Affine Arithmetic and Applications to Real-Number Proving

Mariano M. Moscato[1], César A. Muñoz[2], and Andrew P. Smith[1]

[1] National Institute of Aerospace, Hampton VA 23666, USA,
{mariano.moscato, andrew.smith}@nianet.org,
[2] NASA Langley Research Center, Hampton VA 23681, USA,
cesar.a.munoz@nasa.gov

Abstract. Accuracy and correctness are central issues in numerical analysis. To address these issues, several self-validated computation methods have been proposed in the last fifty years. Their common goal is to provide rigorously correct enclosures for calculated values, sacrificing a measure of precision for correctness. Perhaps the most widely adopted enclosure method is interval arithmetic. Interval arithmetic performs well in a wide range of cases, but often produces excessively large overestimations, unless the domain is reduced in size, e.g., by subdivision. Many extensions of interval arithmetic have been developed in order to cope with this problem. Among them, affine arithmetic provides tighter estimations by taking into account linear correlations between operands. This paper presents a formalization of affine arithmetic, written in the Prototype Verification System (PVS), along with a formally verified branch-and-bound procedure implementing that model. This procedure and its correctness property enables the implementation of a PVS strategy for automatically computing upper and lower bounds of real-valued expressions that are provably correct up to a user-specified precision.

1 Introduction

Formal verification of safety-critical cyber-physical systems often requires proving formulas involving real-valued computations. At NASA, examples of such verification efforts include formally verified algorithms and operational concepts for the next generation of air traffic management systems [6, 16, 19]. Provably correct real-valued computations are also essential in areas such as analysis of floating point programs [1–3, 7], verification of numerical algorithms [9, 22], and in the formalization of mathematical results such as Kepler conjecture [8].

In general, the exact range of a nonlinear function of one or more variables over an interval domain cannot be determined in finite time. Enclosure methods are designed to provide sound intervals that are guaranteed to include, but may however overestimate, the true range of a nonlinear function over a bounded domain. More formally, given a function $f : \mathbb{R}^m \to \mathbb{R}$, an interval-valued function $F : \mathbb{IR}^m \to \mathbb{IR}$ is obtained, where $\mathbb{IR}$ denotes the set of closed non-empty real intervals, such that for all $\mathbf{V} \in \mathbb{IR}^m$

$$\mathbf{v} \in \mathbf{V} \implies f(\mathbf{v}) \in F(\mathbf{V}) \; . \tag{1}$$

Arithmetic may be performed on intervals, by providing definitions for elementary operators, logarithmic and trigonometric functions, and other real-valued functions that satisfy Formula (1). For example, if $\mathbf{a} = [\underline{a}, \overline{a}], \mathbf{b} = [\underline{b}, \overline{b}] \in \mathbb{IR}$, then $\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}]$. Such definitions yield an enclosure method called *interval arithmetic* [14]. A natural interval extension E of any real expression e is then obtained by recursively replacing in e each constant by an interval containing the constant, each variable by its interval range, and each operator and function by their interval equivalents. Formalizations of interval arithmetic are available in several interactive theorem provers [4, 13, 21]. These formalizations also include proof strategies for performing provably correct real-valued computations.

While correct, enclosure methods often provide imprecise calculations of expressions involving real-valued functions due to the fact that approximation errors quickly accumulate. To mitigate this problem, enclosure methods often rely on the following property, which must be satisfied by any interval-valued function F. For all $\mathbf{U}, \mathbf{V} \in \mathbb{IR}^m$,

$$\mathbf{U} \subseteq \mathbf{V} \implies F(\mathbf{U}) \subseteq F(\mathbf{V}) \ . \tag{2}$$

Formula (2) enables the use of domain subdivision techniques, whereby the starting domain is recursively subdivided into smaller box sub-domains, on which the enclosure methods provide suitable precision. *Branch and bound* is a recursive method for computing rigorous approximations that combines domain subdivision with pruning strategies. These strategies avoid unnecessary computations that do not improve already computed bounds. A formally verified branch and bound algorithm for generic enclosure methods is presented in [18].

It is well-known that enclosure methods such as interval arithmetic suffer from the *dependency problem*. This problem occurs when a real variable appears multiple times in an expression. In this case, large over-approximations may occur when each variable is treated as an independent interval. The dependency problem can be reduced by using enhanced data structures that, among other things, keep track of variable dependencies. In the case of polynomial and rational functions, a method based on *Bernstein polynomials* [12], provide better precision than interval arithmetic at the cost of increased computational time. Multivariate Bernstein polynomials and proof-producing strategies for rigorous approximation based on their properties are available in PVS [17].

The use of a particular enclosure method depends on a trade-off between precision and efficiency. At one extreme, interval arithmetic is computationally efficient but may produce imprecise bounds. At the other extreme, Bernstein polynomials offer precise bounds but they are computationally expensive. *Affine arithmetic* [5] is an enclosure method situated between these two extremes. By taking into account linear correlations between operands, affine arithmetic produces better estimates than interval arithmetic at a computational cost that is smaller than Bernstein polynomials.

This paper presents a deep embedding of affine arithmetic in the Prototype Verification System (PVS) [20] that includes addition, subtraction, multiplica-

tion, and power operation on variables. The embedding is used in an instantiation of a generic branch and bound algorithm [18] that yield a provably correct procedure for computing enclosures of polynomials with variables in interval domains. The formally verified branch and bound procedure is the foundation of a PVS proof strategy for mechanically and automatically finding lower and upper bounds for the minimum and maximum values of multivariate polynomials on a bounded domain.

The formal development presented in this paper is available as part of the NASA PVS Library.[3] All theorems presented in this paper are formally verified in PVS. For readability, standard mathematical notation is used throughout this paper. The reader is referred to the formal development for implementation details.

2 Affine Arithmetic

Affine arithmetic is a refinement of interval arithmetic that attempts to reduce the dependency problem by tracking linear dependencies between repeated variable instances and thereby retaining simple shape information. It is based on the idea that any real value a can be represented by an *affine form* $\widehat{a}$, defined as

$$\widehat{a} \stackrel{def}{=} a_{(0)} + \sum_{i=1}^{\infty} a_{(i)}\varepsilon_i \ , \tag{3}$$

where $\varepsilon_i \in [-1, 1]$, and $a_{(j)} \in \mathbb{R}$, with $j > 0$. It is assumed that the set of indices j such that $a_j \neq 0$ is finite. Henceforth, $\ell_{\widehat{a}}$ denotes the maximum element of that set or 0 if the set is empty. Each ε_i stands for an unknown error value introduced during the calculation of a. An affine form grows symmetrically around its *central value* $a_{(0)}$. Each $a_{(i)}$ represents the weight that the corresponding ε_i has on the overall uncertainty of a. The coefficients $a_{(i)}$ are called the *partial deviations* of the affine form.

