less than the machine precision ϵ . The validity of this model is ensured as we neglect both overflow and denormal range values. The operator $\hat{\cdot}$ is called the rounding operator and can be selected among rounding to nearest, rounding toward zero (resp. $\pm\infty$). The scientific notation of a binary (resp. decimal) floating-point number \hat{x} is a triple (s, sig, exp) consisting of a sign bit s , a *significand* $sig \in [1, 2)$ (resp. $[1, 10)$) and an *exponent* exp , yielding numerical evaluation $(-1)^s sig 2^{exp}$ (resp. $(-1)^s sig 10^{exp}$).

The value of ϵ actually gives the upper bound of the relative floating-point error and is equal to 2^{-prec} , where $prec$ is called the *precision*, referring to the number of significand bits used. For single precision floating-point, one has $prec = 24$. For double (resp. quadruple) precision, one has $prec = 53$ (resp. $prec = 113$). Let us define \mathbb{R} the set of real numbers and \mathbb{F} the set of binary floating-point numbers. For each real-valued operation $\text{bop}_{\mathbb{R}} \in \{+, -, \times, /, \}$, the result of the corresponding floating-point operation $\text{bop}_{\mathbb{F}} \in \{\oplus, \ominus, \otimes, \oslash\}$ satisfies the following when complying with IEEE 754 standard arithmetic [3]:

$$\text{bop}_{\mathbb{F}}(\hat{x}, \hat{y}) = \text{bop}_{\mathbb{R}}(x, y)(1 + e) \quad , \quad |e| \leq \epsilon = 2^{-prec} \quad . \quad (3)$$

Other operations include special functions taken from a *dictionary* \mathcal{D} , containing the unary functions \tan , \arctan , \cos , \arccos , \sin , \arcsin , \exp , \log , $(\cdot)^r$ with $r \in \mathbb{R} \setminus \{0\}$. For $f_{\mathbb{R}} \in \mathcal{D}$, the corresponding floating-point evaluation satisfies

$$f_{\mathbb{F}}(\hat{x}) = f_{\mathbb{R}}(x)(1 + e) \quad , \quad |e| \leq \epsilon(f_{\mathbb{R}}) \quad . \quad (4)$$

The value of the relative error bound $\epsilon(f_{\mathbb{R}})$ differs from the machine precision ϵ in Equation (3) and has to be properly adjusted. We refer the interested reader to [12] for relative error bound verification of transcendental functions (see also [37] for formalization in HOL-LIGHT).

Program semantics We support conditional code without procedure calls nor loops. Despite these restrictions, we can consider a wide range of nonlinear programs while assuming that important numerical calculations can be expressed in a loop-free manner. Our programs are encoded in an ML-like language:

```
let box_prog      x1 ... xn = [(a1, b1); ...; (an, bn)];;
let obj_prog      x1 ... xn = [(f(x), eReal2Float)];;
let cstr_prog      x1 ... xn = [g1(x); ...; gk(x)];;
let uncert_prog   x1 ... xn = [u1; ...; un];;
```

Here, the first line encodes interval constraints for input variables, namely $\mathbf{x} := (x_1, \dots, x_n) \in [a_1, b_1] \times \dots \times [a_n, b_n]$. The second line provides the function $f(\mathbf{x})$ as well as the total roundoff error bound $e_{\text{Real2Float}}$. Then, one encodes polynomial nonnegativity constraints over the input variables, namely $g_1(\mathbf{x}) \geq 0, \dots, g_k(\mathbf{x}) \geq 0$. Finally, the last line allows the user to specify a numerical constant u_i to associate a given uncertainty to the variable x_i , for each $i = 1, \dots, n$.

The type of numerical constants is denoted by \mathbb{C} . In our current implementation, the user can choose either 64 bits floating-point or arbitrary-size rational numbers. This type \mathbb{C} is used for the terms $e_{\text{Real2Float}}$, u_1, \dots, u_n , a_1, \dots, a_n , b_1, \dots, b_n . The inductive type of polynomial expressions with coefficients in \mathbb{C} is pExprC defined as follows:

```
type pexprC = | Pc of C | Px of positive
| Psub of pexprC * pexprC | Pneg of pexprC
| Padd of pexprC * pexprC
| Pmul of pexprC * pexprC
```

The constructor Px takes a positive integer as argument to represent either an input or local variable. The inductive type nlexpr of nonlinear expressions (such as $f(\mathbf{x})$) is defined as follows:

```
type nlexpr =
| Pol of pexprC | Neg of nlexpr
| Add of nlexpr * nlexpr
| Mul of nlexpr * nlexpr
| Sub of nlexpr * nlexpr
| Div of nlexpr * nlexpr | Sqrt of nlexpr
| Transc of transc * nlexpr
| IfThenElse of pexprC * nlexpr * nlexpr
| Let of positive * nlexpr * nlexpr
```

The type transc corresponds to the dictionary \mathcal{D} of special functions. For instance, the term $\text{Transc}(exp, f(\mathbf{x}))$ represents the program implementing $\exp(f(\mathbf{x}))$. Given a polynomial expression p and two nonlinear expressions f and g , the term $\text{IfThenElse}(p(\mathbf{x}), f(\mathbf{x}), g(\mathbf{x}))$ represents the conditional program implementing **if** $(p(\mathbf{x}) \geq 0)$ $f(\mathbf{x})$ **else** $g(\mathbf{x})$. The constructor **Let** allows us to define local variables in an ML fashion, e.g. **let** $t_1 = 331.4 + 0.6 * T$ **in** $-t_1 * v / ((t_1 + u) * (t_1 + u))$ (part of the *doppler1* program considered in Section 4).

Finally, one obtains rounded nonlinear expressions using an inductive procedure $\text{round} : \text{nlexpr} \rightarrow \text{nlexpr}$, defined accordingly to Equation (3) and Equation (4). When an uncertainty u_i is specified for an input variable x_i , the corresponding rounded expression is given by $x_i(1 + e)$, with $|e| \leq u_i$.

2.2 SDP relaxations for polynomial optimization

The sums of squares method involves approximation of polynomial inequality constraints by sums of squares (SOS) equality constraints. Here we recall mandatory background about SOS. We apply this method in Section 3 to solve the problems of Equation (1) when the nonlinear function r is a polynomial.

Sums of squares certificates and SDP First we remind basic facts about generation of SOS certificates for polynomial optimization, using semidefinite programming. Denote by $\mathbb{R}[\mathbf{x}]$ the vector space of polynomials and by $\mathbb{R}_{2d}[\mathbf{x}]$ the restriction of $\mathbb{R}[\mathbf{x}]$ to polynomials of degree at most $2d$. Let us define the set of SOS polynomials:

$$\Sigma[\mathbf{x}] := \left\{ \sum_i q_i^2, \text{ with } q_i \in \mathbb{R}[\mathbf{x}] \right\}, \quad (5)$$

as well as its restriction $\Sigma_{2d}[\mathbf{x}] := \Sigma[\mathbf{x}] \cap \mathbb{R}_{2d}[\mathbf{x}]$ to polynomials of degree at most $2d$. For instance, the following bivariate polynomial $\sigma(\mathbf{x}) := 1 + (x_1^2 - x_2^2)^2$ lies in $\Sigma_4[\mathbf{x}] \subseteq \mathbb{R}_4[\mathbf{x}]$.

Optimization methods based on SOS use the implication $p \in \Sigma[\mathbf{x}] \implies \forall \mathbf{x} \in \mathbb{R}^n, p(\mathbf{x}) \geq 0$, i.e. the inclusion of $\Sigma[\mathbf{x}]$ in the set of nonnegative polynomials. Given $r \in \mathbb{R}[\mathbf{x}]$, one considers the following polynomial minimization problem:

$$r^* := \inf_{\mathbf{x} \in \mathbb{R}^n} \{ r(\mathbf{x}) : \mathbf{x} \in \mathbf{K} \}, \quad (6)$$

where the set of constraints $\mathbf{K} \subseteq \mathbb{R}^n$ is defined by

$$\mathbf{K} := \{ \mathbf{x} \in \mathbb{R}^n : g_1(\mathbf{x}) \geq 0, \dots, g_k(\mathbf{x}) \geq 0 \},$$

for polynomial functions g_1, \dots, g_k . The set \mathbf{K} is called a *basic semialgebraic set*. Membership to semialgebraic sets is ensured by satisfying conjunctions of polynomial nonnegativity constraints.

Remark 1. *When the input variables satisfy interval constraints $\mathbf{x} \in [a_1, b_1] \times \dots \times [a_n, b_n]$ then one can easily show that there exists some integer $M > 0$ such that $M - \sum_{i=1}^n x_i^2 \geq 0$. In the sequel, we assume that this nonnegativity constraint appears explicitly in the definition of \mathbf{K} . Such an assumption is mandatory to prove the convergence of semidefinite relaxations recalled in Theorem 2.1.*

In general, the objective function r and the set of constraints \mathbf{K} can be nonconvex, which makes the resolution of Problem (6) difficult to solve in practice. One can rewrite Problem (6) as the equivalent maximization problem:

$$r^* := \sup_{\mathbf{x} \in \mathbb{R}^n, \mu \in \mathbb{R}} \{ \mu : r(\mathbf{x}) - \mu \geq 0, \forall \mathbf{x} \in \mathbf{K} \}. \quad (7)$$

Now we outline how to handle the nonnegativity constraint $r - \mu \geq 0$. Given a nonnegative polynomial $p \in \mathbb{R}[\mathbf{x}]$, the existence of an SOS decomposition $p = \sum_i q_i^2$ valid over \mathbb{R}^n is ensured by the existence of a symmetric real matrix \mathbf{Q} , a solution of the following linear matrix feasibility problem:

$$p(\mathbf{x}) = \mathbf{m}_d(\mathbf{x})^\top \mathbf{Q} \mathbf{m}_d(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^n, \quad (8)$$

where $\mathbf{m}_d(\mathbf{x}) := (1, x_1, \dots, x_n, x_1^2, x_1x_2, \dots, x_n^d)$ and the matrix \mathbf{Q} has only nonnegative eigenvalues. Such a matrix \mathbf{Q} is called *positive semidefinite*. The vector \mathbf{m}_d (resp. matrix \mathbf{Q}) has a size (resp. dimension) equal to $s_n^d := \binom{n+d}{d}$. Problem (8) can be handled with semidefinite programming (SDP) solvers, such as MOSEK [7] or SDPA [67] (see [65] for specific background about SDP). Then, one computes the ‘‘LDL’’ decomposition $\mathbf{Q} = \mathbf{L}^\top \mathbf{D} \mathbf{L}$ (variant of the classical Cholesky decomposition), where \mathbf{L} is a lower triangular matrix and \mathbf{D} is a diagonal matrix. Finally, one obtains $p(\mathbf{x}) = (\mathbf{L} \mathbf{m}_d(\mathbf{x}))^\top \mathbf{D} (\mathbf{L} \mathbf{m}_d(\mathbf{x})) = \sum_{i=0}^{s_n^d} q_i(\mathbf{x})^2$. Such a decomposition is called a sums of squares (SOS) *certificate*.

Example 1. *Let us define $p(\mathbf{x}) := \frac{1}{4} + x_1^4 - 2x_1^2x_2^2 + x_2^4$. With $\mathbf{m}_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$, one solves the linear matrix feasibility problem $p(\mathbf{x}) = \mathbf{m}_2(\mathbf{x})^\top \mathbf{Q} \mathbf{m}_2(\mathbf{x})$. One can show that the solution writes $\mathbf{Q} = \mathbf{L}^\top \mathbf{D} \mathbf{L}$ for a 6×6 matrix \mathbf{L} and a diagonal matrix \mathbf{D} with entries $(\frac{1}{2}, 0, 0, 1, 0, 0)$, yielding the SOS decomposition: $p(\mathbf{x}) = (\frac{1}{2})^2 + (x_1^2 - x_2^2)^2 =: \sigma(\mathbf{x})$. It is enough to prove that p is nonnegative.*

Dense SDP relaxations for polynomial optimization

In order to solve our goal problem (Problem (1)), we are trying to solve Problem (6), recast as Problem (7). We first explain how to obtain tractable approximations of this difficult problem. Define $g_0 := 1$. The hierarchy of SDP relaxations developed by Lasserre [42] provides lower bounds of r^* , through solving the optimization problems (\mathbf{P}_d) :

$$(\mathbf{P}_d) : \begin{cases} p_d^* := \sup_{\sigma_j, \mu} \mu, \\ \text{s.t.} & r(\mathbf{x}) - \mu = \sum_{j=0}^k \sigma_j(\mathbf{x}) g_j(\mathbf{x}), \forall \mathbf{x}, \\ & \mu \in \mathbb{R}, \sigma_j \in \Sigma[\mathbf{x}], \quad j = 0, \dots, k, \\ & \deg(\sigma_j g_j) \leq 2d, \quad j = 0, \dots, k. \end{cases}$$

The next theorem is a consequence of the assumption mentioned in Remark 1.

