

Formalization of an Efficient Representation of Bernstein Polynomials and Applications to Global Optimization

César Muñoz · Anthony Narkawicz

Received: date / Accepted: date

Abstract This paper presents a formalization in higher-order logic of an efficient representation of multivariate Bernstein polynomials. Using this representation, an algorithm for finding lower and upper bounds of the minimum and maximum values of a polynomial has been formalized and verified correct in the Prototype Verification System (PVS). The algorithm is used in the definition of proof strategies for formally and automatically solving polynomial global optimization problems.

1 Introduction

Many engineering problems require determining whether, given bounds on the variables of a multivariate polynomial, the output values of the polynomial always fall within a particular range. These types of problems are called *polynomial global optimization* problems. For example, a common problem used as a test for global optimization algorithms is the Heart Dipole problem [20]. This problem can be reduced to minimizing the following polynomial on variables $x_1 \in [-0.1, 0.4]$, $x_2 \in [0.4, 1]$, $x_3 \in [-0.7, -0.4]$, $x_4 \in [-0.7, 0.4]$, $x_5 \in [0.1, 0.2]$, $x_6 \in [-0.1, 0.2]$, $x_7 \in [-0.3, 1.1]$, and $x_8 \in [-1.1, -0.3]$:

$$\begin{aligned} -x_1x_6^3 + 3x_1x_6x_7^2 - x_3x_7^3 + 3x_3x_7x_6^2 - x_2x_5^3 + 3x_2x_5x_8^2 - x_4x_8^3 + \\ 3x_4x_8x_5^2 - 0.9563453. \end{aligned} \quad (1)$$

The minimum of the polynomial over this range is approximately -1.7434. The tools presented in this paper can be used to *automatically* and *formally* prove that that this polynomial always takes values greater than -1.7435 and that it achieves a value less than -1.7434 in this range.

Global optimization problems appear in critical applications such as air traffic conflict detection and resolution algorithms [12], floating point analysis [6], and uncertainty and reliability analysis of dynamic and control systems [4], [9]. Finding precise bounds to the minimum and maximum values of a function is fundamental to the safety of these applications.

Polynomial global optimization belongs to the category of *non-linear arithmetic* problems. Higher-order logic based theorem provers such as Coq and HOL Light, and specialized theorem provers such as MetiTarski and RAHD have integrated decision procedures for non-linear arithmetic based on quantifier elimination [1, 14, 15, 19]. Although these procedures can handle formulas with complicated logical structures and, in some cases, transcendental functions, quantifier elimination is practical for polynomials with a small number of variables. Verification techniques based on equantifier elimination are currently not able to handle the polynomial in Formula (1).

Bernstein polynomials, which are related to Bézier curves, can be used to determine tight bounds on the range of a multivariate polynomial over a closed rectangle. Hence, they are well-known tools for global optimization [7, 8] and numerical approximation [13]. In the context of formal methods, single and multivariate Bernstein polynomials have been formalized in the Coq theorem prover [3, 23]. In [23], properties of multivariate Bernstein polynomials, including differentiation and integration, have been mechanically verified. That work also includes strategies for solving global optimization problems based on a branch-and-bound algorithm. However, the correctness of that algorithm is not formally proved in Coq. In [3], a mechanized proof of an algorithm for computing Bernstein coefficients is presented. That algorithm is used to formally prove a criterion for the existence of a root of single variable separable polynomials in a bounded interval.

This paper presents a formalization of a representation of Bernstein polynomials in the higher-order logic of the Prototype Verification System (PVS) [17]. Based on this representation, algorithms for global optimization are formalized and verified in PVS. These algorithms are based on recent branch and bound techniques [20] and clever data structures for representing polynomials [21] that have made Bernstein polynomials efficient tools for global optimization. The formally verified algorithms are the building blocks of proof strategies for mechanically and automatically finding lower and upper bounds for the minimum and maximum values of a polynomial and solving simply quantified polynomial inequalities. As far as the authors know, the algorithms presented in this paper are the first algorithms for multivariate global optimization based on Bernstein polynomials that have been *completely* formally verified in an automated theorem prover, in this case PVS.

The rest of the paper is organized as follows. Section 2 gives a general overview of multivariate Bernstein polynomials and their main properties. The formalization of the polynomial representation and the algorithms for global optimization are presented in sections 3 and 4, respectively. Section 5 presents automated strategies for solving polynomial global optimization problems in

PVS and illustrates their use with a few examples. The last section concludes this paper.

The formal development presented in this paper is electronically available from <http://shemesh.larc.nas.gov/people/cam/Bernstein>. 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 Bernstein Polynomials

This section introduces multivariate Bernstein polynomials and presents their main properties. All properties presented in this section have been formally proved in PVS for both univariate and multivariate polynomials. Later sections in this paper provide the actual statements of these properties in PVS. In order to distinguish the mathematical properties from the formal theorems in PVS, the former are called *propositions* and the later are called *theorems*. The formal proofs closely follow the proofs presented here.

Tuples will be typed in lowercase **boldface** and subindices from 0 to $m-1$, where $m > 0$, are used to denote particular elements of an m -tuple, e.g., $\mathbf{a} = (a_0, \dots, a_{m-1})$. Given a positive natural number m , the order $<$ between m -tuples is defined by $\mathbf{a} < \mathbf{b}$ if and only if $a_j < b_j$ for all $0 \leq j < m$. Similarly, the order \leq between m -tuples is defined by $\mathbf{a} \leq \mathbf{b}$ if and only if $a_j \leq b_j$ for all $0 \leq j < m$.

A *bounded box* $[\mathbf{a}, \mathbf{b}]$, where $\mathbf{a} < \mathbf{b}$, denotes the set of m -tuples greater than or equal to \mathbf{a} and less than or equal to \mathbf{b} . The box $[\mathbf{0}, \mathbf{1}]$, where $\mathbf{0} = (0, \dots, 0)$ and $\mathbf{1} = (1, \dots, 1)$, is called the *unit box*.

Let \mathbf{i} be an m -tuple of natural numbers and \mathbf{x} be an m -tuple of variables over \mathbb{R} . The product

$$\mathbf{x}^{\mathbf{i}} = x_0^{i_0} \dots x_{m-1}^{i_{m-1}}$$

is called a *multivariate monomial*. The m -tuple \mathbf{i} is called the *index* of the monomial $\mathbf{x}^{\mathbf{i}}$. A *multivariate polynomial* of degree \mathbf{n} is a finite sum of the form

$$p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \mathbf{x}^{\mathbf{i}},$$

where the elements $c_{\mathbf{i}} \in \mathbb{R}$ are called the *coefficients* of p . Such a polynomial p can be viewed as a function from \mathbb{R}^m into \mathbb{R} .

Several properties in this section are given for polynomials on the unit box. The following proposition states that any multivariate polynomial on an arbitrary box can be transformed into a polynomial on the unit box.

Proposition 1 *Let $[\mathbf{a}, \mathbf{b}]$ be a bounded box, $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \mathbf{x}^{\mathbf{i}}$, and $\sigma: [\mathbf{0}, \mathbf{1}] \rightarrow [\mathbf{a}, \mathbf{b}]$ be defined by*

$$\sigma(\mathbf{x})_j = a_j + x_j(b_j - a_j).$$

For all $\mathbf{x} \in [0, 1]$, $p(\sigma(\mathbf{x})) = p^*(\mathbf{x})$, where $p^*(\mathbf{x}) = \sum_{\mathbf{k} \leq \mathbf{n}} r_{\mathbf{k}} \mathbf{x}^{\mathbf{k}}$ and

$$r_{\mathbf{k}} = \sum_{\mathbf{k} \leq \mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \prod_{j=0}^{m-1} \binom{i_j}{k_j} (b_j - a_j)^{k_j} a_j^{i_j - k_j}.$$

Furthermore, since σ is a bijection, for all $\mathbf{y} \in [\mathbf{a}, \mathbf{b}]$, $p(\mathbf{y}) = p^*(\sigma^{-1}(\mathbf{y}))$.