There is a close relationship between affine forms and intervals. Given an affine form $\widehat{a}$ as in Formula (3), it is clear that the value a is included in the interval

$$[\widehat{a}] \stackrel{def}{=} \left[a_{(0)} - \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}|, a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}| \right] . \tag{4}$$

In fact, for every real value a' in $[\widehat{a}]$, there exists an assignment $\mathbf{N}$ of values from $[-1, 1]$ to each ε_i in $\widehat{a}$ such that $a' = \widehat{a}(\mathbf{N})$, where

$$\widehat{a}(\mathbf{N}) \stackrel{def}{=} a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} a_{(i)}\mathbf{N}(i) \ . \tag{5}$$

As stated in [23], the semantics of affine arithmetic rely on the existence of a single $\mathbf{N}$ for which a' is equal to the ideal real value a. This property is called the *fundamental invariant* of affine arithmetic.

[3] http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/.

Conversely, any given interval $\mathcal{I} = [u, v]$ is equivalent to the affine form

$$\mathsf{V}_k^{\mathcal{I}} \stackrel{def}{=} \frac{v+u}{2} + \frac{v-u}{2}\varepsilon_k \ , \tag{6}$$

where all the partial deviations, except for the k-th coefficient, are equal to 0.

Arithmetic operations on affine forms can be defined. Operations that are affine combinations of their arguments are called *affine operations*. In contrast to affine operations, non-affine operations may require the addition of noise symbols that do not appear in their operands. For simplicity, variables are indexed by positive natural numbers. It is assumed that the number of variables in any expression is at most m. Thus, the first m noise symbols are reserved for variables. Using this convention, the affine form of variable x_m ranging over $\mathcal{I}$ is conveniently defined as $\mathsf{V}_m^{\mathcal{I}}$. An index $k > \max(m, \ell_{\widehat{a}})$ is referred to as a *fresh index* with respect to $\widehat{a}$.

Addition and multiplication of scalars, as unrestricted subtractions and additions, are affine operations. They are defined as shown next. Given an affine form $\widehat{a} = a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} a_{(i)}\varepsilon_i$ for a as in Formula (3) and $c \in \mathbb{R}$,

$$\begin{aligned} \mathsf{add}_c(\widehat{a}) &\stackrel{def}{=} c + a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} a_{(i)}\varepsilon_i \ , \\ \mathsf{mul}_c(\widehat{a}) &\stackrel{def}{=} c \cdot a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} c \cdot a_{(i)}\varepsilon_i \ . \end{aligned} \tag{7}$$

The negation operation is defined as $\mathsf{neg}(\widehat{a}) \stackrel{def}{=} \mathsf{mul}_{-1}(\widehat{a})$.

Given another affine form $\widehat{b} = b_{(0)} + \sum_{i=1}^{\ell_{\widehat{b}}} b_{(i)}\varepsilon_i$ for some real value b,

$$\mathsf{add}(\widehat{a}, \widehat{b}) \stackrel{def}{=} \left(a_{(0)} + b_{(0)}\right) + \sum_{i=1}^{\max(\ell_{\widehat{a}}, \ell_{\widehat{b}})} \left(a_{(i)} + b_{(i)}\right)\varepsilon_i \ . \tag{8}$$

The subtraction operation is defined as $\mathsf{sub}(\widehat{a}, \widehat{b}) \stackrel{def}{=} \mathsf{add}(\widehat{a}, \mathsf{neg}(\widehat{b}))$.

Whilst affine operations can be performed without the introduction of additional error terms, i.e., without any extra overestimation, a suitable rigorous affine approximation must be used for each non-affine operation. Next, definitions for multiplication and power operation on single variables (instead of arbitrary expressions) are presented. Multiplication of two affine forms is defined as

$$\mathsf{mul}_k(\widehat{a}, \widehat{b}) \stackrel{def}{=} a_{(0)}b_{(0)} + \sum_{i=1}^{\max(\ell_{\widehat{a}}, \ell_{\widehat{b}})} (a_{(0)}b_{(i)} + a_{(i)}b_{(0)})\varepsilon_i + \varepsilon_k \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}| \sum_{i=1}^{\ell_{\widehat{b}}} |b_{(i)}| \ , \tag{9}$$

where k is a fresh noise index with respect to $\widehat{a}$ and $\widehat{b}$, i.e., $k > \max(m, \ell_{\widehat{a}}, \ell_{\widehat{b}})$.

Even though the power operation can be implemented by reducing it to successive multiplications, the following definition gives a better performing alternative for the case where a single variable is raised to a power. Given an elementary

affine form $\widehat{a} = a_{(0)} + a_{(m)}\varepsilon_m$ for a real value ranging over the interval $[\widehat{a}]$ and a collection of $n-2$ fresh noise indices I with respect to $\widehat{a}$,

$$\mathsf{pow}_I(\widehat{a}, n) \stackrel{def}{=} \begin{cases} 1 & \text{if } n = 0, \\ a_{(0)}^n + n\ a_{(0)}^{n-1} a_{(m)}\varepsilon_m + \sum_{k=2}^{n} \binom{n}{k} a_{(0)}^{n-k} a_{(m)}^k \varepsilon_{I_{k-2}} & \text{otherwise.} \end{cases} \quad (10)$$

As in the case of interval arithmetic, the affine arithmetic operations satisfy containment properties. However, it is not generally true that $c \in [\widehat{a}]$ and $d \in [\widehat{b}]$ implies $c \circ d \in [\widehat{a} \circ \widehat{b}]$, for an arbitrary affine operation $\circ$. The correctness of the containment properties depends on a careful management of the noise symbols.

In general, the implementation of non-affine operations such as transcendental functions requires a series expansion with a rigorous error term. A Chebyshev approximation is well-suited and is sometimes used.

3 Formalization in PVS

In PVS, an affine form $\widehat{\alpha}$, defined as in Formula (3), is represented by a record type that holds the central value $a_{(0)}$ and the list of coefficients $a_{(1)}, \ldots, a_{(n)}$. Since noise terms may be common to several affine forms, they are represented by an independent type `Noise` that denotes a mapping from positive natural numbers (the noise indices) to values in the interval $[-1, 1]$.