Theorem 2.1 (Lasserre [42]). *Let p_d^* be the optimal value of the SDP relaxation (\mathbf{P}_d) . Then, the sequence of optimal values $(p_d^*)_{d \in \mathbb{N}}$ is nondecreasing and converges to r^* .*

The number of SDP variables grows polynomially with the integer d , called the *relaxation order*. Indeed, at fixed n , the relaxation (\mathbf{P}_d) involves $O((2d)^n)$ SDP variables and $(k+1)$ linear matrix inequalities (LMIs) of size $O(d^n)$. When d increases, then more accurate lower bounds of r^* can be obtained, at an increasing computational cost. At fixed d , the relaxation (\mathbf{P}_d) involves $O(n^{2d})$ SDP variables and $(d+1)$ linear matrix inequalities (LMIs) of size $O(n^d)$.

Exploiting sparsity Here we remind how to exploit the structured sparsity of the problem to replace one SDP problem (\mathbf{P}_d) by an SDP problem (\mathbf{S}_d) of size $O(\kappa^{2d})$ where κ is the average size of the maximal cliques of the correlation pattern of the polynomial variables (see [43, 66] for more details). We now present these notions as well as the formulation of sparse SDP relaxations (\mathbf{S}_d) .

We note \mathbb{N}^n the set of n -tuple of nonnegative integers. The support of a polynomial $r(\mathbf{x}) := \sum_{\alpha \in \mathbb{N}^n} r_\alpha \mathbf{x}^\alpha$ is defined as $\text{supp}(r) := \{ \alpha \in \mathbb{N}^n : r_\alpha \neq 0 \}$. For instance the support of $p(\mathbf{x}) := \frac{1}{4} + x_1^4 - 2x_1^2x_2^2 + x_2^4$ is $\text{supp}(p) = \{ (0, 0), (4, 0), (2, 2), (0, 4) \}$.

Let F_j be the index set of variables which are involved in the polynomial g_j , for each $j = 1, \dots, k$. The correlative sparsity is represented by the $n \times n$ correlative sparsity matrix (csp matrix) \mathbf{R} defined by:

$$\mathbf{R}(i, j) := \begin{cases} 1 & \text{if } i = j, \\ 1 & \text{if } \exists \alpha \in \text{supp}(f) \text{ such that } \alpha_i, \alpha_j \geq 1, \\ 1 & \text{if } \exists k \in \{1, \dots, m\} \text{ such that } i, j \in F_k, \\ 0 & \text{otherwise.} \end{cases}$$

We define the undirected csp graph $G(N, E)$ with $N = \{1, \dots, n\}$ and $E = \{\{i, j\} : i, j \in N, i < j, \mathbf{R}(i, j) = 1\}$. Then, let $C_1, \dots, C_m \subseteq N$ denote the maximal cliques of $G(N, E)$ and define $n_j := \#C_j$, for each $j = 1, \dots, m$.

Remark 2. *From the assumption of Remark 1, one can add the m redundant additional constraints:*

$$g_{k+j} := n_j M^2 - \sum_{i \in C_j} x_i^2 \geq 0, \quad j = 1, \dots, m, \quad (9)$$

set $k' = k + m$, define the compact semialgebraic set:

$$\mathbf{K}' := \{ \mathbf{x} \in \mathbb{R}^n : g_1(\mathbf{x}) \geq 0, \dots, g_{k'}(\mathbf{x}) \geq 0 \},$$

and modify Problem (6) into the following optimization problem:

$$r^* := \inf_{\mathbf{x} \in \mathbb{R}^n} \{ r(\mathbf{x}) : \mathbf{x} \in \mathbf{K}' \}. \quad (10)$$

For each $j = 1, \dots, m$, we note $\mathbb{R}_{2d}[\mathbf{x}, C_j]$ the set of polynomials of $\mathbb{R}_{2d}[\mathbf{x}]$ which involve the variables $(x_i)_{i \in C_j}$. We note $\Sigma[\mathbf{x}, C_j] := \Sigma[\mathbf{x}] \cap \mathbb{R}_{2d}[\mathbf{x}, C_j]$. Similarly, we define $\Sigma[\mathbf{x}, F_j]$, for each $j = 1, \dots, k$. The following program is the sparse variant of the SDP program (\mathbf{P}_d) :

$$(\mathbf{S}_d) : \begin{cases} r_d^* := \sup_{\mu, \sigma_j} \mu, \\ \text{s.t.} & r(\mathbf{x}) - \mu = \sum_{j=0}^{k'} \sigma_j(\mathbf{x}) g_j(\mathbf{x}), \forall \mathbf{x}, \\ & \mu \in \mathbb{R}, \sigma_0 \in \sum_{j=1}^m \Sigma[\mathbf{x}, C_j], \\ & \sigma_j \in \Sigma[\mathbf{x}, F_j], \quad j = 1, \dots, k', \\ & \deg(\sigma_j g_j) \leq 2d, \quad j = 0, \dots, k', \end{cases}$$

where $\sigma \in \sum_{j=1}^m \Sigma[\mathbf{x}, C_j]$ if and only if there exist $\sigma^1 \in \Sigma[\mathbf{x}, C_1], \dots, \sigma^m \in \Sigma[\mathbf{x}, C_m]$ such that $\sigma(\mathbf{x}) = \sum_{j=1}^m \sigma^j(\mathbf{x})$, for all $\mathbf{x} \in \mathbb{R}^n$.

The number of SDP variables of the relaxation (\mathbf{S}_d) is $\sum_{j=1}^m \binom{n_j+2d}{2d}$. At fixed d , it yields an SDP problem with $O(\kappa^{2d})$ variables, where $\kappa := \frac{1}{m} \sum_{j=1}^m n_j$ is the average size of the cliques C_1, \dots, C_m . Moreover, the cliques C_1, \dots, C_m satisfy the running intersection property:

Definition 2.2 (RIP). *Let $m \in \mathbb{N}_0$ and I_1, \dots, I_m be subsets of $\{1, \dots, n\}$. We say that I_1, \dots, I_m satisfy the running intersection property (RIP) when for all $i = 1, \dots, m$, there exists an integer $l < i$ such that $I_i \cap (\cup_{j < i} I_j) \subseteq I_l$.*

This RIP property together with the assumption mentioned in Remark 2 allow to state the sparse variant of Theorem 2.1:

Theorem 2.3 (Lasserre [43, Theorem 3.6]). *Let r_d^* be the optimal value of the sparse SDP relaxation (\mathbf{S}_d) . Then the sequence $(r_d^*)_{d \in \mathbb{N}}$ is nondecreasing and converges to r^* .*

The interested reader can find more details in [66] about additional ways to exploit sparsity in order to derive analogous sparse SDP relaxations. We illustrate the benefits of the SDP relaxations (\mathbf{S}_d) with the following example:

Example 2. *Consider the polynomial f mentioned in Section 1: $f(\mathbf{x}) := x_2x_5 + x_3x_6 - x_2x_3 - x_5x_6 + x_1(-x_1 + x_2 + x_3 - x_4 + x_5 + x_6)$. Here, $n = 6, d = 2, N = \{1, \dots, 6\}$. The 6×6 correlative sparsity matrix \mathbf{R} is:*

$$\mathbf{R} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

The csp graph G associated to \mathbf{R} is depicted in Figure 1. The

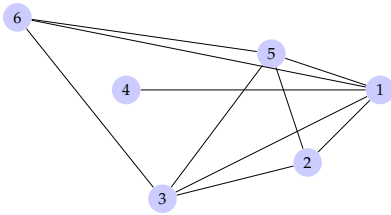


Figure 1. Correlative sparsity pattern graph for the variables of f

maximal cliques of G are $C_1 := \{1, 4\}$, $C_2 := \{1, 2, 3, 5\}$ and $C_3 := \{1, 3, 5, 6\}$. For $d = 2$, the dense SDP relaxation (\mathbf{P}_2) involves $\binom{6+4}{4} = 210$ variables against $\binom{2+4}{4} + 2\binom{4+4}{4} = 115$ for the sparse variant (\mathbf{S}_2) . The dense SDP relaxation (\mathbf{P}_3) involves 924 variables against 448 for the sparse variant (\mathbf{S}_3) . This difference becomes significant while considering that the time complexity of semidefinite programming is polynomial w.r.t. the number of variables with an exponent greater than 3 (see [9, Chapter 4] for more details).

2.3 Computer proofs for polynomial optimization

Here, we briefly recall some existing features of the COQ proof assistant to handle formal polynomial optimization,

when using SDP relaxations. The advantage of such relaxations is that they provide SOS certificates, which can be formally checked *a-posteriori*. For more details on COQ, we recommend the documentation available in [10]. Giving a polynomial r and a set of constraints \mathbf{K} , one can obtain a lower bound of r by solving any instance of Problem (\mathbf{P}_d) . Then, one can verify formally the correctness of the lower bound r_d^* , using the SOS certificate output $\sigma_0, \dots, \sigma_k$. Indeed it is enough to prove the polynomial equality $r(\mathbf{x}) - r_d^* = \sum_{j=0}^k \sigma_j(\mathbf{x})g_j(\mathbf{x})$ inside COQ. Such equalities can be efficiently proved using COQ's ring tactic [31] via the mechanism of computational reflection [17]. Any polynomial of type `peprC` (see Section 2.1) can be normalized to a unique polynomial of type `polC` (see [31] for more details on the constructors of this type). For the sake of clarity, let consider the unconstrained case, i.e. $\mathbf{K} = \mathbb{R}^n$. One encodes an SOS certificate $\sigma_0(\mathbf{x}) = \sum_{i=1}^m q_i^2$ with the sequence of polynomials $[q_1; \dots; q_m]$, each q_i being of type `polC`. To prove the equality $r = \sigma_0$, our version of the ring tactic normalizes both r and the sequence $[q_1; \dots; q_m]$ and compares the two normalization results. This mechanism is illustrated in Figure 2 with the polynomial $p := \frac{1}{4} + x_1^4 - 2x_1^2x_2^2 + x_2^4$ (see Example 1) being encoded by `p` and the polynomials $1/2$ and $x_1^2 - x_2^2$ being encoded respectively by `q1` and `q2`.

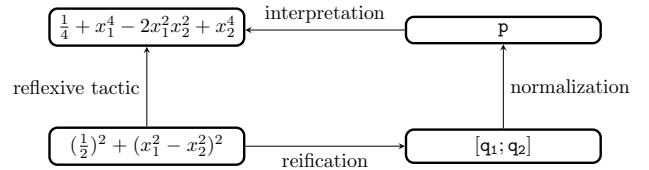


Figure 2. An illustration of computational reflection

In the general case, this computational step is done through a `checker_sos` procedure which returns a Boolean value. If this value is true, one applies a correctness lemma, whose conclusion yields the nonnegativity of $r - r_d^*$ over \mathbf{K} . In practice, the SDP solvers are implemented in floating-point precision, thus the above equality between $r - r_d^*$ and the SOS certificate does not hold. However, following Remark 1, each variable lies in a closed interval, thus one can bound the remainder polynomial $\epsilon(\mathbf{x}) := r(\mathbf{x}) - r_d^* - \sum_{j=0}^k \sigma_j(\mathbf{x})g_j(\mathbf{x})$ using basic interval arithmetic, so that the lower bound ϵ^* of ϵ yields the valid inequality: $\forall \mathbf{x} \in \mathbf{K}, r(\mathbf{x}) \geq r_d^* + \epsilon^*$. For more explanation, we refer the interested reader to the formal framework [50]. Note that this formal verification remains valid when considering the sparse variant (\mathbf{S}_d) .

3. Guaranteed Roundoff Error Bounds using SDP Relaxations

In this section, we present our new algorithm, relying on sparse SDP relaxations, to bound roundoff errors of nonlinear programs. After stating our general algorithm (Section 3.1), we detail how this procedure can handle polynomial programs (Section 3.2) and then present extensions to the non-polynomial case (Section 3.3).

3.1 The General Optimization Framework

Here we consider a given program which implements a nonlinear transcendental expression f with input variables \mathbf{x} satisfying a set of constraints \mathbf{X} . We assume that \mathbf{X} is included in a box (i.e. a product of closed intervals)

$[\mathbf{a}, \mathbf{b}] := [a_1, b_1] \times \cdots \times [a_n, b_n]$ and that \mathbf{X} is encoded as follows:

$$\mathbf{X} := \{ \mathbf{x} \in \mathbb{R}^n : g_1(\mathbf{x}) \geq 0, \dots, g_k(\mathbf{x}) \geq 0 \},$$

for polynomial functions g_1, \dots, g_k . Then, we denote by $\hat{f}(\mathbf{x}, \mathbf{e})$ the rounded expression of f after applying the `round` procedure (see Section 2.1), introducing additional error variables \mathbf{e} .