Proof By the binomial theorem,

$$\begin{aligned} p(\sigma(\mathbf{x})) &= \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \prod_{j=0}^{m-1} (a_j + x_j(b_j - a_j))^{i_j} \\ &= \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \prod_{j=0}^{m-1} \sum_{k_j=0}^{i_j} \binom{i_j}{k_j} (b_j - a_j)^{k_j} a_j^{i_j - k_j} x^{k_j} \\ &= \sum_{\mathbf{i} \leq \mathbf{n}} \sum_{\mathbf{k} \leq \mathbf{i}} c_{\mathbf{i}} \prod_{j=0}^{m-1} \binom{i_j}{k_j} (b_j - a_j)^{k_j} a_j^{i_j - k_j} x^{k_j} \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \left(\sum_{\mathbf{k} \leq \mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \prod_{j=0}^{m-1} \binom{i_j}{k_j} (b_j - a_j)^{k_j} a_j^{i_j - k_j} \right) x^{k_j} \\ &= p^*(\mathbf{x}). \end{aligned}$$

2.1 Bernstein Basis

A Bernstein polynomial is a multivariate polynomial of the form

$$p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n}, \mathbf{i}}(\mathbf{x}), \quad (2)$$

where $\hat{b}_{\mathbf{i}} \in \mathbb{R}$ and

$$B_{\mathbf{n}, \mathbf{i}}(\mathbf{x}) = \prod_{j=0}^{m-1} \binom{n_j}{i_j} x_j^{i_j} (1 - x_j)^{n_j - i_j}. \quad (3)$$

The coefficients $\hat{b}_{\mathbf{i}}$ are called the Bernstein coefficients of p .

The multivariate polynomials $B_{\mathbf{n}, \mathbf{i}}(\mathbf{x})$ in Formula (3) form a basis for the vector space of polynomials of degree \mathbf{n} . As the following proposition states, any polynomial can be written as a polynomial in Bernstein form by a simple transformation.

Proposition 2 Any multivariate polynomial $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \mathbf{x}^{\mathbf{i}}$ can be written in Bernstein form as $p(\mathbf{x}) = \sum_{\mathbf{k} \leq \mathbf{n}} \hat{b}_{\mathbf{k}} B_{\mathbf{n}, \mathbf{k}}(\mathbf{x})$, where

$$\hat{b}_{\mathbf{k}} = \sum_{\mathbf{i} \leq \mathbf{k}} \left(c_{\mathbf{i}} \prod_{j=0}^{m-1} \frac{\binom{k_j}{i_j}}{\binom{n_j}{i_j}} \right).$$

Proof The *trinomial revision* formula states that $\binom{k}{i}\binom{n}{k}/\binom{n}{i} = \binom{n-i}{k-i}$ for all natural numbers i, k , and n such that $i \leq k \leq n$. Thus, if \mathbf{i} and \mathbf{n} are m -tuples of natural numbers such that $\mathbf{i} \leq \mathbf{n}$, then for all $j < m$, by the binomial theorem,

$$\begin{aligned} x_j^{i_j} &= x_j^{i_j} (x_j + (1 - x_j))^{n_j - i_j} \\ &= x_j^{i_j} \sum_{k_j=0}^{n_j - i_j} \binom{n_j - i_j}{k_j} x_j^{k_j} (1 - x_j)^{n_j - i_j - k_j} \\ &= \sum_{k_j=i_j}^{n_j} \binom{n_j - i_j}{k_j - i_j} x_j^{k_j} (1 - x_j)^{n_j - k_j} \\ &= \sum_{k_j=0}^{n_j} \frac{\binom{k_j}{i_j}}{\binom{n_j}{i_j}} \left(\binom{n_j}{k_j} x_j^{k_j} (1 - x_j)^{n_j - k_j} \right) \end{aligned}$$

Thus, the multivariate monomial $\mathbf{x}^{\mathbf{i}}$ can be written in Bernstein form as follows.

$$\begin{aligned} \mathbf{x}^{\mathbf{i}} &= \prod_{j=0}^{m-1} \left(\sum_{k_j=0}^{n_j} \frac{\binom{k_j}{i_j}}{\binom{n_j}{i_j}} \left(\binom{n_j}{k_j} x_j^{k_j} (1 - x_j)^{n_j - k_j} \right) \right) \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \left(\prod_{j=0}^{m-1} \frac{\binom{k_j}{i_j}}{\binom{n_j}{i_j}} \right) B_{\mathbf{n}, \mathbf{k}}(\mathbf{x}). \end{aligned}$$

The result therefore follows from the fact that the property to be proved is linear. \square

2.2 Bernstein Properties

A key result that makes Bernstein polynomials useful for proving polynomial inequalities is the coefficients of a Bernstein polynomial provide lower and upper bounds for the values of the polynomial over the unit box.

Proposition 3 *Let $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n}, \mathbf{i}}(\mathbf{x})$ be a Bernstein polynomial, r be a real number, and \mathfrak{R} be a real order in $\{\leq, <, \geq, >\}$. If $\hat{b}_{\mathbf{i}} \mathfrak{R} r$, for all $\mathbf{i} \leq \mathbf{n}$, then $p(\mathbf{x}) \mathfrak{R} r$, for all $\mathbf{x} \in [0, 1]$.*

Proof It can be easily proved by induction on m that $\sum_{\mathbf{i} \leq \mathbf{n}} B_{\mathbf{n}, \mathbf{i}}(\mathbf{x}) = 1$ for all \mathbf{x} such that $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$. In that argument, the base case follows from the binomial theorem:

$$\sum_{i_0 \leq n_0} B_{n_0, i_0}(x) = \sum_{i_0 \leq n_0} \binom{n_0}{i_0} x_0^{i_0} (1 - x_0)^{n_0 - i_0} = (x + (1 - x))^{n_0} = 1.$$

The inductive step follows from the binomial theorem as well. If $\hat{b}_{\mathbf{i}} \Re r$ for all $\mathbf{i} \leq \mathbf{n}$, then since $B_{\mathbf{n},\mathbf{i}}(\mathbf{x}) \geq 0$ for all \mathbf{x} such that $\mathbf{0} \leq \mathbf{x} \leq \mathbf{1}$,

$$\left(\sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x}) \right) \Re \left(\sum_{\mathbf{i} \leq \mathbf{n}} r B_{\mathbf{n},\mathbf{i}}(\mathbf{x}) \right) = r.$$

□

By Proposition 3, given a Bernstein polynomial $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$, the minimum (resp., maximum) Bernstein coefficient is a lower (resp., upper) bound estimate for the minimum (resp., maximum) value attained by p on the unit box. That is,

$$\begin{aligned} \min_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} &\leq \min_{\mathbf{x} \in [\mathbf{0}, \mathbf{1}]} p(\mathbf{x}), \\ \max_{\mathbf{x} \in [\mathbf{0}, \mathbf{1}]} p(\mathbf{x}) &\leq \max_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}}. \end{aligned} \tag{4}$$

Another useful property of a polynomial in Bernstein form is that the values of the function at the endpoints of the unit box are coefficients of the polynomial. Let \mathbf{n} be an m -tuple of natural numbers. The set $\mathcal{C}_{\mathbf{n}}$ of m -tuples of natural numbers denotes the *endpoint indices* of \mathbf{n} and it is defined as follows.

$$\mathcal{C}_{\mathbf{n}} = \{\mathbf{i} \leq \mathbf{n} \mid \forall 0 \leq j < m : i_j = 0 \text{ or } i_j = n_j\}. \tag{5}$$

Proposition 4 *Let \mathbf{x} be an element of \mathbb{R}^m such that $x_j = 0$ or $x_j = 1$ for all $0 \leq j < m$. Given an m -tuple \mathbf{n} of natural numbers define $\mathbf{i} \in \mathcal{C}_{\mathbf{n}}$ by $i_j = 0$ if $x_j = 0$ and $i_j = n_j$ if $x_j = 1$. If the Bernstein polynomial p is defined by Formula (2), then $p(\mathbf{x}) = \hat{b}_{\mathbf{i}}$.*