The following lemmas, which are proved in PVS, show the correctness of the affine arithmetic operations. In particular, they provide sufficient conditions for the containment properties to hold. For each lemma, the name of the corresponding PVS lemma is included in parentheses.

Lemma 1 (containment_interval). *Let* $\mathbf{N}$ *be a map of type* `Noise`*,* a *be a real number, and* $\widehat{a}$ *be an affine form of* a *over* $\mathbf{N}$*, i.e.,* $\widehat{a}\,(\mathbf{N}) = a$*. Then,* $a \in [\widehat{a}]$*.*

Proof. Using Formula (5), it has to be proved that $a = a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} a_{(i)}\mathbf{N}(i) \in [a_{(0)} - \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}|, a_{(0)} + \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}|]$. A known property from interval arithmetic called `containment_add` (formally proved in the development `interval_arith`, part of the NASA PVS Library) states that $u + v \in [u_1 + v_1, u_2 + v_2]$, when $u \in [u_1, u_2]$ and $v \in [v_1, v_2]$. Then it suffices to show that $a_{(0)} \in [a_{(0)}, a_{(0)}]$, which is trivially true, and $\sum_{i=1}^{\ell_{\widehat{a}}} a_{(i)}\mathbf{N}(i) \in [-\sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}|, \sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}|]$. This latter property can be proved by induction on $\ell_{\widehat{a}}$ and using `containment_add`. □

Lemma 2 (containment_var). *Let* $\mathbf{N}$ *be a map of type* `Noise`*,* k *be a natural number,* $\mathcal{I}$ *be a real interval, and* $v \in \mathcal{I}$*. There exists a real number* $b \in [-1, 1]$ *such that* $\mathsf{V}_k^{\mathcal{I}}(\mathbf{N}\ \mathtt{with}\ [n \mapsto b]) = v$ *.*

Proof. This lemma is proved by using the definition of $\mathsf{V}_k^{\mathcal{I}}$ in Formula (6). □

Lemma 3 (containment_aff_un). *Let* $\mathbf{N}$ *be a map of type* `Noise`*, and* a, c *be a pair of real numbers. Given* $\widehat{a} = a_{(0)} + \sum_{i=0}^{\ell_{\widehat{a}}} a_{(i)}\varepsilon_i$*, an affine form of* a *over* $\mathbf{N}$*,*

$$\mathsf{neg}(\widehat{a})\,(\mathbf{N}) = -a \quad \textit{and} \quad \mathsf{add}_c(\widehat{a})\,(\mathbf{N}) = c + a \quad \textit{and} \quad \mathsf{mul}_c(\widehat{a})\,(\mathbf{N}) = c \cdot a \ .$$

Proof. The equality $\mathsf{add}_c(\widehat{a})\,(\mathbf{N}) = c + a$ is a trivial consequence of Formula (5) and Formula (7). Meanwhile, $\mathsf{neg}(\widehat{a})\,(\mathbf{N}) = -a$ and $\mathsf{mul}_c(\widehat{a})\,(\mathbf{N}) = c \cdot a$ can be proved by induction on the length of the partial deviation of $\widehat{a}$. □

Lemma 4 (containment_aff_bin). *Let* $\mathbf{N}$ *be a map of type* `Noise` *and* a, b *be a pair of real numbers. Given* $\widehat{a} = a_{(0)} + \sum_{i=0}^{\ell_{\widehat{a}}} a_{(i)}\varepsilon_i$ *and* $\widehat{b} = b_{(0)} + \sum_{i=0}^{\ell_{\widehat{b}}} b_{(i)}\varepsilon_i$*, affine forms of* a *and* b*, resp., over* $\mathbf{N}$*, i.e.,* $\widehat{a}\,(\mathbf{N}) = a$ *and* $\widehat{b}\,(\mathbf{N}) = b$*,*

$$\mathsf{add}(\widehat{a}, \widehat{b})\,(\mathbf{N}) = a + b \quad \textit{and} \quad \mathsf{sub}(\widehat{a}, \widehat{b})\,(\mathbf{N}) = a - b \ .$$

Proof. Both properties can be proved by induction on the sum of the lengths of the partial deviations of $\widehat{a}$ and $\widehat{b}$. □

Non-affine operations introduce new noise symbols. The following lemma states that fresh noise symbols can be soundly added to any affine representation.

Lemma 5 (eval_updb_no_idxs). *Let* $\mathbf{N}$ *be a map of type* `Noise`*,* a *be a real number, and* $\widehat{a}$ *an affine form of* a *over* $\mathbf{N}$*, i.e.,* $\widehat{a}\,(\mathbf{N}) = a$*. For any collection* $\{i_k\}_{k=1}^{n}$ *of fresh indices with respect to* $\widehat{a}$ *and* $b_1, \ldots, b_n$ *real numbers in* $[-1, 1]$*,*

$$\widehat{a}\,(\mathbf{N}) = \widehat{a}\,(\mathbf{N}\ \texttt{with}\ [i_1 \mapsto b_1, \ldots, i_n \mapsto b_n]) \ .$$

Proof. The proof proceeds by induction on n. The interesting part is the base case ($n = 1$), which is proved by induction on $\ell_{\widehat{a}}$. That part of the proof relies on the fact that i_1 is a fresh index with respect to $\widehat{a}$. □

Henceforth, the notation $\mathbf{N}_p$, where p is a positive natural number, denotes the map that is equal to $\mathbf{N}$ in every index except in indices $i > p$, where $\mathbf{N}_p(i) = 0$.