The algorithm `bound`, depicted in Figure 3, takes as input $\mathbf{x}, \mathbf{X}, f, \hat{f}, \mathbf{e}$ as well as the set \mathbf{E} of bound constraints over \mathbf{e} . Here we assume that our program implementing f does not involve conditional statements (this case will be discussed later in Section 3.3). For a given machine ϵ , one has $\mathbf{E} := [-\epsilon, \epsilon]^m$, with m being the number of error variables. This algorithm actually relies on the sparse SDP optimization procedure (\mathbf{S}_d) (see Section 2.2 for more details), thus `bound` also takes as input a relaxation order $d \in \mathbb{N}$. The algorithm provides as output an interval enclosure I_d of the absolute error $|\hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})|$ over \mathbf{K} . From this interval $I_d := [\underline{f}_d, \overline{f}_d]$, one can compute $f_d := \max\{-\underline{f}_d, \overline{f}_d\}$, which is a sound upper bound of the maximal absolute error $r^* := \max_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} |\hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})|$.

Input: input variables \mathbf{x} , input constraints \mathbf{X} , nonlinear expression f , rounded expression \hat{f} , error variables \mathbf{e} , error constraints \mathbf{E} , relaxation order d

Output: interval enclosure I_d of the absolute error $|\hat{f} - f|$ over $\mathbf{K} := \mathbf{X} \times \mathbf{E}$

- 1: Define the absolute error $r(\mathbf{x}, \mathbf{e}) := \hat{f}(\mathbf{x}, \mathbf{e}) - f(\mathbf{x})$
- 2: Compute $l(\mathbf{x}, \mathbf{e}) := r(\mathbf{x}, 0) + \sum_{j=1}^m \frac{\partial r(\mathbf{x}, \mathbf{e})}{\partial e_j}(\mathbf{x}, 0) e_j$
- 3: Compute $h := r - l$
- 4: Compute interval bounds for h : $I^h := \text{ia_bound}(h, \mathbf{K})$
- 5: Compute interval bounds for l : $I_d^l := \text{sdp_bound}(l, \mathbf{K}, d)$
- 6: **return** $I_d := I_d^l + I^h$

Figure 3. `bound`: our algorithm to compute roundoff errors bounds of nonlinear programs

After computing the absolute roundoff error $r := \hat{f} - f$ (Line 1), one decomposes r as the sum of an expression l which is affine w.r.t. the error variable \mathbf{e} and a remainder h . One way to obtain l is to compute the vector of partial derivatives of r w.r.t. \mathbf{e} evaluated at $(\mathbf{x}, 0)$ and finally to take the inner product of this vector and \mathbf{e} (Line 2). Then, the idea is to compute a precise bound of l and a coarse bound of h . The underlying reason is that h involves error term products of degree greater than 2 (e.g. $e_1 e_2$), yielding an interval enclosure I^h of *a priori* much smaller width, compared to the interval enclosure I^l of l . One obtains I^h using the procedure `ia_bound` implementing basic interval arithmetic (Line 4).

3.2 Polynomial Programs

We first describe our `sdp_bound` optimization algorithm when implementing polynomial programs. In this case, `sdp_bound` calls an auxiliary procedure `sdp_poly`. The bound of l is provided through solving two sparse SDP instances of Problem (\mathbf{S}_d), at relaxation order d . We now give more explanation about the `sdp_poly` procedure.

We can map each input variable x_i to the integer i , for all $i = 1, \dots, n$, as well as each error variable e_j to $n + j$, for all $j = 1, \dots, m$. Then, define the sets $C_1 := \{1, \dots, n, n+1\}, \dots, C_m := \{1, \dots, n, n+m\}$. Here, we take

advantage of the sparsity correlation pattern of l by using m distinct sets of cardinality $n + 1$ rather than a single one of cardinality $n + m$, i.e. the total number of variables. After noticing that $r(\mathbf{x}, 0) = \hat{f}(\mathbf{x}, 0) - f(\mathbf{x}) = 0$, one can scale the optimization problems by writing

$$l(\mathbf{x}, \mathbf{e}) = \sum_{j=1}^m s_j(\mathbf{x}) e_j = \epsilon \sum_{j=1}^m s_j(\mathbf{x}) \frac{e_j}{\epsilon}, \quad (11)$$

with $s_j(\mathbf{x}) := \frac{\partial r(\mathbf{x}, \mathbf{e})}{\partial e_j}(\mathbf{x}, 0)$, for all $j = 1, \dots, m$. Replacing \mathbf{e} by \mathbf{e}/ϵ leads to compute an interval enclosure of l/ϵ over $\mathbf{K}' := \mathbf{X} \times [-1, 1]^m$. Recall that from Remark 1, there exists an integer $M > 0$ such that $M - \sum_{i=1}^n x_i^2 \geq 0$, as the input variable satisfy box constraints. Moreover, to fulfil the assumption of Remark 2, one encodes \mathbf{K}' as follows:

$$\mathbf{K}' := \{ (\mathbf{x}, \mathbf{e}) \in \mathbb{R}^{n+m} : g_1(\mathbf{x}) \geq 0, \dots, g_k(\mathbf{x}) \geq 0, \\ g_{k+1}(\mathbf{x}, e_1) \geq 0, \dots, g_{k+m}(\mathbf{x}, e_m) \geq 0 \},$$

with $g_{k+j}(\mathbf{x}, e_j) := M + 1 - \sum_{i=1}^n x_i^2 - e_j^2$, for all $j = 1, \dots, m$. The index set of variables involved in g_j is $F_j := N = \{1, \dots, n\}$ for all $j = 1, \dots, k$. The index set of variables involved in g_{k+j} is $F_j := C_j$ for all $j = 1, \dots, m$.

Then, one can compute a lower bound of the minimum of $l'(\mathbf{x}, \mathbf{e}) := l(\mathbf{x}, \mathbf{e})/\epsilon$ over \mathbf{K}' by solving the following optimization problem:

$$\left\{ \begin{array}{l} \underline{l}'_d := \sup_{\mu, \sigma_j} \mu, \\ \text{s.t.} \quad l' - \mu = \sigma_0 + \sum_{j=1}^{k+m} \sigma_j g_j, \\ \mu \in \mathbb{R}, \sigma_0 \in \sum_{j=1}^m \Sigma[(\mathbf{x}, \mathbf{e}), C_j], \\ \sigma_j \in \Sigma[(\mathbf{x}, \mathbf{e}), F_j], j = 1, \dots, k+m, \\ \deg(\sigma_j g_j) \leq 2d, j = 1, \dots, k+m. \end{array} \right. \quad (12)$$

A feasible solution of Problem (12) ensures the existence of $\sigma^1 \in \Sigma[(\mathbf{x}, e_1)], \dots, \sigma^m \in \Sigma[(\mathbf{x}, e_m)]$ such that $\sigma_0 = \sum_{j=0}^m \sigma^j$, allowing the following reformulation:

$$\left\{ \begin{array}{l} \underline{l}'_d := \sup_{\mu, \sigma_j} \mu, \\ \text{s.t.} \quad l' - \mu = \sum_{j=1}^m \sigma^j + \sum_{j=1}^{k+m} \sigma_j g_j, \\ \mu \in \mathbb{R}, \sigma_j \in \Sigma[\mathbf{x}], j = 1, \dots, m, \\ \sigma^j \in \Sigma[(\mathbf{x}, e_j)], \deg(\sigma^j) \leq 2d, j = 1, \dots, m, \\ \deg(\sigma_j g_j) \leq 2d, j = 1, \dots, k+m. \end{array} \right. \quad (13)$$

An upper bound \overline{l}'_d can be obtained by replacing \sup with \inf and $l' - \mu$ by $\mu - l'$ in Problem (13). Our optimization procedure `sdp_poly` computes the lower bound \underline{l}'_d as well as an upper bound \overline{l}'_d of l' over \mathbf{K}' then returns the interval $I'_d := [\epsilon \underline{l}'_d, \epsilon \overline{l}'_d]$, which is a sound enclosure of the values of l over \mathbf{K} .

We emphasize two advantages of the decomposition $r := l + h$ and more precisely of the linear dependency of l w.r.t. \mathbf{e} : scalability and robustness to SDP numerical issues. First, no computation is required to determine the correlation sparsity pattern of l , by comparison to the general case. Thus, it becomes much easier to handle the optimization of l with the sparse SDP Problem (13) rather than with the

corresponding instance of the dense relaxation (\mathbf{P}_d). While the latter involves $\binom{n+m+2d}{2d}$ SDP variables, the former involves only $m \binom{n+1+2d}{2d}$ variables, ensuring the scalability of our framework. In addition, the linear dependency of l w.r.t. \mathbf{e} allows us to scale the error variables and optimize over a set of variables lying in $\mathbf{K}' := \mathbf{X} \times [-1, 1]$. It ensures that the range of input variables does not significantly differ from the range of error variables. This condition is mandatory while considering SDP relaxations because most SDP solvers (e.g. MOSEK [7], SDPA [67]) are implemented using double precision floating-point. It is impossible to optimize l over \mathbf{K} (rather than l' over \mathbf{K}') when the maximal value ϵ of error variables is less than 2^{-53} , due to the fact that SDP solvers would treat each error variable term as 0, and consequently l as the zero polynomial. Hence, this decomposition insures our framework from the numerical issues related to finite-precision implementation of SDP solvers.

Let us define the interval enclosure $I^l := [\underline{l}, \bar{l}]$, with $\underline{l} := \inf_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} l(\mathbf{x}, \mathbf{e})$ and $\bar{l} := \sup_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} l(\mathbf{x}, \mathbf{e})$. The next lemma states that one can approximate I^l as closely as desired using the `sdp_poly` procedure.

Lemma 3.1 (Convergence of the `sdp_poly` procedure). *Let I_d^l be the interval enclosure returned by the procedure `sdp_poly`(l, \mathbf{K}, d). The sequence $(I_d^l)_{d \in \mathbb{N}}$ converges to I^l .*

Proof. It is sufficient to show the similar convergence result for $l' = l/\epsilon$, as it implies the convergence for l by a scaling argument. The sets C_1, \dots, C_m satisfy the RIP property (see Definition 2.2). Moreover, the encoding of \mathbf{K}' satisfies the assumption mentioned in Remark 2. Thus, Theorem 2.3 implies that the sequence of lower bounds $(\underline{l}'_d)_{d \in \mathbb{N}}$ converges to $\underline{l}' := \inf_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}'} l'(\mathbf{x}, \mathbf{e})$. Similarly, the sequence of upper bounds converge to \bar{l}' , yielding the desired result. \square

Lemma 3.1 guarantees asymptotic convergence to the exact enclosure of l when the relaxation order d tends to infinity. However, it is more reasonable in practice to keep this order as small as possible to obtain tractable SDP relaxations. Hence, we generically solve each instance of Problem (13) at the minimal relaxation order, that is $d_0 := \max\{\lceil \deg l/2 \rceil, \max_{1 \leq j \leq k+m} \{\lceil \deg(g_j)/2 \rceil\}\}$.

3.3 Non-polynomial and Conditional Programs

Other classes of programs do not only involve polynomials but also semialgebraic and transcendental functions as well as conditional statements. Such programs are of particular interest as they often occur in real-world applications such as biology modeling, space control or global optimization. We present how the general optimization procedure `sdp_bound` can be extended to these nonlinear programs.

Semialgebraic programs Here we assume that the function l is semialgebraic, that is involves non-polynomial components such as divisions or square roots. Following [45], we explain how to transform the optimization problem $\inf_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}} l(\mathbf{x}, \mathbf{e})$ into a polynomial optimization problem, then use the sparse SDP program (13). One way to perform this reformulation consists of introducing lifting variables to represent non-polynomial operations. We first illustrate the extension to semialgebraic programs with an example.

Example 3. *Let consider the program implementing the rational function $f : [0, 1] \rightarrow \mathbb{R}$ defined by $f(x_1) := \frac{x_1}{1+x_1}$. Applying the rounding procedure (with machine ϵ) yields*

$\hat{f}(x_1, \mathbf{e}) := \frac{x_1(1+e_2)}{(1+x_1)(1+e_1)}$ and the decomposition $r(x_1, \mathbf{e}) := \hat{f}(x_1, \mathbf{e}) - f(x_1) = l(x_1, \mathbf{e}) + h(x_1, \mathbf{e}) = s_1(x_1)e_1 + s_2(x_1)e_2 + h(x_1, \mathbf{e})$. One has $s_1(x_1) = \frac{\partial r(x_1, \mathbf{e})}{\partial e_1}(x_1, 0) = -\frac{x_1}{1+x_1}$ and $s_2(x_1) = -s_1(x_1)$.