Proof It can be seen that for all $\mathbf{k} \leq \mathbf{n}$, with $\mathbf{k} \neq \mathbf{i}$, $B_{\mathbf{n},\mathbf{k}}(\mathbf{x}) = 0$. Thus, $p(\mathbf{x}) = \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$. Since $\binom{n_j}{i_j} = 1$ for all $0 \leq j < m$, it also follows that $B_{\mathbf{n},\mathbf{i}}(\mathbf{x}) = 1$. □

By Proposition 4, given a Bernstein polynomial $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$, the minimum (resp., maximum) Bernstein coefficient at an endpoint index is an upper (resp., lower) bound estimate for the minimum (resp., maximum) value attained by p on the unit box. That is,

$$\begin{aligned} \min_{\mathbf{x} \in [\mathbf{0}, \mathbf{1}]} p(\mathbf{x}) &\leq \min_{\mathbf{i} \in \mathcal{C}_{\mathbf{n}}} \hat{b}_{\mathbf{i}}, \\ \max_{\mathbf{i} \in \mathcal{C}_{\mathbf{n}}} \hat{b}_{\mathbf{i}} &\leq \max_{\mathbf{x} \in [\mathbf{0}, \mathbf{1}]} p(\mathbf{x}). \end{aligned} \tag{6}$$

2.3 Domain Subdivision

The reciprocal implication of Proposition 3 does not hold in general, i.e., the fact that a polynomial inequality holds on the unit box does not imply that the Bernstein coefficients of the polynomial satisfy the same inequality. In particular, the lower and upper bounds of the minimum and maximum values of a polynomial on the unit box given by formulas (4) and (6) are not always exact.

There is, however, a method that can be used to significantly improve the accuracy of the estimates for the minimum and maximum values of a multivariate polynomial p in a bounded box $[\mathbf{a}, \mathbf{b}]$. The basic idea is to subdivide $[\mathbf{a}, \mathbf{b}]$ into two boxes by picking a variable x_j , where $j < m$, and considering the case where $a_j \leq x_j \leq \frac{a_j+b_j}{2}$ separately from the case where $\frac{a_j+b_j}{2} \leq x_j \leq b_j$. This method can be used recursively to compute arbitrarily precise bounds of the minimum and maximum values of the polynomial on $[\mathbf{a}, \mathbf{b}]$.

An important feature of this subdivision method is that the Bernstein coefficients arising from the polynomial on the two subdivided intervals can be computed directly from the Bernstein coefficients of the original polynomial.

The notation \mathbf{a} with $[j \leftarrow k]$, where $j < m$ and $k \in \mathbb{R}$, denotes the m -tuple that is equal to \mathbf{a} in every index, except in j where it has the value k . Since the functions $D^L(y) = \frac{y}{2}$ and $D^R(y) = \frac{y+1}{2}$ are bijections from $[0, 1]$ into $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$, respectively, the Bernstein coefficients of a polynomial p on the boxes $[\mathbf{0}, \mathbf{1}$ with $[j \leftarrow \frac{1}{2}]]$ and $[\mathbf{0}$ with $[j \leftarrow \frac{1}{2}], \mathbf{1}]$ are the Bernstein coefficients of the polynomials

$$\begin{aligned} p^L(\mathbf{x}) &= p(\mathbf{x} \text{ with } [j \leftarrow D^L(x_j)]), \\ p^R(\mathbf{x}) &= p(\mathbf{x} \text{ with } [j \leftarrow D^R(x_j)]), \end{aligned} \quad (7)$$

respectively.

There is an algorithm, called the *sweep procedure* [8], that is commonly used to compute the Bernstein forms of p^L and p^R . In this paper, a simpler algorithm is presented where the Bernstein coefficients are computed by expanding the definitions in Formula (7). Both this algorithm and the sweep procedure have been implemented in PVS and proved correct for both the univariate and multivariate cases.

Proposition 5 *Let $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n}, \mathbf{i}}(\mathbf{x})$ be a Bernstein polynomial. For all $j < m$, $p^L(\mathbf{x}) = \sum_{\mathbf{k} \leq \mathbf{n}} \hat{b}_{\mathbf{k}}^L B_{\mathbf{n}, \mathbf{k}}(\mathbf{x})$, where*

$$\hat{b}_{\mathbf{k}}^L = \sum_{r=0}^{k_j} \frac{1}{2^{k_j}} \binom{k_j}{r} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow r]},$$

and $p^R(\mathbf{x}) = \sum_{\mathbf{k} \leq \mathbf{n}} \hat{b}_{\mathbf{k}}^R B_{\mathbf{n}, \mathbf{k}}(\mathbf{x})$, where

$$\hat{b}_{\mathbf{k}}^R = \sum_{r=0}^{n_j - k_j} \frac{1}{2^{n_j - k_j}} \binom{n_j - k_j}{r} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow n_j - r]}.$$

Proof In the left case, it is noted that for all polynomials $q(\mathbf{x}) = B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$, $q(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j}{2}])$ is given by

$$\binom{n_j}{i_j} \left(\frac{x_j}{2}\right)^{i_j} \left(1 - \left(\frac{x_j}{2}\right)\right)^{n_j - i_j} \prod_{s < m, s \neq j} \binom{n_s}{i_s} x_s^{i_s} (1 - x_s)^{n_s - i_s}.$$

It can be proved using the binomial theorem and the trinomial revision formula that

$$\binom{n_j}{i_j} \left(\frac{x_j}{2}\right)^{i_j} \left(1 - \left(\frac{x_j}{2}\right)\right)^{n_j - i_j} = \sum_{k_j = i_j}^{n_j} \frac{1}{2^{k_j}} \binom{k_j}{i_j} \binom{n_j}{k_j} x_j^{k_j} (1 - x_j)^{n_j - k_j}.$$

From this, it follows immediately that

$$q(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j}{2}]) = \sum_{\mathbf{i} \leq \mathbf{k} \leq \mathbf{i} \text{ with } [j \leftarrow n_j]} \frac{1}{2^{k_j}} \binom{k_j}{i_j} B_{\mathbf{n},\mathbf{k}}(\mathbf{x}). \quad (8)$$

Thus, if $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$, then

$$\begin{aligned} p(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j}{2}]) &= \sum_{\mathbf{i} \leq \mathbf{n}} \left(\hat{b}_{\mathbf{i}} \sum_{\mathbf{i} \leq \mathbf{k} \leq (\mathbf{i} \text{ with } [j \leftarrow n_j])} \frac{1}{2^{k_j}} \binom{k_j}{i_j} B_{\mathbf{n},\mathbf{k}}(\mathbf{x}) \right) \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \left(\sum_{r=0}^{k_j} \frac{1}{2^{k_j}} \binom{k_j}{r} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow r]} \right) B_{\mathbf{n},\mathbf{k}}(\mathbf{x}). \end{aligned}$$

The right case can be reduced to the left case as follows.

$$\begin{aligned} p(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j + 1}{2}]) &= p(\mathbf{x} \text{ with } [j \leftarrow 1 - \frac{1 - x_j}{2}]) \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow n_j - k_j]} B_{\mathbf{n},\mathbf{k}}(\mathbf{x} \text{ with } [j \leftarrow \frac{1 - x_j}{2}]), \text{ from definition of } B_{\mathbf{n},\mathbf{k}}. \end{aligned}$$

The proof continues by applying Formula (8) to the case where the variable x_j is replaced by $1 - x_j$,

$$\begin{aligned} p(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j + 1}{2}]) &= \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \left(\sum_{r=0}^{k_j} \frac{1}{2^{k_j}} \binom{k_j}{r} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow n_j - r]} \right) B_{\mathbf{n},\mathbf{k}}(\mathbf{x} \text{ with } [j \leftarrow 1 - x_j]) \\ &= \sum_{\mathbf{k} \leq \mathbf{n}} \left(\sum_{r=0}^{n_j - k_j} \frac{1}{2^{n_j - k_j}} \binom{n_j - k_j}{r} \hat{b}_{\mathbf{k} \text{ with } [j \leftarrow n_j - r]} \right) B_{\mathbf{n},\mathbf{k}}(\mathbf{x} \text{ with } [j \leftarrow x_j]) \end{aligned}$$

□

Proposition 5 can be used to improve the accuracy of the estimates for the minimum and maximum values of a Bernstein polynomial p on the unit box. This result is captured in the following proposition, the proof of which follows directly from propositions 3 and 5.