Lemma 6 (containment_mul). *Let* $\mathbf{N}$ *be a map of type* `Noise` *and* a, b *be a pair of real numbers. Given* $\widehat{a} = a_{(0)} + \sum_{i=0}^{\ell_{\widehat{a}}} a_{(i)}\varepsilon_i$ *and* $\widehat{b} = b_{(0)} + \sum_{i=0}^{\ell_{\widehat{b}}} b_{(i)}\varepsilon_i$*, affine forms of* a *and* b *(resp.) over* $\mathbf{N}$*, for each index* $p > max(m, \ell_{\widehat{a}}, \ell_{\widehat{b}})$*, if*

$$\mathbf{N}(p) = \frac{\widehat{a}\,(\mathbf{N}_p)\,\widehat{b}\,(\mathbf{N}_p) - a_{(0)}b_{(0)} - \sum_{i=1}^{max(\ell_{\widehat{a}}, \ell_{\widehat{b}})}(a_{(0)}b_{(i)} + a_{(i)}b_{(0)})\mathbf{N}(i)}{\sum_{i=1}^{\ell_{\widehat{a}}} |a_{(i)}| \sum_{i=1}^{\ell_{\widehat{b}}} |b_{(i)}|} \quad ,$$

then $\mathsf{mul}_p(\widehat{a}, \widehat{b})\,(\mathbf{N}) = a \cdot b$*.*

Proof. It can be shown that $\mathsf{mul}_p(\widehat{a}, \widehat{b})\,(\mathbf{N}) = \widehat{a}\,(\mathbf{N}_p) \cdot \widehat{b}\,(\mathbf{N}_p)$, by applying arithmetic manipulations and using Formulas (5) and (9) and the hypothesis on $\mathbf{N}(p)$. By Lemma 5, since $p > \ell_{\widehat{a}}$, $\widehat{a}\,(\mathbf{N}_p) = \widehat{a}\,(\mathbf{N})$. Also, if $\widehat{a}$ is an affine form of a over $\mathbf{N}$, $\widehat{a}\,(\mathbf{N}) = a$. Then, $\widehat{a}\,(\mathbf{N}_p) = a$. Similarly, $\widehat{b}\,(\mathbf{N}_p) = b$. □

Lemma 7 (containment_pow_var_ac). *Let* $\mathbf{N}$ *be a map of type* `Noise`*,* n *be natural number,* a *be a real number, and* $\widehat{a} = a_{(0)} + a_{(l)}\varepsilon_l$ *an affine form of* a *over* $\mathbf{N}$*. Given* $\{i_k\}_{k=0}^{n-2}$*, a collection of fresh indices with respect to* $\widehat{a}$*, if*

$$\mathbf{N}(i_k) = \mathbf{N}(l)^{k+2} \quad \textit{for every } k, \textit{with } 0 \leq k \leq n$$

then $\mathsf{pow}_{\{i_k\}_{k=0}^{n-2}}(\widehat{a}, n)\,(\mathbf{N}) = a^n$ *.*

Proof. The proof proceeds by separating cases according to the definition of $\mathsf{pow}_{\{i_k\}_{k=0}^{n-2}}(\widehat{a}, n)$. The case $n = 0$ is trivial. When $n > 0$, using Formulas (5) and (10) and the hypothesis on $\mathbf{N}(i_k)$, it can be shown that $\mathsf{pow}_{\{i_k\}_{k=0}^{n-2}}(\widehat{a}, n)\,(\mathbf{N})$ is the combinatorial expansion of $\widehat{a}\,(\mathbf{N})^k$. The hypothesis assuring $\widehat{a}\,(\mathbf{N}) = a$ can be used to conclude the proof. □

In PVS, the fundamental property of affine arithmetic (Formula (2)) is proved on formal expressions containing constants, variables, addition, multiplication, and power operation on variables. Variables are indexed by positive natural numbers. A formal expression `e` represents a real number e by means of an evaluation function. More precisely, the PVS function eval_Γ from formal expressions into real numbers is recursively defined as follows, where Γ is a map from positive natural numbers, representing variable indices, into real values.

$$\begin{aligned}
\mathrm{eval}_\Gamma(v_i) &\stackrel{def}{=} \Gamma(i), \text{where } v_i \text{ represents the } i\text{-th variable},\\
\mathrm{eval}_\Gamma(\mathtt{c}) &\stackrel{def}{=} c, \text{where } \mathtt{c} \text{ represents the numerical constant } c,\\
\mathrm{eval}_\Gamma(-\mathtt{e}) &\stackrel{def}{=} -\mathrm{eval}_\Gamma(\mathtt{e}),\\
\mathrm{eval}_\Gamma(\mathtt{e}\ +\ \mathtt{f}) &\stackrel{def}{=} \mathrm{eval}_\Gamma(\mathtt{e}) + \mathrm{eval}_\Gamma(\mathtt{e}),\\
\mathrm{eval}_\Gamma(\mathtt{e}\ -\ \mathtt{f}) &\stackrel{def}{=} \mathrm{eval}_\Gamma(\mathtt{e}) - \mathrm{eval}_\Gamma(\mathtt{e}),\\
\mathrm{eval}_\Gamma(\mathtt{e}\ \times\ \mathtt{f}) &\stackrel{def}{=} \mathrm{eval}_\Gamma(\mathtt{e}) \cdot \mathrm{eval}_\Gamma(\mathtt{e}),\\
\mathrm{eval}_\Gamma(\mathtt{e}^n) &\stackrel{def}{=} \mathrm{eval}_\Gamma(\mathtt{e})^n\ .
\end{aligned} \tag{11}$$

Algorithm 1, which is formally defined in PVS, recursively constructs an affine form of a formal expression `e`. It has as parameters the formal expression `e` containing at most m variables, a collection $\{\mathcal{I}_i\}_{i=1}^m$ of m intervals (one per variable), and a map that caches affine forms of sub-expressions of `e`. It returns a map of all sub-expressions of `e`, including itself, to affine forms. The cache map ensures that noise symbol indices are shared among common sub-expressions. In the algorithm, the notation $I_{[0\ldots k]}$ stands for a collection containing the first $k+1$ indices in I, and $[a \ldots b]$ stands for the collection of consecutive indices from a to b.

The following theorem, which is proved in PVS, states the fundamental theorem of affine arithmetic.

Theorem 1. *Let* `e` *be a formal expression,* $\{\mathcal{I}_i\}_{i=1}^m$ *be a collection of intervals,* Γ *be map from variable indices in* `e` *to real numbers such that* $\Gamma(i) \in \mathcal{I}_i$*,* $e =$

```
1  RE2AF(e, {I_i}_{i=1}^m, cache)
2      if e is in cache then return cache;
3      else if e is the variable v_j then return cache with [e ↦ V_j^{I_j}];
4      else
5          do
6              switch e do
                   // Affine Operations
7                  case −e_1 return cache with [e ↦ neg(ê_1)];
8                  case e_1 + k or k + e_1 return cache with [e ↦ add_k(ê_1)];
9                  case e_1 × k or k × e_1 return cache with [e ↦ mul_k(ê_1)];
10                 case e_1 + e_2 return cache with [e ↦ add(ê_1, ê_2)];
11                 case e_1 − e_2 return cache with [e ↦ sub(ê_1, ê_2)];
                   // Non-Affine Operations
12                 case e_1 × e_2 return cache with [e ↦ mul_p(ê_1, ê_2)];
13                 case v_i^k
14                     if exists some (v_i^{k'} ↦ pow_I(V_j^{I_j}, k)) ∈ cache then
15                         if k < k' then
16                             return cache with [e ↦ pow_{I[0...k−2]}(V_j^{I_j}, k)];
17                         else
18                             return cache with [e ↦ pow_{I ⋃ [p...p+k−k']}(V_j^{I_j}, k)];
19                         end
20                     else return cache with [e ↦ pow_{[p...p+(k−2)]}(V_j^{I_j}, k)];
21                 end
22             endsw
23         where k, k' are constants expressions and e_1, e_2 are non-constant
           expressions and I is a collection of noise indices
           and p > max(m, the greatest noise index in cache)
           and cache_1 = RE2AF(e_1,{I_i}_{i=1}^m,cache) and ê_1 = cache_1(e_1)
           and cache_2 = RE2AF(e_2,{I_i}_{i=1}^m,cache_1) and ê_2 = cache_2(e_2);
24     end
25 end
```