Let $\mathbf{K} := [0, 1] \times [-\epsilon, \epsilon]^2$. One introduces a lifting variable $x_2 := \frac{x_1}{1+x_1}$ to handle the division operator and encode the equality constraint $p(\mathbf{x}) := x_2(1+x_1) - x_1 = 0$ with the two inequality constraints $p(\mathbf{x}) \geq 0$ and $-p(\mathbf{x}) \geq 0$. To ensure the compactness assumption, one bounds x_2 within $I := [0, 1/2]$, using basic interval arithmetic.

Let $\mathbf{K}_{\text{poly}} := \{(\mathbf{x}, \mathbf{e}) \in [0, 1] \times I \times [\epsilon, \epsilon]^2 : p(\mathbf{x}) \geq 0, -p(\mathbf{x}) \geq 0\}$. Then the rational optimization problem involving l is equivalent to $\inf_{(\mathbf{x}, \mathbf{e}) \in \mathbf{K}_{\text{poly}}} x_2(-e_1 + e_2)$, a polynomial optimization problem that we can handle with the `sdp_poly` procedure, described in Section 3.2.

In the semialgebraic case, `sdp_bound` calls an auxiliary procedure `sdp_sa`. Given input variables $\mathbf{y} := (\mathbf{x}, \mathbf{e})$, input constraints $\mathbf{K} := \mathbf{X} \times \mathbf{E}$ and a semialgebraic function l , `sdp_sa` first applies a recursive procedure `lift` which returns variables \mathbf{y}_{poly} , constraints \mathbf{K}_{poly} and a polynomial f_{poly} such that the interval enclosure I^l of $l(\mathbf{y})$ over \mathbf{K} is equal to the interval enclosure of the polynomial $l_{\text{poly}}(\mathbf{y}_{\text{poly}})$ over \mathbf{K}_{poly} . Calling `sdp_sa` yields the interval enclosure $I_d^l := \text{sdp_poly}(l_{\text{poly}}, \mathbf{K}_{\text{poly}}, d)$. We detail the lifting procedure `lift` in Figure 4 for the constructors `Pol` (Line (2)), `Div` (Line (3)) and `Sqrt` (Line (8)). The interval I obtained through the `ia_bound` procedure (Line (1)) allows us to constrain the additional variable x to ensure the assumption of Remark 2. For the sake of consistency, we omit the other cases (`Neg`, `Add`, `Mul` and `Sub`) where the procedure is straightforward. For a similar procedure in the context of global optimization, we refer the interested reader to [47, Chapter 2].

Input: input variables \mathbf{y} , input constraints \mathbf{K} , semialgebraic expression f
Output: variables \mathbf{y}_{poly} , constraints \mathbf{K}_{poly} , polynomial expression f_{poly}

- 1: $I := \text{ia_bound}(f, \mathbf{K})$
- 2: **if** $f = \text{Pol}(p)$ **then** $\mathbf{y}_{\text{poly}} := \mathbf{y}$, $\mathbf{K}_{\text{poly}} := \mathbf{K}$, $f_{\text{poly}} := p$
- 3: **else if** $f = \text{Div}(g, h)$ **then**
- 4: $\mathbf{y}_g, \mathbf{K}_g, g_{\text{poly}} := \text{lift}(\mathbf{y}, \mathbf{K}, g)$
- 5: $\mathbf{y}_h, \mathbf{K}_h, h_{\text{poly}} := \text{lift}(\mathbf{y}, \mathbf{K}, h)$
- 6: $\mathbf{y}_{\text{poly}} := (\mathbf{y}_g, \mathbf{y}_h, x)$ $f_{\text{poly}} := x$
- 7: $\mathbf{K}_{\text{poly}} := \{\mathbf{y}_{\text{poly}} \in \mathbf{K}_g \times \mathbf{K}_h \times I : x g_{\text{poly}} = f_{\text{poly}}\}$
- 8: **else if** $f = \text{Sqrt}(g)$ **then**
- 9: $\mathbf{y}_g, \mathbf{K}_g, g_{\text{poly}} := \text{lift}(\mathbf{y}, \mathbf{K}, g)$
- 10: $\mathbf{y}_{\text{poly}} := (\mathbf{y}_g, x)$ $f_{\text{poly}} := x$
- 11: $\mathbf{K}_{\text{poly}} := \{\mathbf{y}_{\text{poly}} \in \mathbf{K}_g \times I : x^2 = g_{\text{poly}}\}$
- 12: ...
- 13: **end**
- 14: **return** $\mathbf{y}_{\text{poly}}, \mathbf{K}_{\text{poly}}, f_{\text{poly}}$

Figure 4. lift: a recursive procedure to reduce semialgebraic problems to polynomial problems

The set of variables \mathbf{y}_{poly} can be decomposed as $(\mathbf{x}_{\text{poly}}, \mathbf{e})$, where \mathbf{x}_{poly} gathers input variables with lifting variables and has a cardinality equal to n_{poly} . Then, one easily shows that the sets $\{1, \dots, n_{\text{poly}}, e_1\}, \dots, \{1, \dots, n_{\text{poly}}, e_m\}$ satisfy the RIP, thus ensuring to solve efficiently the corresponding instances of Problem (13).

Transcendental programs Here we assume that the function l is transcendental, i.e. involves univariate non-

semialgebraic components such as \exp or \sin . For each univariate transcendental function $f_{\mathbb{R}}$ in our dictionary set \mathcal{D} , one assumes that $f_{\mathbb{R}}$ is twice differentiable, so that the univariate function $g := f_{\mathbb{R}} + \frac{\gamma}{2} \cdot | \cdot |^2$ is convex on I for large enough $\gamma > 0$ (for more details, see the references [4, 52]). It follows that there exists a constant $\gamma \leq \sup_{x \in I} -f_{\mathbb{R}}''(x)$ such that for all $x_i \in I$:

$$\begin{aligned} \forall x \in I, \quad f_{\mathbb{R}}(x) &\geq f_{x_i}^-(x), \\ \text{with } f_{x_i}^- &:= -\frac{\gamma}{2}(x - x_i)^2 + f'_{\mathbb{R}}(x_i)(x - x_i) + f_{\mathbb{R}}(x_i), \end{aligned} \quad (14)$$

implying that for all $x \in I$, $f_{\mathbb{R}}(x) \geq \max_{x_i \in I} f_{x_i}^-(x)$. Similarly, one obtains an upper-approximation $\min_{x_i \in I} f_{x_i}^+(x)$. Figure 5 provides such approximations for the function $f_{\mathbb{R}}(x) := \log(1 + \exp(x))$ on the interval $I := [-8, 8]$.

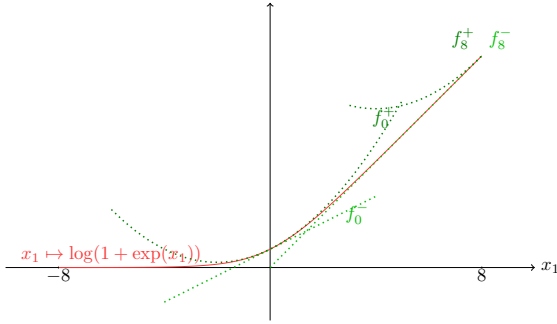


Figure 5. Semialgebraic Approximations for $x \mapsto \log(1 + \exp(x))$: $\max\{f_0^-(x), f_8^-(x)\} \leq \log(1 + \exp(x)) \leq \min\{f_0^+(x), f_8^+(x)\}$

For transcendental programs, our procedure `sdp_bound` calls the auxiliary procedure `sdp_transc`. Given input variables (\mathbf{x}, \mathbf{e}) , constraints \mathbf{K} and a transcendental function l , `sdp_transc` first computes a semialgebraic lower (resp. upper) approximation l^- (resp. l^+) of l over \mathbf{K} . For more details in the context of global optimization, we refer the reader to [49]. Then, calling the procedure `sdp_sa` allows us to get interval enclosures of l^- as well as l^+ . We illustrate the procedure to handle transcendental programs with an example.

Example 4. Let consider the program implementing the transcendental function $f : [-8, 8] \rightarrow \mathbb{R}$ defined by $f(x_1) := \log(1 + \exp(x_1))$. Applying the rounding procedure yields $\hat{f}(x_1, \mathbf{e}) := \log[(1 + \exp(x_1)(1 + e_1))(1 + e_2)(1 + e_3)]$. Here, $|e_2|$ is bounded by the machine ϵ while $|e_1|$ (resp. $|e_3|$) is bounded with an adjusted absolute error $\epsilon_1 := \epsilon(\exp)$ (resp. $\epsilon_3 := \epsilon(\log)$). Let $\mathbf{K} := [-8, 8] \times [-\epsilon_1, \epsilon_1] \times [\epsilon, \epsilon] \times [-\epsilon_3, \epsilon_3]$.

One obtains the decomposition $r(x_1, \mathbf{e}) := \hat{f}(x_1, \mathbf{e}) - f(x_1) = l(x_1, \mathbf{e}) + h(x_1, \mathbf{e}) = s_1(x_1)e_1 + s_2(x_1)e_2 + s_3(x_1)e_3 + h(x_1, \mathbf{e})$, with $s_1(x_1) = \frac{\exp(x_1)}{1 + \exp(x_1)}$, $s_2(x_1) = 1$ and $s_3(x_1) = \log(1 + \exp(x_1)) = f(x_1)$. Figure 5 provides a lower approximation $s_3^- := \max\{f_0^-, f_8^-\}$ of s_3 as well as an upper approximation $s_3^+ := \min\{f_0^+, f_8^+\}$. One can get similar approximations s_1^- and s_1^+ for s_1 . One first obtains (coarse) interval enclosures $I_2 = \mathbf{ia_bound}(s_1, \mathbf{K})$ and $I_3 = \mathbf{ia_bound}(s_3, \mathbf{K})$ and one introduces extra variables $x_2 \in I_2$ and $x_3 \in I_3$ to represent s_1 and s_3 respectively. Then, the interval enclosure of l over \mathbf{K} is equal to the interval enclosure of $l_{sa}(\mathbf{x}, \mathbf{e}) := x_2e_1 + e_2 + x_3e_3$ over the set $\mathbf{K}_{sa} := \{(x_1, \mathbf{e}) \in \mathbf{K}, (x_2, x_3) \in I_2 \times I_3, s_1^-(x_1) \leq x_2 \leq s_1^+(x_1), s_3^-(x_1) \leq x_3 \leq s_3^+(x_1)\}$.

Programs with conditionals Finally, we explain how to extend our bounding procedure to nonlinear programs involving conditionals through the recursive algorithm given in Figure 6. The `bound_nlprog` algorithm relies on the `bound` procedure (see Figure 3 in Section 3.1) to compute round-off error bounds of programs implementing transcendental functions (Line 12). From Line 1 to Line 11, the algorithm handles the case when the program implements a function f defined as follows:

$$f(\mathbf{x}) := \begin{cases} g(\mathbf{x}) & \text{if } p(\mathbf{x}) \geq 0, \\ h(\mathbf{x}) & \text{otherwise.} \end{cases}$$

The first branch output is g while the second one is h .