Proposition 6 *Let $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} B_{\mathbf{n},\mathbf{i}}(\mathbf{x})$ be a Bernstein polynomial, r be a real number, and \mathfrak{R} be a real order in $\{\leq, <, \geq, >\}$. If $\hat{b}_{\mathbf{i}}^L \mathfrak{R} r$ and $\hat{b}_{\mathbf{i}}^R \mathfrak{R} r$, for all $\mathbf{i} \leq \mathbf{n}$, then $p(\mathbf{x}) \mathfrak{R} r$, for all $\mathbf{x} \in [0, 1]$.*

2.4 Solving Simply Quantified Polynomial Inequalities

Proposition 1 enables the use of propositions 3 and 4 for polynomials, not just on the unit box, but on any bounded box. In particular, the minimum and maximum values of a polynomial p in the bounded box $[\mathbf{a}, \mathbf{b}]$ satisfy the inequalities

$$\begin{aligned} \min_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} &\leq \min_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} p(\mathbf{x}) \leq \min_{\mathbf{i} \in \mathcal{C}_{\mathbf{n}}} \hat{b}_{\mathbf{i}}, \\ \max_{\mathbf{i} \in \mathcal{C}_{\mathbf{n}}} \hat{b}_{\mathbf{i}} &\leq \max_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} p(\mathbf{x}) \leq \max_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}}, \end{aligned} \quad (9)$$

where the Bernstein coefficients $\hat{b}_{\mathbf{i}}$ are given by Proposition 2 for the polynomial p^* defined in Proposition 1. It is usually the case that these estimates are not exact. For example, while it may be true that $p(\mathbf{x}) > 0$ for all $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$, it may not be true that $\min_{\mathbf{i} \leq \mathbf{n}} \hat{b}_{\mathbf{i}} > 0$. In this case, Proposition 6 can be recursively applied to compute bounds that are precise to any required approximation. This is the key idea for developing a procedure that solves simply quantified polynomial inequalities on bounded boxes.

To check whether the universally quantified polynomial inequality

$$\forall \mathbf{x} \in [\mathbf{a}, \mathbf{b}] : p(\mathbf{x}) \mathfrak{R} r, \quad (10)$$

holds or not, and if not to find a counterexample, the polynomial p , for $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$, is first translated into the polynomial p^* , for $\mathbf{x} \in [0, 1]$, as defined in Proposition 1. Then, the following procedure is used.

1. Compute the Bernstein coefficients $\hat{b}_{\mathbf{i}}$, for $\mathbf{i} \leq \mathbf{n}$, of p^* .
2. If for all $\mathbf{i} \leq \mathbf{n}$, $\hat{b}_{\mathbf{i}} \mathfrak{R} r$, then, by Proposition 3, the polynomial inequality $p^*(\mathbf{x}) \mathfrak{R} r$ holds for all $\mathbf{x} \in [0, 1]$.
3. If there is $\mathbf{i} \in \mathcal{C}_{\mathbf{n}}$ such that $\neg(\hat{b}_{\mathbf{i}} \mathfrak{R} r)$, then, by Proposition 4, the polynomial inequality $p^*(\mathbf{x}) \mathfrak{R} r$ does not hold for $\mathbf{x} \in [0, 1]$ defined as $x_j = 0$ if $i_j = 0$ and $x_j = 1$ if $i_j = n_j$, for $0 \leq j < m$.
4. Otherwise, chose any $0 \leq j < m$ and recursively apply this procedure to prove that $p^*(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j}{2}]) \mathfrak{R} r$ and $p^*(\mathbf{x} \text{ with } [j \leftarrow \frac{x_j+1}{2}]) \mathfrak{R} r$.
 - (a) If both statements hold, then, by Proposition 6, the polynomial inequality $p^*(\mathbf{x}) \mathfrak{R} r$ holds for all $\mathbf{x} \in [0, 1]$.
 - (b) If the first statement does not hold for some \mathbf{x} (returned in Step 3), then the polynomial inequality $p^*(\mathbf{x}) \mathfrak{R} r$ does not hold for $\mathbf{x} \text{ with } [j \leftarrow \frac{x_j}{2}]$.

- (c) If the second statement does not hold for some \mathbf{x} (returned in Step 3), then the polynomial inequality $p^*(\mathbf{x}) \Re r$ does not hold for \mathbf{x} with $[j \leftarrow \frac{x_j+1}{2}]$.

If the procedure above states that the polynomial inequality $p^*(\mathbf{x}) \Re r$ holds for all $\mathbf{x} \in [\mathbf{0}, \mathbf{1}]$, then, by Proposition 1, Formula (10) holds. If the procedure above states that the polynomial inequality $p^*(\mathbf{x}) \Re r$ does not hold for some $\mathbf{x} \in [\mathbf{0}, \mathbf{1}]$, then Formula (10) does not hold for $\mathbf{y} \in [\mathbf{a}, \mathbf{b}]$ defined as $y_j = a_j + x_j \cdot (b_j - a_j)$, for $0 \leq j < m$.

To check whether the existentially quantified polynomial inequality $\exists \mathbf{x} \in [\mathbf{a}, \mathbf{b}] : p(\mathbf{x}) \Re r$ holds or not, i.e., whether $p(\mathbf{x}) \Re r$ is satisfiable or not, and if so to find a witness, the same procedure is used with the negated relation.

The procedure given in this section does not necessarily terminate. Furthermore, due to the subdivision algorithm in Step 4, its complexity is exponential in the number of variables. However, recent advances in polynomial representation and heuristics for the selection of the variable to subdivide in Step 4 make this algorithm practical even for polynomials with several variables. Section 3 presents a formalization of an efficient representation of multivariate Bernstein polynomials. Using this formalization, algorithms for finding lower and upper bounds of the minimum and maximum values of a polynomial are described in Section 4. These algorithms have been specified and proved correct in PVS. They are the building blocks of automated proof strategies in PVS for solving global optimization problems. Section 5 illustrates the use of these strategies.

3 Formalization of Multivariate Polynomials

Smith has introduced a representation of Bernstein polynomials that can increase the efficiency of global optimization algorithms for sparse polynomials [21]. A sparse polynomial is one where the ratio of the actual number of monomials and the total number of possible monomials is small. In the context of formalized mathematics, Smith's representation also has the advantage of reducing multivariate polynomial properties to properties of polynomials in one variable. While Smith's representation is stated in [21] as a representation of Bernstein polynomials, the development presented in this paper uses Smith's representation for both Bernstein polynomials and standard polynomials. With this representation, domain reduction and subdivision respect and preserve the structures of the represented polynomials.

3.1 Smith's Representation

Smith's representation of a polynomial, in either Bernstein or standard form, is a finite sum of the form

$$p(\mathbf{x}) = \sum_{k=0}^{t-1} c_k \prod_{j=0}^{m-1} p_{k,j}(x_j), \quad (11)$$

where, for $0 \leq k < t$ and $0 \leq j < m$, $c_k \in \mathbb{R}$ and $p_{k,j}$ is a single variable polynomial, in either Bernstein or standard form. A multivariate polynomial $p(\mathbf{x}) = \sum_{\mathbf{i} \leq \mathbf{n}} c_{\mathbf{i}} \mathbf{x}^{\mathbf{i}}$ can be written using this form, since a monomial $\mathbf{x}^{\mathbf{i}}$ is equal to $\prod_{k=0}^{m-1} x_j^{i_j}$ and, therefore, $p(\mathbf{x})$ can be written as $\sum_{k=0}^{t-1} c_k \prod_{j=0}^{m-1} x_j^{i_j}$, where t is the number of monomials in p .