Algorithm 1: Construction of affine forms of all sub-expressions in e

eval$_\Gamma$(e), *and* $\widehat{e} =$ RE2AF(e, $\{\mathcal{I}_j\}_{j=1}^m, \emptyset$)(e). *There exists a map* $\mathbf{N}$ *of type* `Noise` *such that* $\widehat{e}(\mathbf{N}) = e$.

Proof. The proof proceeds by structural induction on e of a more general statement, where the cache map may be non-empty. The proof of that statement uses the fact that every expression in the cache map is a sub-expression of e. Since this condition is encoded in the type of the parameter *cache*, it is guaranteed by the PVS type checker. The base cases are discharged with Lemmas 2–7. □

4 Proof Strategy

The motivation for the formalization of affine arithmetic presented in this paper is not only to verify the correctness of its operations, but more importantly to

develop a practical method for performing guaranteed computations inside a theorem prover. In particular, the following problem is considered.

> Given a polynomial expression p with variables $x_1, \ldots, x_m$ ranging over real intervals $\{\mathcal{I}_i\}_{i=1}^m$, respectively, and a positive natural number n (called *precision*) compute an interval enclosure $[u, v]$ for p, i.e., $p \in [u, v]$, that is provably correct up to the accuracy $\epsilon = 10^{-n}$, i.e., $v - \max(p) \leq \epsilon$ and $\min(p) - u \leq \epsilon$.

Using Algorithm 1 and Theorem 1, it is possible to construct an affine form $\widehat{p}$ of any polynomial expression p. Lemma 1 guarantees that the interval $[\widehat{p}]$, as defined in Formula (4), is a correct enclosure of p. This approach for computing correct polynomial enclosures can be easily automated in most theorem provers that support a soundness-preserving strategy language. However, this approach does not guarantee the quality of the enclosure.

As outlined in the introduction, a way to improve the quality of an enclosure consists in dividing the original range of the variables into smaller intervals and considering the union of all the enclosures computed on these smaller subdomains. This technique typically yields tighter range enclosures, albeit at computational cost.

The NASA PVS Library includes the formalization of a branch and bound algorithm that implements domain subdivision on a generic enclosure method [18]. The algorithm, namely `simple_bandb`, can be instantiated with concrete data types and heuristics for deciding the subdivision and pruning schemas. The instantiation presented here is similar to the one given in [18] using interval arithmetic. A simplified version of the signature of `simple_bandb` has as parameters a formal expression `e`, a domain `box` for the variables in `e`, an enclosure method `evaluate`, a subdivision schema `subdivide`, a function `combine` that combines results from recursive calls, and an upper bound `maxd` for the maximum recursion depth. The output of the algorithm is an interval indicating the maximum and the minimum of the values that the expression `e` takes over `box` and additional information regarding the performance of the algorithm such as number of subdivisions, maximum recursion depth, and precision of the solution.

Intervals are represented by the data type `Interval`. The parameter `box` is an element of type `Box`, which is a list of intervals. The abstract data type `RealExpr` is used to represent formal expressions such as `e`. All these types are available from the development `interval_arith` in the NASA PVS Library. The parameter *evaluate* corresponds to a generic enclosure method. In the case of affine arithmetic, that parameter corresponds to the following function.

$$\texttt{Eval(}e\texttt{,}box\texttt{)} \stackrel{def}{=} [\texttt{RE2AF(}e\texttt{, }box\texttt{, }\emptyset\texttt{)(e)}] \ . \tag{12}$$

The functions that correspond to the parameters `subdivide` and `combine` are defined as in [18]. The former takes as parameter a box and a natural number n, it returns two boxes, which only differ from the original box in the n-th interval. That interval is replaced in the first (resp. second) box by the lower (resp. upper)

half of the n-th interval in the original box. The latter function is just the union of two intervals.

The soundness property of `simple_bandb` is expressed in terms of the following predicate.

$$\mathit{sound?}(\mathtt{e}, \mathtt{box}, \mathcal{I}) \stackrel{def}{=} \forall \Gamma \in \mathit{box} \colon \mathrm{eval}_{\Gamma}(e) \in \mathcal{I} \ . \tag{13}$$

Corollary 1 in [18] states that when $\mathcal{I}$ is the interval returned by `simple_bandb` applied to `e`, `box`, `Eval`, `subdivide`, `combine`, and `maxd`, three specific properties suffice to prove $\mathit{sound?}(\mathtt{e}, \mathtt{box}, \mathcal{I})$. Two of them are properties about the functions `subdivide` and `combine`. As they are the same as the interval arithmetic instantiation of the generic algorithm [18], the proofs of these properties are also the same. The remaining property is stated below.

$$\forall\ \mathtt{box}, \mathtt{e} : \mathit{sound?}(\mathtt{e}, \mathtt{box}, \mathtt{Eval(}e\mathtt{,}\mathit{box}\mathtt{)}) \ .$$

This property follows directly from Theorem 1, Formula (12), and Formula (13). The development in [18] includes a more sophisticated algorithm `bandb` that has some additional parameters. These parameters, which do not affect the soundness of the algorithm, enable the specification of a pruning heuristic and early termination conditions.

The formalization presented in [18] includes infrastructure for developing strategies via computational reflection using a provably correct instantiation of the generic branch and bound algorithm. In particular, it includes a function, written in the PVS strategy language, for constructing a formal expression `e` of type `RealExpr` representing a PVS arithmetic expression e of type `real` and an element `box` of type `Box` that contains of the interval ranges of the variables in e. Based on that infrastructure, the development presented in this paper includes a proof-producing PVS strategy `aff-numerical` that computes probably correct bounds of polynomial expressions up to a user specified precision.