Input: input variables \mathbf{x} , input constraints \mathbf{X} , nonlinear expression f , rounded expression \hat{f} , error variables \mathbf{e} , error constraints \mathbf{E} , relaxation order d

Output: interval enclosure I_d of the error $|\hat{f} - f|$ over $\mathbf{K} := \mathbf{X} \times \mathbf{E}$

```

1: if  $f = \text{IfThenElse}(p, g, h)$  then
2:    $I_d^p := \text{bound}(\mathbf{x}, \mathbf{X}, p, \hat{p}, \mathbf{e}, \mathbf{E}, d) = [\underline{p}_d, \overline{p}_d]$ 
3:    $\mathbf{X}_1 := \{\mathbf{x} \in \mathbf{X} : 0 \leq p(\mathbf{x}) \leq \overline{p}_d\}$ 
4:    $\mathbf{X}_2 := \{\mathbf{x} \in \mathbf{X} : \underline{p}_d \leq p(\mathbf{x}) \leq 0\}$ 
5:    $\mathbf{X}_3 := \{\mathbf{x} \in \mathbf{X} : 0 \leq p(\mathbf{x})\}$ 
6:    $\mathbf{X}_4 := \{\mathbf{x} \in \mathbf{X} : p(\mathbf{x}) \leq 0\}$ 
7:    $I_d^1 := \text{bound\_nlprog}(\mathbf{x}, \mathbf{X}_1, g, \hat{h}, \mathbf{e}, \mathbf{E}, d)$ 
8:    $I_d^2 := \text{bound\_nlprog}(\mathbf{x}, \mathbf{X}_2, h, \hat{g}, \mathbf{e}, \mathbf{E}, d)$ 
9:    $I_d^3 := \text{bound\_nlprog}(\mathbf{x}, \mathbf{X}_3, g, \hat{g}, \mathbf{e}, \mathbf{E}, d)$ 
10:   $I_d^4 := \text{bound\_nlprog}(\mathbf{x}, \mathbf{X}_4, h, \hat{h}, \mathbf{e}, \mathbf{E}, d)$ 
11:  return  $I_d := I_d^1 \cup I_d^2 \cup I_d^3 \cup I_d^4$ 
12: else return  $I_d := \text{bound}(\mathbf{x}, \mathbf{X}, f, \hat{f}, \mathbf{e}, \mathbf{E}, d)$ 
13: end

```

Figure 6. `bound_nlprog`: our algorithm to compute round-off error bounds of programs with conditional statements

A preliminary step consists of computing the roundoff error enclosure $I_d^p := [\underline{p}_d, \overline{p}_d]$ (Line 2) for the program implementing the polynomial p . Then the procedure computes bounds related to the divergence path error, that is the maximal value between the four following errors:

- (Line 7) the error obtained while computing the rounded result \hat{h} of the second branch instead of computing the exact result g of the first one, occurring for the set of variables (\mathbf{x}, \mathbf{e}) such that $\hat{p}(\mathbf{x}, \mathbf{e}) \leq 0 \leq p(\mathbf{x})$. For scalability and numerical issues, we consider an over-approximation \mathbf{X}_1 (Line 3) of this set, where the variables \mathbf{x} satisfy the relaxed constraints $0 \leq p(\mathbf{x}) \leq \overline{p}_d$.
- (Line 8) the error obtained while computing the rounded result \hat{g} of the first branch instead of computing the exact result h of the second one, occurring for the set of variables (\mathbf{x}, \mathbf{e}) such that $p(\mathbf{x}) \leq 0 \leq \hat{p}(\mathbf{x}, \mathbf{e})$. We also consider an over-approximation \mathbf{X}_2 (Line 4), where the variables \mathbf{x} satisfy the relaxed constraints $\underline{p}_d \leq p(\mathbf{x}) \leq 0$.
- (Line 9) the roundoff error corresponding to the program implementation of g .
- (Line 10) the roundoff error corresponding to the program implementation of h .

Simplification of error terms In addition, our algorithm `bound_nlprog` integrates several features to reduce the number of error variables. First, it memorizes all sub-expressions of the nonlinear expression tree to perform common sub-expressions elimination. We can also simplify error term products, thanks to the following lemma.

Lemma 3.2 (Higham [38, Lemma 3.3]). *Let $\epsilon < \frac{1}{k}$ and $\gamma_k := \frac{k\epsilon}{1-k\epsilon}$. Then, for all $e_1, \dots, e_k \in [-\epsilon, \epsilon]$, there exists θ_k such that $\prod_{i=1}^k (1 + e_i) = 1 + \theta_k$ and $|\theta_k| \leq \gamma_k$.*

Lemma 3.2 implies that for any k such that $\epsilon < \frac{1}{k}$, one has $\gamma_k \leq (k+1)\epsilon$. Our algorithm has an option to automatically derive safe over-approximations of the absolute roundoff error while introducing only one variable e_1 (bounded by $(k+1)\epsilon$) instead of k error variables e_1, \dots, e_k (bounded by ϵ). The cost of solving the corresponding optimization problem can be significantly reduced but it yields coarser error bounds.

4. Experimental Evaluation

Now, we present experimental results obtained by applying our general `bound_nlprog` algorithm (see Section 3, Figure 6) to various examples coming from physics, biology, space control and optimization. The `bound_nlprog` algorithm is implemented in a tool called `Real2Float`, built in top of the `NLCertify` nonlinear verification package, relying on `OCAML` (Version 4.02.1), `COQ` (Version 8.4pl5) and interfaced with the `SDP` solver `SDPA` (Version 7.3.6). The `SDP` solver output numerical `SOS` certificates, which are converted into rational `SOS` using the `ZARITH` `OCAML` library (Version 1.2), implementing arithmetic operations over arbitrary-precision integers. For more details about the installation and usage of `Real2Float`, we refer to the dedicated web-page² and the setup instructions³. All examples are displayed in Appendix A as the corresponding `Real2Float` input text files and satisfy our nonlinear program semantics (see Section 2.1). All results have been obtained on an Intel Core i5 CPU (2.40 GHz). Execution timings have been computed by averaging over five runs.

4.1 Benchmark Presentation

For each example, we compared the quality of the roundoff error bounds (Table 1) and corresponding execution times (Table 2) while running our tool `Real2Float`, `FPTaylor` (version from May 2015) [62] and `Rosa` (version from May 2014) [23]. A given program implements a nonlinear function $f(\mathbf{x})$, involving variables \mathbf{x} lying in a set \mathbf{X} contained in a box $[\mathbf{a}, \mathbf{b}]$. Applying our rounding model on f yields the nonlinear expression $\hat{f}(\mathbf{x}, \mathbf{e})$, involving additional error variables \mathbf{e} lying in a set \mathbf{E} . At a given semidefinite relaxation order d , our tool computes the upper bound f_d of the absolute roundoff error $|f - \hat{f}|$ over $\mathbf{K} := \mathbf{X} \times \mathbf{E}$ and verifies that it is less than a requested number $\epsilon_{\text{Real2Float}}$. As we keep the relaxation order d as low as possible to ensure tractable `SDP` programs, it can happen that $f_d > \epsilon_{\text{Real2Float}}$. In this case, we subdivide a randomly chosen interval of the box $[\mathbf{a}, \mathbf{b}]$ in two halves to obtain two sub-sets \mathbf{X}_1 and \mathbf{X}_2 , fulfilling $\mathbf{X} := \mathbf{X}_1 \cup \mathbf{X}_2$, and apply the `bound_nlprog` algorithm on both sub-sets until we succeed to certify that $\epsilon_{\text{Real2Float}}$ is a sound upper bound of the roundoff error.

The number $\epsilon_{\text{Real2Float}}$ is compared with the upper bounds computed by `FPTaylor`, which relies on Taylor Symbolic expansions [62], `Rosa`, which relies on SMT and affine arithmetic [23], as well as a third procedure, `IA`, relying on interval arithmetic. We designed `IA` to follow the same steps than `bound_nlprog`, together with the sub-procedure `bound`, but

²<http://nl-certify.forge.ocamlcore.org/real2float.html> (not anonymized)

³see the `README.md` file in the top level directory

to compute an interval enclosure of l with the procedure `ia_bound` using a basic interval arithmetic procedure, instead of calling the `sdp_bound` procedure (see Line 5 of the algorithm depicted in Figure 3). For comparison purpose, we also executed each program using random inputs, following the approach used in the `Rosa` paper [23]. Specifically, we executed each program on 10^7 random inputs satisfying the input restrictions. The results from these random samples provide lower bounds on the absolute error. For the sake of further presentation, we associate an alphabet character (from `a` to `w`) to identify each of the 23 nonlinear programs:

- The first 14 programs implement polynomial and semi-algebraic functions: `a-e` come from physics, `f` and `h` from biology, `g` from control, `i-k` are derived from expressions involved in the proof of Kepler Conjecture (for more details, see [32] and the related formalization project [33]) and `l-n` implement polynomial approximations of the sine and square root functions. With the exception of `i-k`, all these programs are used to compare `FPTaylor` and `Rosa` in [62].
- The three polynomial programs `o-q` come from the global optimization literature and correspond respectively to Problem 3.3, 4.6 and 4.7 in [27]. We selected them as they typically involve nontrivial polynomial preconditions (i.e. \mathbf{X} is not a simple box but rather a set defined with conjunction of nonlinear polynomial inequalities).
- The four programs `r-u` involve transcendental functions. The two programs `r` and `s` are used in the `FPTaylor` paper [62] and correspond respectively to the program `logexp` (see Example 4) and the program `sphere` taken from NASA World Wind Java SDK [1]. The 2 programs `t` and `q` respectively implement the functions coming from the optimization problems *Hartman 3* and *Hartman 6* in [6], involving both sums of exponential functions composed with quadratic polynomials.
- The last two programs `v-w` involve conditional statements and come from the static analysis literature. They correspond to the two respective running examples of [30] and [5]. The first program `v` is used in the `Rosa` paper [23] for the analysis of divergence path error.

The three tools `Real2Float`, `Rosa` and `FPTaylor` can handle input variable uncertainties as well as multi-precision programs, but for the sake of conciseness, we only considered to compare their performance on programs implemented in single ($\epsilon = 2^{-24}$) or double ($\epsilon = 2^{-53}$) precision floating-point. For the programs involving transcendental functions, we followed the same procedure as in `FPTaylor` while adjusting the precision $\epsilon(f_{\mathbb{R}}) = 1.5\epsilon$ for each special function $f_{\mathbb{R}} \in \{\sin, \cos, \log, \exp\}$. Each univariate transcendental function is approximated from below (resp. from above) using suprema (resp. infima) of linear or quadratic polynomials (see Example 4 for the case of program `logexp`).

4.2 Comparison Results

Comparison results for error bound computation are presented in Table 1. Our `Real2Float` tool computes the tightest upper bounds for 18 out of 23 benchmarks. For the 3 programs `a-c` involving rational functions and the program `t` encoding Problem *Hartman 3*, `FPTaylor` computes the tightest bounds. The symbol `Div0` means that the `IA` procedure aborts with a division by zero exception, which typically occurs when computing too coarse interval enclosures of rational function denominators. One current limitation of `Real2Float` is its ability to manipulate symbolic expressions, e.g. computing rational function derivatives or yield-

Table 1. Comparison results of upper and lower bounds for absolute roundoff errors (the best results are emphasized using bold fonts)

Benchmark	id	precision	Real2Float	Rosa	FPTaylor	IA	lower bound
Programs involving polynomial and semialgebraic functions							
doppler1	a	(double)	2.75e-13	4.97e-13	1.57e-13	Div0	7.11e-14
doppler2	b	(double)	8.04e-13	1.29e-12	2.87e-13	Div0	1.14e-13
doppler3	c	(double)	1.37e-13	2.03e-13	8.16e-14	Div0	4.27e-14
rigidBody1	d	(double)	3.80e-13	5.08e-13	3.87e-13	3.80e-13	2.28e-13
rigidBody2	e	(double)	3.98e-11	6.48e-11	5.24e-11	3.98e-11	2.19e-11
verhulst	f	(double)	3.37e-16	6.82e-16	3.50e-16	6.22e-01	2.23e-16
carbonGas	g	(double)	1.09e-08	1.60e-08	1.25e-08	3.10e-03	4.11e-09
predPrey	h	(double)	1.57e-16	1.98e-16	1.87e-16	2.02e-16	1.47e-16
kepler0	i	(double)	8.76e-14	9.07e-14	1.04e-13	1.03e-13	2.23e-14
kepler1	j	(double)	3.90e-13	4.43e-13	4.48e-13	6.52e-13	7.58e-14
kepler2	k	(double)	1.90e-12	2.16e-12	2.07e-12	3.00e-12	3.03e-13
sineTaylor	l	(double)	5.53e-16	9.57e-16	6.71e-16	9.39e-16	4.45e-16
		(float)	2.97e-07	1.03e-06	3.51e-07	5.07e-07	1.79e-07
sineOrder3	m	(double)	6.68e-16	1.11e-15	9.96e-16	8.82e-16	3.34e-16
		(float)	3.58e-07	1.19e-06	5.35e-07	4.74e-07	2.12e-07
sqrt	n	(double)	7.56e-16	8.41e-16	7.87e-16	8.48e-16	4.45e-16
		(float)	4.06e-07	9.03e-07	4.23e-07	4.56e-07	2.45e-07
Programs implementing polynomial functions with polynomial preconditions							
floudas1	o	(double)	3.00e-13	2.99e-13	6.20e-13	6.36e-13	1.48e-13
floudas2	p	(double)	8.79e-16	1.12e-15	9.96e-16	1.12e-15	2.35e-16
floudas3	q	(double)	7.33e-15	1.00e-14	1.16e-14	1.87e-14	7.31e-15
Programs implementing transcendental functions							
logexp	r	(double)	1.65e-15	–	1.71e-15	8.29e-13	1.19e-15
sphere	s	(double)	7.78e-15	–	1.29e-14	7.78e-15	5.05e-15
hartman3	t	(double)	1.50e-13	–	1.34e-14	3.46e+05	1.10e-14
hartman6	u	(double)	2.00e-13	–	OoM	2.82e+03	6.50e-14
Programs involving conditional loops							
cav10	v	(double)	2.91e+00	2.91e+00	–	1.02e+02	2.90e+00
perin	w	(double)	2.01e+00	2.01e+00	–	4.91e+01	2.00e+00

ing reduction to the same denominator. For that reason, we omitted some programs studied in [62] (e.g. *turbine*) involving rational functions. Performance of *FPTaylor* are better to analyze such functions, as it handles properly symbolic operations through the interface with the *MAXIMA* computer algebra system [51]. Accordingly with the *FPTaylor* paper [62], the *Rosa* real compiler often provides coarser bounds, except for the two conditional programs *v* and *w* and the polynomial program *o*. Program *u* can only be tackled with *Real2Float* as *FPTaylor* aborted after running out of memory (meaning of the symbol OoM). A possible failure explanation is the complexity of the corresponding Problem *Hartman 6*, involving 133 arithmetic operations and 6 input variables. To the best of our knowledge, *Real2Float* is the only academic tool which is able to handle the general class of programs involving either transcendental functions or conditional statements. The *FPTaylor* (resp. *Rosa*) does not currently handle conditionals (resp. transcendental functions), as meant by the symbol – in the corresponding column entries. However, an interface between both software would embed them with their respective missing feature.