One advantage of this representation of polynomials is that domain reduction and subdivision algorithms can be decomposed into separate cases for each variable. In the case of domain reduction, a multivariate polynomial p on a bounded box $[\mathbf{a}, \mathbf{b}]$ that is written in the form of Formula (11) can be translated into a polynomial on the unit box by simply translating each polynomial $p_{r,j}$ to a single variable polynomial on the unit interval $[0, 1]$. Verifying the resulting translation algorithm can be easily reduced to proving that the algorithm is correct for single variable polynomials.

Similarly, given a Bernstein polynomial p written in the form of Formula (11), the polynomials $p^L(\mathbf{x})$ and $p^R(\mathbf{x})$ of Formula (7) in Section 2 only affect the j -th variable, so the representations of these polynomials can be computed by subdividing the domain of the j -th polynomial in each product $p_{k,0}(x_0) \cdots p_{k,m-1}(x_{m-1})$. Thus, verifying this algorithm also reduces to proving correctness only for single variable polynomials. This makes this representation of both standard and Bernstein polynomials appealing for applications in theorem proving.

3.2 A Formal Representation of Polynomials

Smith's representation of polynomials can be written in formal mathematical language, such as the specification language of PVS, as follows. A single variable polynomial is represented by a tuple $\langle A, n \rangle$, where A is a function from the natural numbers into the real numbers, i.e., $A: \mathbb{N} \rightarrow \mathbb{R}$, and n is the degree of the polynomial. To evaluate the polynomial, it suffices to define $A(i) = 0$ whenever i is greater than n . In particular, there is an evaluation function `eval` that has A and n as parameters and it is defined by

$$\text{eval}(A, n)(x) = \sum_{i=0}^n A(i)x^i,$$

for $x \in \mathbb{R}$. There is similarly a Bernstein evaluation function `evalbern` defined by

$$\text{evalbern}(A, n)(x) = \sum_{i=0}^n A(i) \binom{n}{i} x^i (1-x)^{n-i}.$$

In Section 2, the degree of a multivariate polynomial and the multivariables are given by m -tuples, where m is the number of variables. In the formal development, degree and multivariable m -tuples are represented by functions $\mathbb{N} \rightarrow \mathbb{N}$ and $\mathbb{N} \rightarrow \mathbb{R}$, respectively, that return 0 for values greater than or equal to m . If \mathbf{a} and \mathbf{b} are functional terms representing m -tuples, the order $\mathbf{a} < \mathbf{b}$ is

defined as $\mathbf{a}(j) < \mathbf{b}(j)$, for all $j \geq 0$. Similarly, $\mathbf{a} \leq \mathbf{b}$ is defined as $\mathbf{a}(j) \leq \mathbf{b}(j)$, for all $j \geq 0$.

Using Smith's representation, a multivariate monomial is seen as a product of single variable monomials. Formally, a multivariate monomial is represented by a triple $\langle \mathbf{A}, \mathbf{n}, m \rangle$, where \mathbf{A} is function from the natural numbers into single variable polynomial functions, i.e., $\mathbf{A}: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R})$, \mathbf{n} is the degree of the multivariate polynomial, and m is the number of variables. The function `evalprod`, which evaluates a product of polynomials, is defined by

$$\text{evalprod}(\mathbf{A}, \mathbf{n}, m)(\mathbf{x}) = \prod_{j=0}^{m-1} \text{eval}(\mathbf{A}(j), \mathbf{n}(j))(\mathbf{x}(j)),$$

where $\mathbf{x}: \mathbb{N} \rightarrow \mathbb{R}$ is a multivariable. Similarly, there is an evaluation function `evalbernprod` for products of Bernstein polynomials defined by

$$\text{evalbernprod}(\mathbf{A}, \mathbf{n}, m)(\mathbf{x}) = \prod_{j=0}^{m-1} \text{evalbern}(\mathbf{A}(j), \mathbf{n}(j))(\mathbf{x}(j)).$$

Finally, a multivariate polynomial is represented by a quintuple $\langle \boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m \rangle$, where $\boldsymbol{\alpha}$ is a function from the natural numbers into multivariate monomial functions, i.e., $\boldsymbol{\alpha}: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, \mathbf{n} is the degree of the multivariate polynomial, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$ is such that $\mathbf{c}(k)$, for $0 \leq k < t$, is the coefficient of the k -th multivariate monomial, t is the number of multivariate monomials, and m is the number of variables. The function `evalmulti`, which evaluates a multivariate polynomial, is defined by

$$\text{evalmulti}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}) = \sum_{k=0}^{t-1} \mathbf{c}(k) \cdot \text{evalprod}(\boldsymbol{\alpha}(k), \mathbf{n}, m),$$

where $\mathbf{x}: \mathbb{N} \rightarrow \mathbb{R}$ is a multivariable. As above, there is an evaluation function `evalmultibern` for multivariate Bernstein polynomials defined by

$$\text{evalmultibern}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}) = \sum_{k=0}^{t-1} \mathbf{c}(k) \cdot \text{evalbernprod}(\boldsymbol{\alpha}(k), \mathbf{n}, m),$$

where $\mathbf{x}: \mathbb{N} \rightarrow \mathbb{R}$ is a multivariable.

The formalization presented here strongly favors functional terms where the domain is the full set of natural numbers even when the domain of the function is bounded, e.g., m -tuples. The authors have found that this design choice greatly simplifies the formal development. The fact that the domain of a function is bounded is taken care of through the application of sums, products, and iterators, which in PVS are defined for functions whose domains are the natural numbers.

The development presented in this paper does not constitute a deep embedding of multivariate polynomials. In particular, neither a formal type of multivariate polynomials nor algebraic operators on multivariate polynomials have

been defined. In this paper, notations such as $\langle A, n \rangle$ for single variable polynomials and similar expressions for multivariate monomials and multivariate polynomials are used for convenience but they are not explicitly represented in the formalization. As noted above, one of the advantages of Smith's representation is that it enables the implementation of algorithms for multivariate polynomials using single variable polynomials. Hence, a deep-embedding of multivariate polynomials would add notational complexity without a clear benefit.

3.3 Bernstein Basis

Proposition 2 states that any multivariate polynomial can be written as a multivariate Bernstein polynomial. This can be accomplished for polynomials formalized as in the previous section by first defining a function `tobern` that converts single variable polynomials to Bernstein polynomials, and then using this function to define a similar function `tomultibern` for multivariate polynomials. The function `tobern` takes as input a single variable polynomial represented by $\langle A, n \rangle$ and returns a polynomial function $\mathbb{N} \rightarrow \mathbb{R}$ defined as follows for $i \in \mathbb{N}$.

$$\text{tobern}(A, n)(i) = \sum_{k=0}^i A(k) \frac{\binom{i}{k}}{\binom{n}{k}}. \quad (12)$$

The following theorem presents Proposition 2 as it has been proved in PVS for the case of single variable polynomials.

Theorem 1 *For all $A: \mathbb{N} \rightarrow \mathbb{R}$, $n \in \mathbb{N}$, and $x \in \mathbb{R}$,*

$$\text{eval}(A, n)(x) = \text{eval}(\text{tobern}(A, n), n)(x).$$

The function `tomultibern` takes as inputs a multivariate polynomial function $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$ and the degree \mathbf{n} of the multivariate polynomial. It is defined as follows for $k, j \in \mathbb{N}$.

$$\text{tomultibern}(\alpha, \mathbf{n})(k)(j) = \text{tobern}(\alpha(k)(j), \mathbf{n}(j)). \quad (13)$$

The functions `tobern` and `tomultibern` illustrate the use of higher-order logic in this formalization. In particular, the application `tomultibern` $(\alpha, \mathbf{n})(k)(j)(i)$ has the type \mathbb{R} , but the partial application `tomultibern` (α, \mathbf{n}) has the type $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, which is the type of a multivariate polynomial function.

The following theorem presents Proposition 2 as it has been proved in PVS for the case of multivariate polynomials. The proof uses Theorem 1.