In its more general form, the strategy `aff-numerical` has as parameter an arithmetic expression e. It adds to the current proof sequent the hypothesis $e \in \mathcal{I}$, where $\mathcal{I}$ is an enclosure computed by the affine arithmetic instantiation of `bandb` on default parameters. Optional parameters that can be specified by the user include desired precision, upper bound to the maximum depth, and strategy for selecting the variable direction for the subdivision schema.

Example 1. The left-hand side of Figure 1 illustrates a PVS sequent involving the polynomial $P_1(x) = x^5 - 2x^3$, where the variable x is assumed to range over the open interval $(-1000, 0)$. In a sequent, the formulas appearing above the horizontal line are the *antecedent*, while the formulas appearing below the horizontal line are the *consequent*. The right-hand side of Figure 1 illustrates the sequent after the application of the proof command

```
(aff-numerical "x^5-2*x^3" :precision 3 :maxdepth 14)
```

This command introduces a new formula to the antecedent, i.e., sequent formula `{-1}`. The new formula states that $P_1(x) \in [-999998000000000, 1.066]$, when

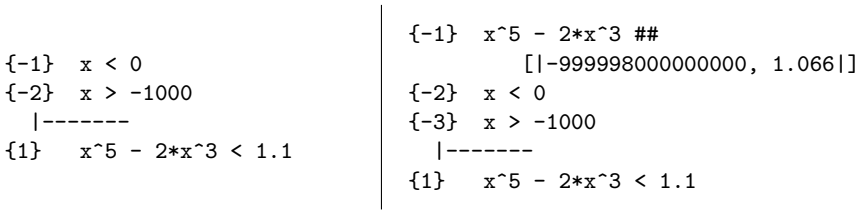

Fig. 1. Example of use of the strategy `aff-numerical`.

$x \in (-1000, 0)$. The sequent can be easily discharged by unfolding the definition of `##`, which stands for inclusion of a real number in an interval. The optional parameters `:precision` and `:maxdepth` in the strategy `aff-numerical` are used by the branch and bound algorithm to stop the recursion. In this case, the algorithm stops when either the enclosure is guaranteed to be accurate up to 10^{-3} or when a maximum depth of 14 is reached. The branch and bound algorithm uses rational arithmetic to compute enclosures. Since the upper and lower bounds of these enclosures tend to have large denominators and numerators, the strategy computes another enclosure whose upper and lower bounds are decimal numbers. These numbers are the closest decimal numbers, for the given precision, to the rational numbers of the original enclosure.

The strategy `aff-numerical` does not depend on external oracles since all the required theorems are proved within the PVS logic. Indeed, any particular instantiation of the strategy can be unfolded into a tree of proof commands that only includes PVS proof rules. The strategy does depend on the PVS ground evaluator [15], which is part of the PVS trusted code base, for the evaluation of the function branch and bound algorithm. It should be noted that while the soundness of the strategy depends on the correctness of the ground evaluator, the formal development presented in Section 3 does not. Furthermore, it is theoretically possible, although impractical, to replace every instance of the PVS ground evaluator in a proof by another strategy that only depends on deductive steps such as PVS's `grind`.

As part of the development presented in this paper, there is also available a strategy `aff-interval` that solves to a target enclosure or inequality as opposed to a target precision. For that kind of problem, `aff-interval` is more efficient than `aff-numerical`, since `aff-interval` takes advantage of early termination criteria, which are not available with `aff-numerical`.

5 Experimental Results

The objective of the experiments described in this section is to illustrate the performance of affine arithmetic compared to interval arithmetic using their PVS formalizations. The experiments use the strategies `numerical`, which is part of `interval_arith` in the NASA PVS Library, and `aff-numerical`, which

is part of the development presented in this paper. The strategies share most of the strategy code and only differ in the enclosure method.

Both strategies were used to find enclosures of polynomials with different characteristics. The performance was measured not only in terms of the time consumed by each strategy in every case but also with respect to the quality of the results. These experiments were performed on a desktop PC with an Intel Quad Core i5-4200U 1.60 GHz processor, 3.9 GiB of RAM, and 32-bit Ubuntu Linux.

The first part of this section presents the results obtained for polynomials in a single variable. The polynomials to be considered are $P_1 = x^5 - 2x^3$, from Example 1, and

$$P_2(x) = -\frac{10207769}{65536}x^{20} + \frac{3002285}{4096}x^{18} - \frac{95851899}{65536}x^{16} + \frac{6600165}{4096}x^{14} - \frac{35043645}{32768}x^{12} + \frac{1792791}{4096}x^{10} - \frac{3558555}{32768}x^8 + \frac{63063}{4096}x^6 - \frac{72765}{65536}x^4 + \frac{3969}{65536},$$

where $x \in (-1, 1)$. Turan's inequality for Legendre polynomials states that, for $x \in (-1, 1)$, $L_j(x)^2 > L_{j-1}(x)L_{j+1}(x)$ for all $x \in (-1, 1)$, where L_i stands for the i-th Legendre polynomial. The formula $P_2(x) > 0$ states Turan's inequality for $j = 10$.

The result of the comparison of the two enclosure methods using these examples is depicted in Fig. 2. The top graphic shows the magnitude of the overestimation produced by both strategies for each maximum subdivision depth (up to 43). In the bottom graphic, the y axis represents the time spent by the strategies for both examples and each depth.

The expected behavior of the affine arithmetic method — second-order convergence in overestimation with respect to depth, as compared to first-order for interval arithmetic — can be clearly seen in these graphs. For both P_1 and P_2 the rate of convergence for affine arithmetic method is significantly faster. Regarding P_1, even though the leftmost result is significantly worse than the one given by the interval arithmetic method, the convergence of the former is so fast that it reaches its best approximation at depth 16, while the latter needs a depth of 43 to reach an equivalent result. The difference in performance for P_2 is even sharper: only a depth of 4 is needed for the affine method to achieve its best result, which could not be matched with a depth below 20 for interval arithmetic.

For a given depth, the computation time for the affine method is always higher than the time for the interval method. Nevertheless, when considering the quality of the result, the former achieves a much better performance in the same amount of time.

The difference in the performance for P_2 is mostly due to the high level of dependency in the polynomial. In the corresponding affine form almost every sub-expression has noise terms shared with others in the expression. This sharing constrains the overall number of noise terms and tracks the dependencies to a considerable extent, allowing the affine method to reach much better results in less time.