These error bound comparison results together with their corresponding execution timings (given in Table 2) are used to plot the data points shown in Figure 7. For

each benchmark identified by *id*, let $t_{\text{Real2Float}}$ (in 3rd column of Table 2) refer to the execution time of *Real2Float* to obtain the corresponding upper bound $\epsilon_{\text{Real2Float}}$ (in 4th column of Table 1). Similarly, let us define the execution times t_{Rosa} , t_{FPTaylor} and the corresponding error bounds ϵ_{Rosa} , $\epsilon_{\text{FPTaylor}}$. Then the x-axis coordinate of the point id (resp. id) displayed in Figure 7 corresponds to the relative difference between the execution time of *Rosa* (resp. *FPTaylor*) and *Real2Float*, i.e. the ratio $\frac{t_{\text{Rosa}} - t_{\text{Real2Float}}}{t_{\text{Real2Float}}}$ (resp. $\frac{t_{\text{FPTaylor}} - t_{\text{Real2Float}}}{t_{\text{Real2Float}}}$). Similarly, the y-axis coordinate of the point id (resp. id) is $\frac{\epsilon_{\text{Rosa}} - \epsilon_{\text{Real2Float}}}{\epsilon_{\text{Real2Float}}}$ (resp. $\frac{\epsilon_{\text{FPTaylor}} - \epsilon_{\text{Real2Float}}}{\epsilon_{\text{Real2Float}}}$). For readability purpose, we used an appropriate logarithmic scale for the x-axis.

The axes of the coordinate system of Figure 7 divide the plane into four quadrants: the nonnegative quadrant (+, +) contains data points referring to programs for which *Real2Float* computes the tighter bounds in less time, the second one (+, –) contains points referring to programs for which *Real2Float* is slower but more accurate, the non-positive quadrant (–, –) for which *Real2Float* is slower and computes coarser bounds and the last one (–, +) for which *Real2Float* is faster but less accurate. On the quadrant

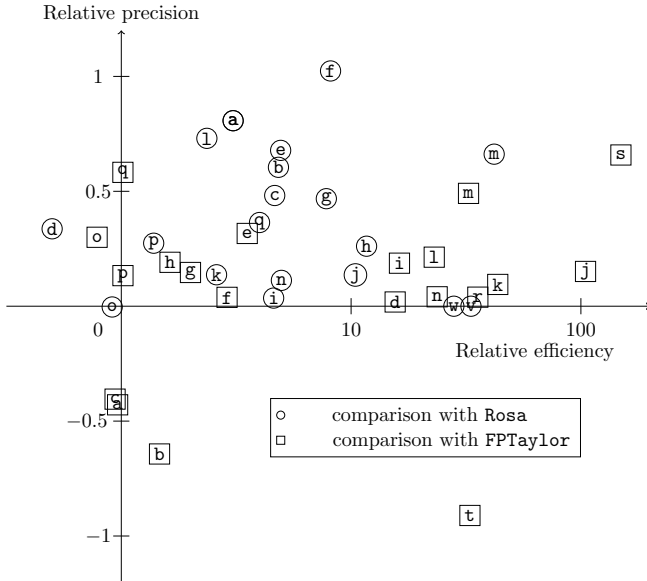


Figure 7. Comparisons of execution times and upper bounds of roundoff errors obtained with Rosa and FPTaylor, relatively to Real2Float

Table 2. Comparison of execution times (in seconds) for absolute roundoff error bounds (the best results are emphasized using **bold fonts**)

Benchmark	id	Real2Float	Rosa	FPTaylor
doppler1	a	7.73	23.7	7.45
doppler2	b	4.97	24.0	7.29
doppler3	c	7.61	35.4	7.15
rigidBody1	d	0.40	0.20	6.22
rigidBody2	e	2.37	11.7	8.37
verhulst	f	1.22	9.93	3.52
carbonGas	g	5.05	39.4	10.1
predPrey	h	3.10	36.2	5.05
kepler0	i	0.93	4.28	15.1
kepler1	j	3.15	32.9	98.0
kepler2	k	21.6	56.1	215
sineTaylor	l	0.42	0.99	9.66
sineOrder3	m	0.16	6.71	5.19
sqrroot	n	0.38	1.89	10.1
floudas1	o	19.5	17.8	15.3
floudas2	p	1.80	2.49	1.83
floudas3	q	4.33	17.3	4.40
logexp	r	0.06	–	2.04
sphere	s	0.03	–	4.48
hartman3	t	1.90	–	62.5
hartman6	u	15.6	–	OoM
cav10	v	0.37	12.3	–
perin	w	1.38	41.4	–

(+, –), one can see that Real2Float computes bound which are less accurate than FPTaylor on programs a–c and t, but much faster for program b. The quadrant (–, +) indicates that Rosa (resp. FPTaylor) is more efficient but less precise than Real2Float on program d (resp. o). The presence of the majority of plots on the nonnegative quadrant (+, +)

confirms that Real2Float does not compromise efficiency at the expense of accuracy.

For each program implementing polynomials, our tool has an option to provide formal guarantees for the corresponding roundoff error bound $\epsilon_{\text{Real2Float}}$. Using the formal mechanism described in Section 2.3, Real2Float formally checks inside Coq the SOS certificates generated by the SDP solver.

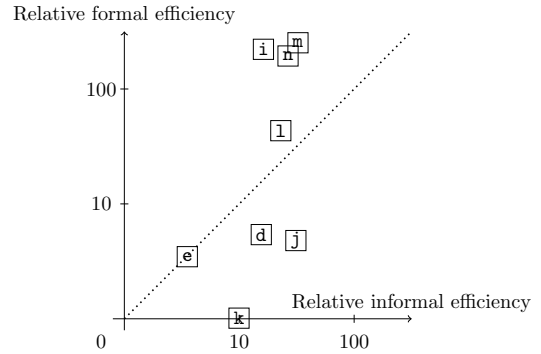


Figure 8. Comparisons of informal and formal execution times to certify bounds of roundoff errors obtained with FPTaylor, relatively to Real2Float

The FPTaylor software has a similar option to provide formal scripts which can be checked inside the HOL-LIGHT proof assistant, for programs involving polynomial and transcendental functions. Our tool has more limitations to perform formal verification as it cannot handle non-polynomial programs. Next, we describe the formal proof results obtained while verifying the bounds for the eight polynomial programs d–f and i–n. Figure 8 allows to compare the execution times of Real2Float and FPTaylor required to analyze the eight programs in both informal (i.e. without verification inside Coq or HOL-LIGHT) and formal settings. The x-axis coordinate of each point \boxed{id} displayed in Figure 7 is the same as on Figure 7. The dotted line represents the bisector of the first quadrant angle. The y-axis coordinate of the point \boxed{id} corresponds to the relative difference between the execution time $t_{\text{HOL-LIGHT}}$ to obtain a HOL-LIGHT formal proof with FPTaylor and the execution time t_{Coq} to obtain a Coq formal proof with Real2Float, i.e. the ratio $\frac{t_{\text{HOL-LIGHT}} - t_{\text{Coq}}}{t_{\text{Coq}}}$. Note that all points have nonnegative y-axis coordinates, as our tool formally checks all program bounds more efficiently than FPTaylor (more than hundred times faster for the three programs i, m–n). The figure shows that the speedup ratio in the formal setting is higher than in the informal setting for the four programs i and l–n, as the corresponding points are above the first bisector. This emphasizes the benefit of using SOS certificates for formal verification rather than optimization methods based on Taylor approximations. The latter performs almost as well as the former to analyze program k but yields a coarser bound.

5. Related Works

Satisfiability Modulo Theories (SMT) SMT solvers allow to analyze programs with various semantics or specifications but are limited for the manipulation of problems involving nonlinear arithmetic. Several solvers, including Z3 [25], provide partial support for the IEEE floating-point standard [60]. They suffer from a lack of scalability when

used for roundoff error analysis in isolation (as emphasized in [23]), but can be integrated into existing frameworks, e.g. FPHILE [58]. The procedure in [28] can solve SMT problems over the real numbers, using interval constraint propagation but has not yet been applied to quantification of roundoff error. The *Rosa* tool [23] provides a way to compile real programs involving semialgebraic functions and conditional statements. The tool uses affine arithmetic to provide sound over-approximations of roundoff errors, allowing for generation of finite precision implementations which fulfill the required precision given as input by the user. Bounds of the affine expressions are provided through an optimization procedure based on SMT. In our case, we use the same rounding model but provide approximations which are affine w.r.t. the additional error variables and nonlinear w.r.t. the input variables. Instead of using SMT, we bound the resulting expressions with optimization techniques based on semidefinite programming. At the moment (and in contrast to our method) the *Rosa* tool does not formally verify the output bound provided by the SMT solver, but such a feature could be embedded through an interface with the *smtcoq* framework [8]. This latter tool allows the proof witness generated by an SMT solver to be formally (and independently) re-checked inside COQ. The *smtcoq* framework uses tactics based on *computational reflection* to enable this re-checking to be performed efficiently.

Abstract Domains Abstract interpretation [22] has been extensively used in the context of static analysis to provide sound over-approximations, called *abstractions*, of the sets of values taken by program variables. The effects of variable assignments, guards and conditional loops statements are handled with several domain specific operators (e.g. inclusion, meet and join). Well studied abstract domains include intervals [55] as well as more complicated frameworks based on affine arithmetic [63], octogons [54], zonotopes [30], polyhedra [18], interval polyhedra [19], some of them being implemented inside a tool called APRON [39]. Abstract domains provide sound over-approximations of program expressions, and allow upper bounds on roundoff error to be computed. The GAPPA tool [24] relies on interval abstract domains with an extension to affine domains [46], to reason about roundoff errors. As demonstrated in [62], the bounds obtained inside GAPPA are often coarser than other methods which take into account the variable correlations. Formal guarantees can be provided as GAPPA benefits from an interface with COQ while making use of interval libraries [53] relying on formalized floating-points [15]. The static analysis tool FLUCTUAT relies on affine abstract domains [13]. GAPPA and FLUCTUAT tools use a different rounding model (also available as an option inside *FPTaylor*) based on a piecewise constant relative error bound. This is more precise than our current rounding model but requires extensive use of a branch and bound algorithm as each interval has to be subdivided in intervals $[2^n, 2^{n+1}]$ for several values of the integer n . As a side effect, the corresponding optimization procedure is computationally demanding, as noticed in [62].

Global Optimization Frameworks Computing sound bounds of nonlinear expressions is mandatory to perform formal analysis of finite precision implementations and can be performed with various optimization tools. In the polynomial case, alternative approaches to semidefinite relaxations are based on decomposition in the multivariate Bernstein basis. Formal verification of bounds obtained with this decomposition has been investigated in the thesis of

Zumkeller [68] and by Munõz and Narkawicz [56] in the PVS theorem prover [57]. We are not aware of any work based on these techniques which can quantify roundoff errors. Another decomposition of nonnegative polynomials into SOS certificates consists in using the Krivine [40]-Handelman [35] representation and boils down to solving linear programming (LP) relaxations. In our case, we use a different representation, leading to solve SDP relaxations. The Krivine-Handelman representation has been used in [14] to compute roundoff error bounds. LP relaxations often provide coarser bounds than SDP relaxations and it has been proven in [41] that generically finite convergence does not occur for convex problems, at the exception of the linear case. Branch and bound methods with Taylor models [11] are not restricted to polynomial systems and have been formalized [61] to solve nonlinear inequalities occurring in the proof of Kepler Conjecture. Symbolic Taylor Expansions [62] have been implemented in the *FPTaylor* tool to compute formal bounds of roundoff errors for programs involving both polynomial and transcendental functions. This method happens to be efficient and precise to analyze various programs and it would be interesting to design a procedure combining *FPTaylor* with our tool on specific subsets of input constraints.