Theorem 2 *For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, and $\mathbf{x}: \mathbb{N} \rightarrow \mathbb{R}$,*

$$\text{evalmulti}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}) = \text{evalmultibern}(\text{tomultibern}(\alpha, \mathbf{n}), \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}).$$

3.4 Domain Reduction

As noted in Section 3.1, a multivariate polynomial on a bounded box $[\mathbf{a}, \mathbf{b}]$ that is written using Smith's representation can be translated to a polynomial on the unit box by simply translating each single-variable polynomial in its decomposition to the unit interval. In PVS, this is accomplished by first defining a translation function `translate` that takes a single variable polynomial represented by $\langle A, n \rangle$ and real numbers r and s as parameters. It is defined as follows for $j \in \mathbb{N}$.

$$\text{translate}(A, n, r, s)(j) = (s - r)^j \sum_{i=j}^n A(i) \binom{i}{j} r^{i-j}.$$

The function `translate` translates the single variable polynomial $\langle A, n \rangle$ from the interval $[r, s]$ to the interval $[0, 1]$. Indeed, the following theorem has been formally proved in PVS. It is a formal statement of Proposition 1 for the case of single variable polynomials.

Theorem 3 For all $A: \mathbb{N} \rightarrow \mathbb{R}$, $n \in \mathbb{N}$, $r, s \in \mathbb{R}$, with $r \neq s$, and $x \in \mathbb{R}$,

$$\text{eval}(A, n)(r + x(s - r)) = \text{eval}(\text{translate}(A, n, r, s), n)(x).$$

As in the case of evaluation, this domain reduction function can be used to define a domain reduction function `translatemulti` for multivariate polynomials. It has as parameters a multivariate polynomial function $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, the degree \mathbf{n} of the multivariate polynomial, and m -tuples \mathbf{a} and \mathbf{b} represented as functions of type $\mathbb{N} \rightarrow \mathbb{R}$. It returns a function of type $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, corresponding to a new multivariate polynomial function. The function `translatemulti` is defined for $k \in \mathbb{N}$, $i \in \mathbb{N}$ by

$$\text{translatemulti}(\alpha, \mathbf{n}, \mathbf{a}, \mathbf{b})(k)(i) = \text{translate}(\alpha(k)(i), \mathbf{n}(i), \mathbf{a}(i), \mathbf{b}(i)).$$

The following theorem is proved in PVS using Theorem 3. It presents Proposition 1 for the case of multivariate polynomials.

Theorem 4 For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $\mathbf{a}, \mathbf{b}: \mathbb{N} \rightarrow \mathbb{R}$, with $\mathbf{a} < \mathbf{b}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, and $\mathbf{x} \in \mathbb{N} \rightarrow \mathbb{R}$,

$$\begin{aligned} \text{evalmulti}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}^*) = \\ \text{evalmulti}(\text{translatemulti}(\alpha, \mathbf{n}, \mathbf{a}, \mathbf{b}), \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}), \end{aligned}$$

where $\mathbf{x}^*(j) = \mathbf{a}(j) + \mathbf{x}(j) \cdot (\mathbf{b}(j) - \mathbf{a}(j))$, for $0 \leq j < m$.

3.5 Domain Subdivision

Since the polynomials p^L and p^R from Formula 7 only affect the j -th variable of the polynomial p , Smith's representations of these polynomials can be computed by only subdividing the j -th polynomial in each product. Hence, an algorithm for domain subdivision for single variable polynomials can be used to define an algorithm for multivariate polynomials. Domain subdivision for single variable polynomials is accomplished by the functions `subdivl` and `subdivr`. These functions take a single variable polynomial $\langle A, n \rangle$ as input and return new polynomial functions of type $\mathbb{N} \rightarrow \mathbb{R}$ representing the Bernstein polynomial form in the subdivided domains. These functions are defined as follows for $i \in \mathbb{N}$.

$$\begin{aligned} \text{subdivl}(A, n)(i) &= \frac{1}{2^i} \sum_{j=0}^i \binom{i}{j} A(j), \\ \text{subdivr}(A, n)(i) &= \frac{1}{2^{n-i}} \sum_{j=0}^{n-i} \binom{n-i}{j} A(n-j). \end{aligned} \tag{14}$$

The following theorem presents Proposition 5 for the case of single variable polynomials.

Theorem 5 *For all $A: \mathbb{N} \rightarrow \mathbb{R}$, $n \in \mathbb{N}$, and $x \in \mathbb{R}$,*

$$\text{evalbern}(\text{subdivl}(A, n), n)(x) = \text{evalbern}(A, n)\left(\frac{x}{2}\right),$$

$$\text{evalbern}(\text{subdivr}(A, n), n)(x) = \text{evalbern}(A, n)\left(\frac{x+1}{2}\right).$$

It follows from these equations that if the polynomial $\langle A, n \rangle$ corresponds to the single variable Bernstein polynomial p in the sense that $p = \text{evalbern}(A, n)$, then the polynomials $\langle \text{subdivl}(A, n), n \rangle$ and $\langle \text{subdivr}(A, n), n \rangle$ correspond to $p(\frac{x}{2})$ and $p(\frac{x+1}{2})$, respectively.

There are functions `subdivlmulti` and `subdivrmulti` that correspond to subdividing the j -th variable of a multivariate polynomial. These functions take as inputs a multivariate Bernstein polynomial function $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, a degree $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{R}$, and the variable number j for subdivision. These functions return functions of type $\mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$. They are defined as follows.

$$\text{subdivlmulti}(\alpha, \mathbf{n}, j)(k)(i) = \begin{cases} \alpha(k)(i) & \text{if } i \neq j, \\ \text{subdivl}(\mathbf{A}(k)(j), \mathbf{n}(j)) & \text{otherwise.} \end{cases}$$

$$\text{subdivrmulti}(\alpha, \mathbf{n}, j)(k)(i) = \begin{cases} \alpha(k)(i) & \text{if } i \neq j \\ \text{subdivr}(\mathbf{A}(k)(j), \mathbf{n}(j)) & \text{otherwise.} \end{cases}$$

The following theorem presents Proposition 5 for the case of single variable polynomials.

Theorem 6 For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, and $\mathbf{x}: \mathbb{N} \rightarrow \mathbb{R}$,

$$\begin{aligned} \text{evalmultibern}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x} \text{ with } [j \leftarrow \frac{\mathbf{x}(j)}{2}]) &= \\ \text{evalmultibern}(\text{subdivlmult}(\alpha, \mathbf{n}, j), \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}). & \\ \text{evalmultibern}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x} \text{ with } [j \leftarrow \frac{\mathbf{x}(j) + 1}{2}]) &= \\ \text{evalmultibern}(\text{subdivrmult}(\alpha, \mathbf{n}, j), \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}). & \end{aligned}$$

Thus, the functions `subdivlmulti` and `subdivrmulti` compute the Bernstein expansions of p^L and p^R , respectively.

3.6 Bernstein Coefficients

In Section 2, the index of a multivariate monomial $x_0^{i_0} \cdots x_{m-1}^{i_{m-1}}$ is the m -tuple (i_0, \dots, i_{m-1}) . In the formal development, the index (i_0, \dots, i_{m-1}) is represented by $\mathbf{q}: \mathbb{N} \rightarrow \mathbb{N}$ where $\mathbf{q}(j) = i_j$ for $j < m$ and $\mathbf{q}(j) = 0$ for $j \geq m$. Given a multivariate polynomial degree \mathbf{n} , an index \mathbf{q} that satisfies $\mathbf{q}(j) = 0$ or $\mathbf{q}(j) = \mathbf{n}(j)$, for all j , is called an *endpoint index* of \mathbf{n} .

If p is a multivariate polynomial such that $p(\mathbf{x}) = \text{evalmulti}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x})$, then the coefficient of the monomial at index \mathbf{q} in p can be computed by the following function.

$$\text{multicoeff}(\alpha, \mathbf{c}, t, m)(\mathbf{q}) = \sum_{i=0}^{t-1} \mathbf{c}(i) \prod_{j=0}^{m-1} \alpha(i)(j)(\mathbf{q}(j)).$$

This function also computes the coefficients of a Bernstein polynomial, i.e., $\text{multicoeff}(\alpha, \mathbf{c}, t, m)(\mathbf{q})$ is the coefficient of the Bernstein monomial $B_{\mathbf{n}, \mathbf{q}}(\mathbf{x})$ (Section 2) in the Bernstein polynomial $p(\mathbf{x}) = \text{evalmultibern}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x})$.