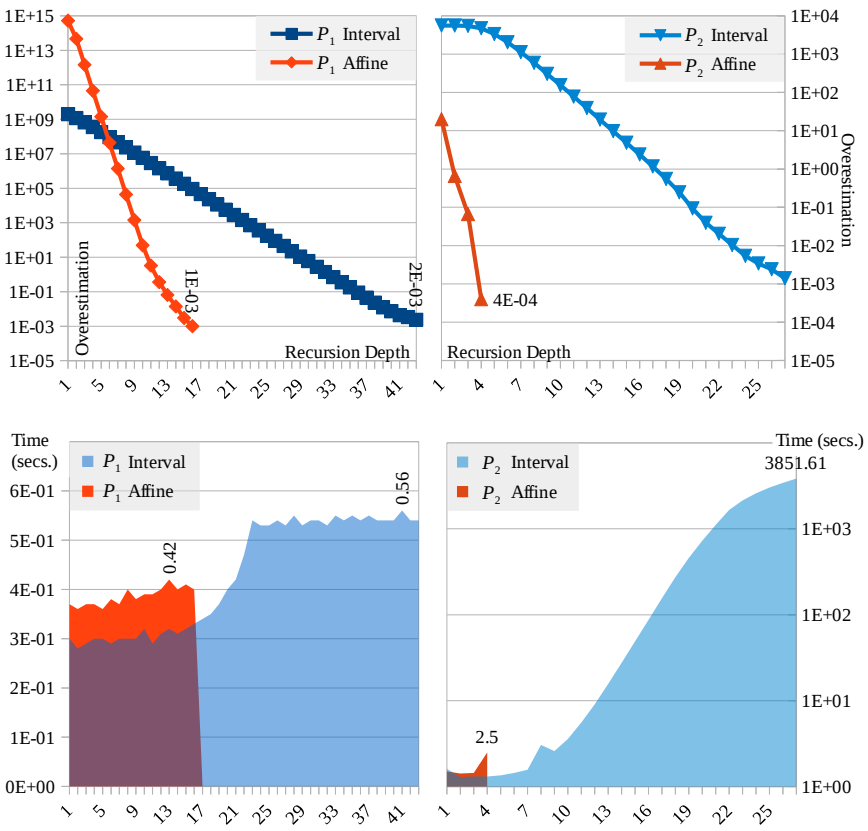

Fig. 2. Convergence rate for affine and interval arithmetic methods in P_1 and P_2

Finally, both enclosure methods are tested on the following non-trivial multivariate polynomial.

$$P_3(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7) = -x_0x_5^3 + 3x_0x_5x_6^2 - x_2x_6^3 + 3x_2x_6x_5^2 - x_1x_4^3 \\ +3x_1x_4x_7^2 - x_3x_7^3 + 3x_3x_7x_4^2 - 0.9563453\ ,$$

where $x_0 \in [-10, 40]$, $x_1 \in [40, 100]$, $x_2 \in [-70, -40]$, $x_3 \in [-70, 40]$, $x_4 \in [10, 20]$, $x_5 \in [-10, 20]$, $x_6 \in [-30, 110]$, and $x_7 \in [-110, -30]$. This polynomial is taken from a database of demanding test problems for global optimization algorithms [24].

As shown in Table 1, both methods have similar results in this case. Despite starting with a better result (as in P_2) the interval method is overpassed by the affine method when the depth is set to 30. Nevertheless, the time taken by the latter is almost the double of the consumed by the former at this depth. The affine method is likely to perform better for smaller boxes or at even greater depth.

max depth	Interval result	Interval time	Affine result	Affine time
10	$[-895381 \times 10^2, 247381 \times 10^3]$	3.63 s.	$[-951004 \times 10^2, 247079 \times 10^3]$	9.2 s.
20	$[-824131 \times 10^2, 242342 \times 10^3]$	16.91 s.	$[-833045 \times 10^2, 236464 \times 10^3]$	55.96 s.
25	$[-800610 \times 10^2, 238942 \times 10^3]$	54.09 s.	$[-804002 \times 10^2, 233724 \times 10^3]$	150.87 s.
30	$[-794831 \times 10^2, 237518 \times 10^3]$	157.22 s.	$[-796329 \times 10^2, 233405 \times 10^3]$	281.45 s.
35	$[-793982 \times 10^2, 235733 \times 10^3]$	322.54 s.	$[-790525 \times 10^2, 232709 \times 10^3]$	530.89 s.
40	$[-791026 \times 10^2, 234318 \times 10^3]$	755.40 s.	$[-789961 \times 10^2, 232657 \times 10^3]$	963.66 s.
45	$[-790319 \times 10^2, 233936 \times 10^3]$	1524.40 s.	$[-789319 \times 10^2, 232480 \times 10^3]$	1627.00 s.

Table 1. Data from P_3.

6 Conclusion and Further Work

The main contribution of this paper is a formalization of the affine arithmetic model [23] for self-validated numerical analysis. Although it has been performed in the Prototype Verification System (PVS), it does not depend on any specific feature of that system. The techniques presented in this paper could be implemented in any theorem prover with similar characteristics to PVS such as Coq, HOL, among others. Additionally, a proof-producing strategy for computing provably correct bounds of polynomials with variables over bounded domains was developed. This strategy relies heavily on the generic branch and bound algorithm introduced in [18]. The entire formalization, which is called `affine`, is available as part of the NASA PVS Library. The PVS formalization is organized into 11 theories, including the proofs of 193 properties and 483 non-trivial proof obligations automatically generated from typing conditions.

The performance of affine arithmetic is compared to interval arithmetic on some test cases using the PVS strategies developed for both enclosure methods. These experiments illustrate that, when dealing with problems with a high level of coupling between sub-expressions, the affine method performs significantly better than the interval method. The observed second-order rate of overestimation convergence for affine arithmetic accords with the theoretical result. In the presence of non-trivial functions over many variables, for a wide initial domain, it is possible that a large subdivision depth is necessary in order to realize this convergence.

Immler presents a formalization of affine arithmetic in Isabelle and uses it as part of developments intended to solve ordinary differential equations [9] and to calculate intersections between zonotopes and hyperplanes [10]. A minor difference with respect to the work presented in this paper is that that formalization considers the multiplication inverse, which is not considered here, but it does not consider the power operation. The authors are not aware of any formalization of a subdivision technique or the development of proof-producing strategies using affine arithmetic as the one presented in the current paper. There are implementations of affine arithmetic in C and C++ [11], but a comparison with these tools would be also unfair since these are non-formal compiled codes, whereas the developed affine arithmetic strategy presented in this paper yields a formal proof.