6. Conclusion and Perspectives

Our verification framework allows us to over-approximate roundoff errors occurring while executing nonlinear programs implemented with finite precision. The framework relies on semidefinite optimization, ensuring certified approximations. Our approach extends to medium-size nonlinear problems, due to automatic detection of the correlation sparsity pattern of input variables and roundoff error variables. Experimental results indicate that the optimization algorithm implemented in our *Real2Float* software package often produces tighter error bounds than the ones provided by the competitive solvers *Rosa* and *FPTaylor*. In addition, *Real2Float* produces sums of squares certificates which guarantee the correctness of these upper bounds and can be efficiently verified inside the COQ proof assistant.

This work yields several directions for further research investigation. First, we intend to increase the size of graspable instances by exploiting symmetry patterns of certain program sub-classes to use specific SDP hierarchies [59]. We could also provide roundoff error bounds for more general programs, involving either finite or infinite conditional loops. A preliminary mandatory step is to be able to generate inductive invariants with SDP relaxations. Another interesting direction would be to apply the method used in [44] to derive sequences of lower roundoff error bounds together with SDP-based certificates. On the formal proof side, we could benefit from floating-point/interval arithmetic libraries available inside COQ, first to improve the efficiency of the formal polynomial checker, currently relying on exact arithmetic, then to extend the formal verification to non-polynomial programs. Finally, we plan to combine this optimization framework with the procedure in [29] to improve the automatic re-ordering of arithmetic expressions, allowing more efficient optimization of FPGA implementations.

Acknowledgments

This work was partly funded by the Engineering and Physical Sciences Research Council (EPSRC) Challenging Engineering Grant (EP/I020457/1).

References

- [1] NASA World Wind Java SDK. <http://worldwind.arc.nasa.gov/java/>.
- [2] Objective Caml (OCaml) programming language website. <http://caml.inria.fr/>.
- [3] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008. .
- [4] M. Akian, S. Gaubert, and V. Kolokoltsov. Set coverings and invertibility of Functional Galois Connections. In G. Litvinov and V. Maslov, editors, *Idempotent Mathematics and Mathematical Physics*, volume 377 of *Contemporary Mathematics*, pages 19–51. American Mathematical Society, 2005. URL <http://hal.inria.fr/inria-00000966>. Also ESI Preprint 1447, <http://arXiv.org/abs/math.FA/0403441>.
- [5] M. P. Alexandre Maréchal. Three linearization techniques for multivariate polynomials in static analysis using convex polyhedra. Technical Report TR-2014-7, Verimag Research Report, 2014.
- [6] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky. A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems. *J. of Global Optimization*, 31(4):635–672, Apr. 2005. ISSN 0925-5001. .
- [7] E. Andersen and K. Andersen. The mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. In H. Frenk, K. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, volume 33 of *Applied Optimization*, pages 197–232. Springer US, 2000. ISBN 978-1-4419-4819-9. . URL http://dx.doi.org/10.1007/978-1-4419-4819-9_8.
- [8] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In J.-P. Jouannaud and Z. Shao, editors, *Certified Programs and Proofs*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-25378-2. . URL http://dx.doi.org/10.1007/978-3-642-25379-9_12.
- [9] A. Ben-Tal and A. S. Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001. ISBN 0-89871-491-5.
- [10] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. ISBN 9783540208549.
- [11] M. Berz and K. Makino. Rigorous global search using Taylor models. In *Proceedings of the 2009 conference on Symbolic numeric computation*, SNC '09, pages 11–20, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-664-9. . URL <http://doi.acm.org/10.1145/1577190.1577198>.
- [12] J. Bingham and J. Leslie-Hurd. Verifying Relative Error Bounds Using Symbolic Simulation. In A. Biere and R. Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 277–292. Springer International Publishing, 2014. ISBN 978-3-319-08866-2. . URL http://dx.doi.org/10.1007/978-3-319-08867-9_18.
- [13] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A Static Analyzer for Large Safety-Critical Software. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI'03)*, pages 196–207, San Diego, California, USA, June 7–14 2003. ACM Press.
- [14] D. Bolland and G. A. Constantinides. Automated precision analysis: A polynomial algebraic approach. In *FCCM'10*, pages 157–164, 2010.
- [15] S. Boldo and G. Melquiond. Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq. In *Computer Arithmetic (ARITH)*, 2011 20th IEEE Symposium on, pages 243–252, July 2011. .
- [16] M. Borges, M. d'Amorim, S. Anand, D. Bushnell, and C. S. Pasareanu. Symbolic execution with interval solving and meta-heuristic search. In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, ICST '12, pages 111–120, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4670-4. . URL <http://dx.doi.org/10.1109/ICST.2012.91>.
- [17] S. Boutin. Using Reflection to Build Efficient and Certified Decision Procedures. In *TACS'97. Springer-Verlag LNCS 1281*, pages 515–529. Springer-Verlag, 1997.
- [18] L. Chen, A. Miné, and P. Cousot. A Sound Floating-Point Polyhedra Abstract Domain. In G. Ramalingam, editor, *Programming Languages and Systems*, volume 5356 of *Lecture Notes in Computer Science*, pages 3–18. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-89329-5. . URL http://dx.doi.org/10.1007/978-3-540-89330-1_2.
- [19] L. Chen, A. Miné, J. Wang, and P. Cousot. Interval Polyhedra: An Abstract Domain to Infer Interval Linear Relationships. In J. Palsberg and Z. Su, editors, *Static Analysis*, volume 5673 of *Lecture Notes in Computer Science*, pages 309–325. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-03236-3. . URL http://dx.doi.org/10.1007/978-3-642-03237-0_21.
- [20] W.-F. Chiang, G. Gopalakrishnan, Z. Rakamaric, and A. Solovyev. Efficient search for inputs causing high floating-point errors. In *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPOPP '14, pages 43–52, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2656-8. . URL <http://doi.acm.org/10.1145/2555243.2555265>.
- [21] Coq. The Coq Proof Assistant. <http://coq.inria.fr/>.
- [22] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
- [23] E. Darulova and V. Kuncak. Sound Compilation of Reals. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 235–248, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2544-8. . URL <http://doi.acm.org/10.1145/2535838.2535874>.
- [24] M. Daumas and G. Melquiond. Certification of Bounds on Expressions Involving Rounded Operators. *ACM Trans. Math. Softw.*, 37(1):2:1–2:20, Jan. 2010. ISSN 0098-3500. . URL <http://doi.acm.org/10.1145/1644001.1644003>.
- [25] L. De Moura and N. Björner. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78799-2, 978-3-540-78799-0. URL <http://dl.acm.org/citation.cfm?id=1792734.1792766>.
- [26] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védryne. Towards an industrial use of fluctuat on safety-critical avionics software. In M. Alpuente, B. Cook, and C. Joubert, editors, *Formal Methods for Industrial Critical Systems*, volume 5825 of *Lecture Notes in Computer Science*, pages 53–69. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04569-1. . URL http://dx.doi.org/10.1007/978-3-642-04570-7_6.
- [27] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-53032-0.

- [28] S. Gao, S. Kong, and E. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In M. Bonacina, editor, *Automated Deduction – CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer Berlin Heidelberg, 2013. URL http://dx.doi.org/10.1007/978-3-642-38574-2_14.
- [29] X. Gao and G. A. Constantinides. Numerical Program Optimization for High-Level Synthesis. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 210–213, New York, NY, USA, 2015. ACM. URL <http://doi.acm.org/10.1145/2684746.2689090>.
- [30] K. Ghorbal, E. Goubault, and S. Putot. A logical product approach to zonotope intersection. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 212–226. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14294-9. URL http://dx.doi.org/10.1007/978-3-642-14295-6_22.
- [31] B. Grégoire and A. Mahboubi. Proving Equalities in a Commutative Ring Done Right in Coq. In J. Hurd and T. F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005. ISBN 3-540-28372-2.
- [32] T. C. Hales. Some algorithms arising in the proof of the kepler conjecture. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Discrete and Computational Geometry*, volume 25 of *Algorithms and Combinatorics*, pages 489–507. Springer Berlin Heidelberg, 2003. ISBN 978-3-642-62442-1. URL http://dx.doi.org/10.1007/978-3-642-55566-4_22.
- [33] T. C. Hales. Introduction to the flyspeck project. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. URL <http://drops.dagstuhl.de/opus/volltexte/2006/432>.
- [34] L. Haller, A. Griggio, M. Brain, and D. Kroening. Deciding floating-point logic with systematic abstraction. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 131–140, 2012. ISBN 978-1-4673-4832-4.
- [35] D. Handelman. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific Journal of Mathematics*, 132(1):35–62, 1988. URL <http://projecteuclid.org/euclid.pjm/1102689794>.
- [36] J. Harrison. HOL Light: A Tutorial Introduction. In M. K. Srivastava and A. J. Camilleri, editors, *FMCAD*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer, 1996. ISBN 3-540-61937-2.
- [37] J. Harrison. Formal verification of floating point trigonometric functions. In J. Hunt, Warren A. and S. Johnson, editors, *Formal Methods in Computer-Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 254–270. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41219-9. URL http://dx.doi.org/10.1007/3-540-40922-X_14.
- [38] N. Higham. *Accuracy and Stability of Numerical Algorithms: Second Edition*. Society for Industrial and Applied Mathematics, 2002. ISBN 9780898715217. URL <http://books.google.fr/books?id=epilvM5MMxwC>.
- [39] B. Jeannot and A. Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In A. Bouajjani and O. Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 661–667. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02657-7. URL http://dx.doi.org/10.1007/978-3-642-02658-4_52.
- [40] J.-L. Krivine. Quelques propriétés des préordres dans les anneaux commutatifs unitaires. *Comptes Rendus de l'Académie des Sciences*, 258:3417–3418, 1964.
- [41] J. Lasserre. *Moments, Positive Polynomials and Their Applications*. Imperial College Press optimization series. Imperial College Press, 2009. ISBN 9781848164468. URL <http://books.google.nl/books?id=VY6imTsdIrEC>.
- [42] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.
- [43] J. B. Lasserre. Convergent SDP-Relaxations in Polynomial Optimization with Sparsity. *SIAM Journal on Optimization*, 17(3):822–843, 2006.
- [44] J. B. Lasserre. A new look at nonnegativity on closed sets and polynomial optimization. 21(3):864–885, 2011. ISSN 1052-6234 (print), 1095-7189 (electronic). URL http://epubs.siam.org/siopt/resource/1/sjope8/v21/i3/p864_s1.
- [45] J. B. Lasserre and M. Putinar. Positivity and Optimization for Semi-Algebraic Functions. *SIAM Journal on Optimization*, 20(6):3364–3383, 2010.
- [46] M. D. Linderman, M. Ho, D. L. Dill, T. H. Meng, and G. P. Nolan. Towards Program Optimization Through Automated Analysis of Numerical Precision. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 230–237, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-635-9. URL <http://doi.acm.org/10.1145/1772954.1772987>.
- [47] V. Magron. *Formal Proofs for Global Optimization – Templates and Sums of Squares*. PhD thesis, Ecole Polytechnique, 2013.
- [48] V. Magron. NLCertify: A Tool for Formal Nonlinear Optimization. In H. Hong and C. Yap, editors, *Mathematical Software – ICMS 2014*, volume 8592 of *Lecture Notes in Computer Science*, pages 315–320. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44198-5. URL http://dx.doi.org/10.1007/978-3-662-44199-2_49.
- [49] V. Magron, X. Allamigeon, S. Gaubert, and B. Werner. Certification of real inequalities: templates and sums of squares. *Mathematical Programming*, 151(2):477–506, 2015. ISSN 0025-5610. URL <http://dx.doi.org/10.1007/s10107-014-0834-5>.
- [50] V. Magron, X. Allamigeon, S. Gaubert, and B. Werner. Formal proofs for Nonlinear Optimization. *Journal of Formalized Reasoning*, 8(1):1–24, 2015.
- [51] Maxima. Maxima, a Computer Algebra System. Version 5.30.0, 2013. URL <http://maxima.sourceforge.net/>.
- [52] W. M. McEneaney. *Max-plus methods for nonlinear control and estimation*. Systems & Control: Foundations & Applications. Birkhäuser Boston Inc., Boston, MA, 2006. ISBN 978-0-8176-3534-3; 0-8176-3534-3.
- [53] G. Melquiond. Floating-point arithmetic in the Coq system. *Information and Computation*, 216(0):14 – 23, 2012. ISSN 0890-5401. URL <http://www.sciencedirect.com/science/article/pii/S0890540112000739>. Special Issue: 8th Conference on Real Numbers and Computers.
- [54] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006. ISSN 1388-3690. URL <http://dx.doi.org/10.1007/s10990-006-8609-1>.
- [55] R. E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Department of Computer Science, Stanford University, 1962.
- [56] C. Muñoz and A. Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2):151–196, August 2013. URL <http://dx.doi.org/10.1007/s10817-012-9256-3>.
- [57] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A Prototype Verification System. In D. Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag. URL <http://www.csl.sri.com/papers/CADE92-pvs/>.