As noted in Section 2.2, the coefficients of a Bernstein polynomial can be used to find lower and upper bounds to the minimum and maximum values of a Bernstein polynomial on the unit box. The following result is the formal statement in PVS of Proposition 3.

Theorem 7 For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, real order $\Re \in \{\leq, <, \geq, >\}$, $r \in \mathbb{R}$, and $\mathbf{x} \in [0, 1]$, if for all indices $\mathbf{q} \leq \mathbf{n}$, $\text{multicoeff}(\alpha, \mathbf{c}, t, m)(\mathbf{q}) \Re r$, then

$$\text{evalmultibern}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}) \Re r.$$

The next PVS theorem, which is the formal version of Proposition 4, states that the function `multicoeff` can be used to compute values of multivariate Bernstein polynomials.

Theorem 8 For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, and endpoint index \mathbf{q} of \mathbf{n} ,

$$\text{multicoeff}(\alpha, \mathbf{c}, t, m)(\mathbf{q}) = \text{evalmultibern}(\alpha, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x}), \quad \text{where}$$

$$\mathbf{x}(j) = \begin{cases} 0 & \text{if } \mathbf{q}(j) = 0, \\ 1 & \text{if } \mathbf{q}(j) = \mathbf{n}(j). \end{cases}$$

4 Verified Algorithms for Polynomial Global Optimization

Given a formalization of multivariate polynomials in a proof assistant, such as the formalization presented in Section 3, it is straightforward to implement the procedure presented in Section 2.4 as a tactic/strategy using the theorem prover's proof-script language. This approach was initially taken by the authors using the strategy language provided by PVS [2]. The major advantage of this approach is that since proof-scriptPa languages preserve the logical consistency of theorem provers, strategies do not need to be proved correct. Proofs built by strategies are correct by construction. However, since proofs built using the procedure described in Section 2.4 mimic the recursive structure of the branch-and-prune method, this approach is very inefficient for practical use.

An alternative approach, based on computational reflection [11], was then followed. The core components of the procedure presented in Section 2.4 are implemented as functions in the PVS specification language. The function `BernMinmax`, described in Section 4.1, computes lower and upper bounds of the minimum and maximum values of a Bernstein polynomial within the unit box. The function `PolyMinmax`, described in Section 4.2, computes range information for standard multivariate polynomials in arbitrary closed intervals. The function `Bernstein`, described in Section 4.4, solves simply quantified polynomial inequalities. These functions and their correctness properties have been verified in PVS and are used in automated strategies for solving global optimization problems. The use of these strategies is illustrated in Section 5. The basic function `BernMinmax` is still recursive. However, since it is applied to concrete ground terms, it can be efficiently executed by the theorem prover using a ground evaluator.

4.1 Function `BernMinmax`

The function `BernMinmax` is presented in Figure 1. It has as basic parameters a multivariate polynomial $\langle \alpha, \mathbf{n}, \mathbf{c}, t, m \rangle$ in Bernstein form, a maximum recursion depth $\mathbf{d} \in \mathbb{N}$, and the current recursion depth $\mathbf{k} < \mathbf{d}$. Additional inputs include a function `varsel` that determines the variable on which to subdivide at each iteration and which direction to explore first, predicates `localex` and `globalex` on the output type that cause the algorithm to exit locally and globally, respectively, and an accumulative parameter `omm` of the same type as the output value. These inputs are described in Section 4.3.

```

BernMinmax( $\alpha, \mathbf{n}, \mathbf{c}, t, m, d, k, \text{localex}, \text{globalex}, \text{varsel}, \text{omm}$ ) : Outminmax =
  let
    bmm = berncoeffsminmax( $\alpha, \mathbf{n}, \mathbf{c}, t, m$ )
  in
    if ( $k = d$ )  $\vee$  localex(bmm)  $\vee$  ( $k > 0 \wedge \text{between?}(\text{omm}, \text{bmm})$ )  $\vee$  globalex(bmm) then
      bmm
    else
      let
        (left?, j) = varsel( $\alpha, \mathbf{n}, \mathbf{c}, t, m, k$ ),
        sl = subdivlmulti( $\alpha, \mathbf{n}, j$ ),
        sr = subdivrmulti( $\alpha, \mathbf{n}, j$ ),
        ( $\alpha_1, \alpha_2$ ) = if left? then (sl, sr) else (sr, sl) endif,
         $\sigma$  = if left? then  $\lambda x.x/2$  else  $\lambda x.(x+1)/2$  endif,
        omm = if  $k > 0$  then combine(omm, bmm) else bmm endif,
        k = k + 1,
        bmm1 = BernMinmax( $\alpha_1, \mathbf{n}, \mathbf{c}, t, m, d, k, \text{localex}, \text{globalex}, \text{varsel}, \text{omm}$ )
      in
        if globalex(bmm1) then
          combine(UpdateOutminmax(bmm1,  $\sigma, j$ ), bmm)
        else
          let
            omm = combine(omm, bmm1),
            bmm2 = BernMinmax( $\alpha_2, \mathbf{n}, \mathbf{c}, t, m, d, k, \text{localex}, \text{globalex}, \text{varsel}, \text{omm}$ ),
            bmmleft = if left? then bmm1 else bmm2 endif,
            bmmright = if left? then bmm2 else bmm1 endif
          in
            combine(UpdateOutminmax(bmmleft,  $\lambda x.x/2, j$ ),
              UpdateOutminmax(bmmright,  $\lambda x.(x+1)/2, j$ ))
          endif
        endif
      endif
    endif
  endif

```

Fig. 1 The function BernMinmax

The function `BernMinmax` returns a record of type `Outminmax`, which stores information about the range of a Bernstein polynomial over the unit box $[0, 1]$. Elements of this type have six fields:

- `lbmin`: a minimum estimate for the lower bound of the polynomial.
- `lbmax`: a maximum estimate for the lower bound of the polynomial.
- `lbvar`: a point in the box $[0, 1]$ where the polynomial attains the value `lbmax`.
- `ubmin`: a minimum estimate for the upper bound of the polynomial.
- `ubmax`: a maximum estimate for the upper bound of the polynomial.

- **ubvar**: a point in the box $[0, 1]$ where the polynomial attains the value **ubmin**.

The fields **lbmin**, **lbmax**, **ubmin**, and **ubmax** are all real numbers, and the fields **lbvar** and **ubvar** are m -tuples of real numbers.

Let p be a polynomial such that $p(\mathbf{x}) = \text{evalmultibern}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x})$. The function $\text{berncoeffsminmax}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)$ iterates over all possible monomial indices below the degree \mathbf{n} and computes an element of **Outminmax** whose fields satisfy the following properties, with $\hat{b}_i = \text{multicoeff}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(i)$ for $i < \mathbf{n}$.

1. **lbmin** = $\min_{i < \mathbf{n}} \hat{b}_i$.
2. **lbmax** = $\min_{i \in \mathcal{C}_n} \hat{b}_i$.
3. **lbvar** is a m -tuple representing a value \mathbf{c} where $p(\mathbf{c}) = \text{lbmax}$.
4. **ubmin** = $\max_{i \in \mathcal{C}_n} \hat{b}_i$.
5. **ubmax** = $\max_{i < \mathbf{n}} \hat{b}_i$. is the maximum Bernstein coefficient.
6. **ubvar** is is a m -tuple representing a value \mathbf{c} where $p(\mathbf{c}) = \text{ubmin}$.

Assuming that $\text{bmm} = \text{berncoeffsminmax}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)$, Formula (9) states that $\min_{\mathbf{x} \in [0, 1]} p(\mathbf{x})$ is in the closed interval $[\text{bmm.lbmin}, \text{bmm.lbmax}]$ and that $\max_{\mathbf{x} \in [0, 1]} p(\mathbf{x})$ is in the closed interval $[\text{bmm.ubmin}, \text{bmm.ubmax}]$.