The current approach only supports polynomial expressions. This support can easily be extended to a larger set of real-valued functions when affine forms for these functions are implemented. The performance of the proposed approach could be improved in several ways. The definition of Algorithm 1 uses simple data structures, which could be replaced by better-performing ones. Furthermore, the algorithm could also take advantage of some of the improvements proposed to the basic model of affine arithmetic. A comprehensive survey of such improvements can be found in [11]. Another well-known way to achieve better results is to combine both interval and affine methods by first applying the more efficient interval arithmetic approach and, after a certain subdivision depth, to take advantage of the better convergence rate of the affine algorithm.

References

1. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Wave equation numerical resolution: A comprehensive mechanized proof of a C program. Journal of Automated Reasoning 50(4), 423–456 (Apr 2013), http://hal.inria.fr/hal-00649240/en/
2. Boldo, S., Clément, F., Filliâtre, J.C., Mayero, M., Melquiond, G., Weis, P.: Trusting computations: a mechanized proof from partial differential equations to actual program. Computers and Mathematics with Applications 68(3), 325–352 (2014), http://www.sciencedirect.com/science/article/pii/S0898122114002636
3. Boldo, S., Marché, C.: Formal verification of numerical programs: from C annotated programs to mechanical proofs. Mathematics in Computer Science 5, 377–393 (2011), http://dx.doi.org/10.1007/s11786-011-0099-9
4. Daumas, M., Lester, D., Muñoz, C.: Verified real number calculations: A library for interval arithmetic. IEEE Transactions on Computers 58(2), 1–12 (February 2009)
5. de Figueiredo, L.H., Stolfi, J.: Affine arithmetic: Concepts and applications. Numerical Algorithms 37(1-4), 147–158 (2004)
6. Galdino, A., Muñoz, C., Ayala, M.: Formal verification of an optimal air traffic conflict resolution and recovery algorithm. In: Leivant, D., de Queiroz, R. (eds.) Proceedings of the 14th Workshop on Logic, Language, Information and Computation. Lecture Notes in Computer Science, vol. 4576, pp. 177–188. Springer-Verlag, Rio de Janeiro, Brazil (July 2007)
7. Goodloe, A., Muñoz, C., Kirchner, F., Correnson, L.: Verification of numerical programs: From real numbers to floating point numbers. In: Brat, G., Rungta, N., Venet, A. (eds.) Proceedings of the 5th NASA Formal Methods Symposium (NFM 2013). Lecture Notes in Computer Science, vol. 7871, pp. 441–446. Springer, Moffett Field, CA (May 2013)
8. Hales, T., Adams, M., Bauer, G., Tat Dang, D., Harrison, J., Le Hoang, T., Kaliszyk, C., Magron, V., McLaughlin, S., Tat Nguyen, T., Quang Nguyen, T., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Hoai Thi Ta, A., Tran, T.N., Thi Trieu, D., Urban, J., Khac Vu, K., Zumkeller, R.: A formal proof of the Kepler conjecture. ArXiv e-prints (Jan 2015)
9. Immler, F.: Formally verified computation of enclosures of solutions of ordinary differential equations. In: Badger, J.M., Rozier, K.Y. (eds.) NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May

1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8430, pp. 113–127. Springer (2014), http://dx.doi.org/10.1007/978-3-319-06200-6
10. Immler, F.: A verified algorithm for geometric zonotope/hyperplane intersection. In: Proceedings of the 2015 Conference on Certified Programs and Proofs (CPP). pp. 129–136. ACM, New York, NY, USA (2015), http://doi.acm.org/10.1145/2676724.2693164
11. Kiel, S.: Yalaa: Yet another library for affine arithmetic. Reliable Computing 16, 114–129 (2012)
12. Lorentz, G.G.: Bernstein Polynomials. Chelsea Publishing Company, New York, N.Y., second edn. (1986)
13. Melquiond, G.: Proving bounds on real-valued functions with computations. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Proceedings of the 4th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, vol. 5195, pp. 2–17. Sydney, Australia (2008)
14. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM, Philadelphia (2009)
15. Muñoz, C.: Rapid prototyping in PVS. Contractor Report NASA/CR-2003-212418, NASA, Langley Research Center, Hampton VA 23681-2199, USA (May 2003)
16. Muñoz, C., Carreño, V., Dowek, G., Butler, R.: Formal verification of conflict detection algorithms. International Journal on Software Tools for Technology Transfer 4(3), 371–380 (2003)
17. Muñoz, C., Narkawicz, A.: Formalization of a representation of Bernstein polynomials and applications to global optimization. Journal of Automated Reasoning 51(2), 151–196 (August 2013), http://dx.doi.org/10.1007/s10817-012-9256-3
18. Narkawicz, A., Muñoz, C.: A formally verified generic branching algorithm for global optimization. In: Cohen, E., Rybalchenko, A. (eds.) Proceedings of the 5th International Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2013). Lecture Notes in Computer Science, vol. 8164, pp. 326–343. Springer, Menlo Park, CA, US (May 2014)
19. Narkawicz, A., Muñoz, C., Dowek, G.: Provably correct conflict prevention bands algorithms. Science of Computer Programming 77(1–2), 1039–1057 (September 2012), http://dx.doi.org/10.1016/j.scico.2011.07.002
20. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) Proceedings of the 11th International Conference on Automated Deduction (CADE). Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer (June 1992)
21. Solovyev, A., Hales, T.C.: Formal verification of nonlinear inequalities with Taylor interval approximations. In: Brat, G., Rungta, N., Venet, A. (eds.) Proceedings of the 5th International Symposium NASA Formal Methods. Lecture Notes in Computer Science, vol. 7871, pp. 383–397 (2013)
22. Solovyev, A., Hales, T.: Efficient formal verification of bounds of linear programs. In: Davenport, J., Farmer, W., Urban, J., Rabe, F. (eds.) Intelligent Computer Mathematics, Lecture Notes in Computer Science, vol. 6824, pp. 123–132. Springer Berlin Heidelberg (2011)
23. Stolfi, J., Figueiredo, L.H.D.: Self-validated numerical methods and applications (1997)
24. Verschelde, J.: Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. ACM Transactions on Mathematical Software 25(2), 251–276 (1999)