- [58] G. Paganelli and W. Ahrendt. Verifying (In-)Stability in Floating-Point Programs by Increasing Precision, Using SMT Solving. In N. Bjørner, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie, editors, *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2013, Timisoara, Romania, September 23-26, 2013*, pages 209–216. IEEE Computer Society, 2013. ISBN 978-1-4799-3035-7. . URL <http://dx.doi.org/10.1109/SYNASC.2013.35>.
- [59] C. Riener, T. Theobald, L. J. Andr en, and J. B. Lasserre. Exploiting Symmetries in SDP-Relaxations for Polynomial Optimization. *Mathematics of Operations Research*, 38(1): 122–141, 2013. . URL <http://dx.doi.org/10.1287/moor.1120.0558>.
- [60] P. R ummer and T. Wahl. An smt-lib theory of binary floating-point arithmetic. In *Informal proceedings of 8th International Workshop on Satisfiability Modulo Theories (SMT) at FLoC, Edinburgh, Scotland, 2010*.
- [61] A. Solovyev and T. C. Hales. Formal verification of nonlinear inequalities with taylor interval approximations. In G. Brat, N. Rungta, and A. Venet, editors, *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, volume 7871 of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2013. ISBN 978-3-642-38087-7. . URL http://dx.doi.org/10.1007/978-3-642-38088-4_26.
- [62] A. Solovyev, C. Jacobsen, Z. Rakamari c, and G. Gopalakrishnan. Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions. In *Proceedings of the 20th International Symposium on Formal Methods (FM)*, 2015. to appear.
- [63] J. Stolfi and L. de Figueiredo. An Introduction to Affine Arithmetic. *TEMA Tend. Mat. Apl. Comput.*, 4(3):297 – 312, 2003.
- [64] M. Todd. Semidefinite Optimization. *Acta Numerica*, 10: 515–560, 2001.
- [65] L. Vandenberghe and S. Boyd. Semidefinite Programming. *SIAM Review*, 38:49–95, 1994.
- [66] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of Squares and Semidefinite Programming Relaxations for Polynomial Optimization Problems with Structured Sparsity. *SIAM Journal on Optimization*, 17(1):218–242, 2006.
- [67] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs : SDPA7. Technical report, Dept. of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 2010. URL http://www.optimization-online.org/DB_FILE/2010/01/2531.pdf.
- [68] R. Zumkeller. *Rigorous Global Optimization*. PhD thesis,  cole Polytechnique, 2008.

Appendix A: Nonlinear Programs

```

let box_doppler1 u v T = [(-100, 100); (20, 20e3); (-30, 50)];
let obj_doppler1 u v T = [(let t1 = 331.4 + 0.6 * T in
-t1 * v / ((t1 + u) * (t1 + u)), 2.75e-13)];;

let box_doppler2 u v T = [(-125, 125); (15, 25e3); (-40, 60)];;
let obj_doppler2 u v T = [(let t1 = 331.4 + 0.6 * T in
-t1 * v / ((t1 + u) * (t1 + u)), 8.04e-13)];;

let box_doppler3 u v T = [(-30, 120); (320, 20300); (-50, 30)];;
let obj_doppler3 u v T = [(let t1 = 331.4 + 0.6 * T in
-t1 * v / ((t1 + u) * (t1 + u)), 1.37e-13)];;

let box_rigidbody1 x1 x2 x3 = [(-15, 15); (-15, 15); (-15, 15)];;
let obj_rigidbody1 x1 x2 x3 = [(-x1 * x2 - 2 * x2 * x3 - x1 - x3, 3.80e-13)];;

let box_rigidbody2 x1 x2 x3 = [(-15, 15); (-15, 15); (-15, 15)];;
let obj_rigidbody2 x1 x2 x3 = [(2 * x1 * x2 * x3 + 3 * x3 * x3
-x2 * x1 * x2 * x3 + 3 * x3 * x3 - x2, 3.98e-11)];;

let box_verhulst x = [(0.1, 0.3)];;
let obj_verhulst x = [(4 * x / (1 + (x/1.11)), 3.37e-16)];;

let box_carbonGas v = [(0.1, 0.5)];;
let obj_carbonGas v = [(let p = 3.5e7 in let a = 0.401 in
let b = 42.7e-6 in let t = 300 in let n = 1000 in
(p + a * (n/v) ** 2) * (v - n * b) - 1.3806503e-23 * n * t, 1.09e-8)];;

let box_predPrey x = [(0.1, 0.3)];;
let obj_predPrey x = [(4 * x * x / (1 + (x/1.11) ** 2), 1.57e-16)];;

let box_kepler0 x1 x2 x3 x4 x5 x6 =
[(4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36)];;
let obj_kepler0 x1 x2 x3 x4 x5 x6 = [(x2 * x5 + x3 * x6 - x2 * x3
-x5 * x6 + x1 * (-x1 + x2 + x3 - x4 + x5 + x6), 8.76e-14)];;

let box_kepler1 x1 x2 x3 x4 =
[(4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36)];;
let obj_kepler1 x1 x2 x3 x4 = [(x1 * x4 * (-x1 + x2 + x3 - x4)
+x2 * (x1 - x2 + x3 + x4) + x3 * (x1 + x2 - x3 + x4)
-x2 * x3 * x4 - x1 * x3 - x1 * x2 - x4, 3.90e-13)];;

let box_kepler2 x1 x2 x3 x4 x5 x6 =
[(4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36); (4, 6.36)];;
let obj_kepler2 x1 x2 x3 x4 x5 x6 = [(x1 * x4 * (-x1 + x2 + x3
-x4 + x5 + x6) + x2 * x5 * (x1 - x2 + x3 + x4 - x5 + x6)
+x3 * x6 * (x1 + x2 - x3 + x4 + x5 - x6) - x2 * x3 * x4
-x1 * x3 * x5 - x1 * x2 * x6 - x4 * x5 * x6, 1.90e-12)];;

let box_sineTaylor x = [(-1.57079632679, 1.57079632679)];;
let obj_sineTaylor x = [(x - (x * x * x) / 6.0
+(x * x * x * x * x) / 120.0
-(x * x * x * x * x * x * x) / 5040.0, 5.53e-16)];;

let box_sineOrder3 z = [(-2, 2)];;
let obj_sineOrder3 z = [(0.954929658551372 * z
- 0.12900613773279798 * (z * z * z), 6.68e-16)];;

let box_sqrt y = [(0, 1)];;
let obj_sqrt y = [(1.0 + 0.5 * y - 0.125 * y * y
+ 0.0625 * y * y * y - 0.0390625 * y * y * y * y, 7.56e-16)];;

let box_floudas1 x1 x2 x3 x4 x5 x6 =
[(0, 6); (0, 6); (1, 5); (0, 6); (1, 5); (0, 10)];;
let cstr_floudas1 x1 x2 x3 x4 x5 x6 =
[(x3 - 3) ** 2 + x4 - 4; (x5 - 3) ** 2 + x6 - 4;
2 - x1 + 3 * x2; 2 + x1 - x2; 6 - x1 - x2; x1 + x2 - 2];;
let obj_floudas1 x1 x2 x3 x4 x5 x6 = [(-25 * (x1 - 2) ** 2
-(x2 - 2) ** 2 - (x3 - 1) ** 2 - (x4 - 4) ** 2
-(x5 - 1) ** 2 - (x6 - 4) ** 2, 3.00e-13)];;

let box_floudas2 x1 x2 = [(0, 3); (0, 4)];;
let cstr_floudas2 x1 x2 = [
2 * x1 ** 4 - 8 * x1 ** 3 + 8 * x1 * x1 - x2;
4 * x1 ** 4 - 32 * x1 ** 3 + 88 * x1 * x1 - 96 * x1 + 36 - x2];;
let obj_floudas2 x1 x2 = [(-x1 - x2, 8.79e-16)];;

```

```

let box_floudas3 x1 x2 = [(0, 2); (0, 3)];;
let cstr_floudas3 x1 x2 = [-2 * x1 ** 4 + 2 - x2];;
let obj_floudas3 x1 x2 = [(-12 * x1 - 7 * x2 + x2 * x2, 7.33e-15)];;

let box_logexp x = [(-8, 8)];;
let obj_logexp x = [(log(1 + exp(x)), 1.65e-15)];;

let box_sphere x r y z = [(-10, 10); (0, 10);
(-1.570796, 1.570796); (-3.14159265, 3.14159265)];;
let obj_sphere x r y z = [(x + r * sin(y) * cos(z), 7.78e-15)];;

let box_hartman3 x1 x2 x3 = [(0, 1); (0, 1); (0, 1)];;
let obj_hartman3 x1 x2 x3 = [
let e1 = 3.0 * (x1 - 0.3689) ** 2 + 10.0 * (x2 - 0.117) ** 2
+ 30.0 * (x3 - 0.2673) ** 2 in
let e2 = 0.1 * (x1 - 0.4699) ** 2 + 10.0 * (x2 - 0.4387) ** 2
+ 35.0 * (x3 - 0.747) ** 2 in
let e3 = 3.0 * (x1 - 0.1091) ** 2 + 10.0 * (x2 - 0.8732) ** 2
+ 30.0 * (x3 - 0.5547) ** 2 in
let e4 = 0.1 * (x1 - 0.03815) ** 2 + 10.0 * (x2 - 0.5743) ** 2
+ 35.0 * (x3 - 0.8828) ** 2 in
-(1.0 * exp(-e1) + 1.2 * exp(-e2)
+ 3.0 * exp(-e3) + 3.2 * exp(-e4)), 1.50e-13)];;

let box_hartman6 x1 x2 x3 x4 x5 x6 =
[(0, 1); (0, 1); (0, 1); (0, 1); (0, 1); (0, 1)];;
let obj_hartman6 x1 x2 x3 x4 x5 x6 = [
let e1 = 10.0 * (x1 - 0.1312) ** 2 + 3.0 * (x2 - 0.1696) ** 2
+ 17.0 * (x3 - 0.5569) ** 2 + 3.5 * (x4 - 0.0124) ** 2
+ 1.7 * (x5 - 0.8283) ** 2 + 8.0 * (x6 - 0.5886) ** 2 in
let e2 = 0.05 * (x1 - 0.2329) ** 2 + 10.0 * (x2 - 0.4135) ** 2
+ 17.0 * (x3 - 0.8307) ** 2 + 0.1 * (x4 - 0.3736) ** 2
+ 8.0 * (x5 - 0.1004) ** 2 + 14.0 * (x6 - 0.9991) ** 2 in
let e3 = 3.0 * (x1 - 0.2348) ** 2 + 3.5 * (x2 - 0.1451) ** 2
+ 1.7 * (x3 - 0.3522) ** 2 + 10.0 * (x4 - 0.2883) ** 2
+ 17.0 * (x5 - 0.3047) ** 2 + 8.0 * (x6 - 0.665) ** 2 in
let e4 = 17.0 * (x1 - 0.4047) ** 2 + 8.0 * (x2 - 0.8828) ** 2
+ 0.05 * (x3 - 0.8732) ** 2 + 10.0 * (x4 - 0.5743) ** 2
+ 0.1 * (x5 - 0.1091) ** 2 + 14.0 * (x6 - 0.0381) ** 2 in
-(1.0 * exp(-e1) + 1.2 * exp(-e2)
+ 3.0 * exp(-e3) + 3.2 * exp(-e4)), 2.00e-13)];;

let box_cav10 x = [(0, 10)];;
let obj_cav10 x = [(if (x * x - x > 0) then x * 0.1
else x * x + 2, 2.91)];;

let box_perin xy = [(1, 7); (-2, 7)];;
let cstr_perin xy = [x - 1; y + 2; x - y; 5 - y - x];;
let obj_perin xy = [(if (x * x + y * y ≤ 4) then y * x
else 0, 2.01)];;

```