If a better precision is desired, the function **varsel** is used to select a variable to subdivide and a direction (left or right) for the recursive calls. Then, the domain subdivision functions **subdivlmulti** and **subdivrmulti**, presented in Section 3.5, are used to subdivide the unit box $[0, 1]$ into smaller subboxes. At each subdivision, the function **multicoeff** is used to compute an element of **Outminmax** that stores information about the range of the polynomial on the given subbox.

After subdividing a variable j in $[0, 1]$ using the functions **subdivlmulti** and **subdivrmulti** and applying **multicoeff** to each subdivision, two elements bmm_1 and bmm_2 of **Outminmax** are produced; one representing range information for $[0, 1]$ with $[j \leftarrow \frac{1}{2}]$ and the other representing range information for $[0]$ with $[j \leftarrow \frac{1}{2}, 1]$. Since the points represented by **lbvar** and **ubvar** are computed in a unit box, they must be translated back to the half intervals from the full interval. There is a function **update** that takes as parameters a m -tuple ℓ of real numbers, a function $\sigma: \mathbb{R} \rightarrow \mathbb{R}$, and an index $j < m$, and it returns a new m -tuple that is equal to ℓ in every entry except the j -th entry, where it is updated by the function σ . Formally,

$$\text{update}(\ell, \sigma, j) = \ell \text{ with } [j \leftarrow \sigma(\ell(j))].$$

The function **update** is used to define the function **UpdateOutminmax** on elements of **Outminmax** that updates the j -th element of its fields **lbvar** and **ubvar**.

$$\begin{aligned} \text{UpdateOutminmax}(\text{bmm}, \sigma, j) = & \text{bmm with } [\text{lbvar} \leftarrow \text{update}(\text{bmm.lbvar}, \sigma, j)] \\ & \text{with } [\text{ubvar} \leftarrow \text{update}(\text{bmm.ubvar}, \sigma, j)] \end{aligned}$$

The two elements of type `Outminmax` resulting from applying `UpdateOutminmax` to `omm1` and `omm2` are combined into a new element of type `Outminmax` that represents range information for the Bernstein polynomial over the unit box. The function `combine(omm1, omm2)` returns an element `omm` that satisfies

1. `omm.lbmin = min(omm1.lbmin, omm2.lbmin)`
2. `omm.lbmax = min(omm1.lbmax, omm2.lbmax)`
3. `omm.lbvar` is either `omm1.lbvar` or `omm2.lbvar`, depending on the value of `omm.lbmax`
4. `omm.ubmin = max(omm1.ubmin, omm2.ubmin)`
5. `omm.ubmax = max(omm1.ubmax, omm2.ubmax)`
6. `omm.ubvar` is either `omm1.ubvar` or `omm2.ubvar`, depending on the value of `omm.ubmin`

The correctness property of the function `BernMinmax` states that it computes an element of type `Outminmax` that bounds the range of a given Bernstein polynomial on the unit box. The following theorem has been proved in PVS by induction on the structure of the definition of `BernMinmax`. In PVS, the corresponding induction scheme is generated by the type-checker by restricting the output type of the function to elements that satisfy the correctness property. Theorems 7 and 8 are used to prove the base case. The inductive case is discharged by Theorem 6.

Theorem 9 *For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $n: \mathbb{N} \rightarrow \mathbb{N}$, $c: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, $d \in \mathbb{N}$, $k \in \mathbb{N}$, with $k \leq d$, $localex, globalex: Outminmax \rightarrow boolean$, $varsel: \mathbb{N} \rightarrow [boolean, below(m)]$, and $omm: Outminmax$, if $p = evalmultibern(\alpha, n, c, t, m)$ and $bmm \in Outminmax$ is given by*

$$bmm = BernMinmax(\alpha, n, c, t, m, d, k, localex, globalex, varsel, omm),$$

then

1. $p(bmm.lbvar) = bmm.lbmax$,
2. $p(bmm.ubvar) = bmm.ubmin$, and
3. $bmm.lbmin \leq p(x) \leq bmm.ubmax$,

for all $x \in [0, 1]$.

It is noted that Theorem 9 holds for all possible values of the input parameters `varsel`, `localex`, `globalex`, and `omm`. These parameters are added for practicality and efficiency reasons. They are explained in Section 4.3.

4.2 Function PolyMinmax

The function `PolyMinmax` computes range information, not on the unit box as for the algorithm `BernMinmax`, but on an arbitrary box $[a, b]$. The algorithm works in four steps:

1. Convert the polynomial from the box $[a, b]$ to the unit box $[0, 1]$ using the function `translatemulti` from Section 3.4.

```

PolyMinmax( $\alpha, \mathbf{n}, \mathbf{c}, t, m, \mathbf{a}, \mathbf{b}, d, \text{localex}, \text{globalex}, \text{varsel}$ ) =
  let
     $\alpha' = \text{translatemulti}(\alpha, \mathbf{n}, \mathbf{a}, \mathbf{b}),$ 
     $\alpha'' = \text{tomultibern}(\alpha', \mathbf{n}),$ 
    bsmnmax = BernMinmax( $\alpha'', \mathbf{n}, \mathbf{c}, t, m, d, 0,$ 
                        localex, globalex, varsel, Emptymm)
  in
    bsmnmax with [lbvar  $\leftarrow$  denormalize( $\mathbf{a}, \mathbf{b}$ )(bsmnmax.lbvar)]
              with [ubvar  $\leftarrow$  denormalize( $\mathbf{a}, \mathbf{b}$ )(bsmnmax.ubvar)]

```

Fig. 2 The function PolyMinmax

2. Convert the translated polynomial to a Bernstein polynomial using the function `tomultibern` from Section 3.3.
3. Apply `BernMinmax` to compute an element `bmm` of `Outminmax` that gives range information for the Bernstein polynomial on the unit box.
4. Translate the fields `lbvar` and `ubvar` of `bmm` from `[0, 1]` back to `[a, b]` linearly. This can be accomplished by defining a function `denormalize(a, b)` that maps `[0, 1]` to `[a, b]` componentwise. It is given on the j -th component by $x \mapsto a_j + x \cdot b_j$.

The function `PolyMinmax` is defined in Figure 2. The constant element `Emptymm` of type `Outminmax` is defined such that all the numerical fields are 0 and the m -tuples are $(0, \dots, 0)$.

The following correctness property of the function `PolyMinmax` has been proved in PVS.

Theorem 10 *For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, $d \in \mathbb{N}$, $\text{localex}, \text{globalex}: \text{Outminmax} \rightarrow \text{boolean}$, and $\text{varsel}: \mathbb{N} \rightarrow [\text{boolean}, \text{below}(m)]$, if $p = \text{evalmulti}(\alpha, \mathbf{n}, \mathbf{c}, t, m)$ and $\text{bmm} \in \text{Outminmax}$ is given by*

$$\text{bmm} = \text{PolyMinmax}(\alpha, \mathbf{n}, \mathbf{c}, t, m, \mathbf{a}, \mathbf{b}, d, \text{localex}, \text{globalex}, \text{varsel}),$$

then

1. $p(\text{bmm.lbvar}) = \text{bmm.lbmax}$,
2. $p(\text{bmm.ubvar}) = \text{bmm.ubmin}$, and
3. $\text{bmm.lbmin} \leq p(\mathbf{x}) \leq \text{bmm.ubmax}$,

for all $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$.

4.3 Parameters `varsel`, `omm`, `globalex`, and `localex`

The parameter `varsel` is used to determine two things: (1) Which variable to subdivide at each recursive step, and (2) Whether to compute bounds to the left or the right first in that variable. The algorithm takes as inputs $\alpha, \mathbf{n}, \mathbf{c}, t$,

m , and k . It returns a pair $(\text{left?}, \text{var})$, where left? is a Boolean value and $\text{var} < m$. The value left? being true means that the given variable should be subdivided to the left first, and var is a natural number representing the index of the variable to be subdivided. The most basic example of such a function is given by $\text{varsel}(\alpha, n, c, t, m, k) = (\text{true}, \text{mod}(m, k))$, which alternates the variables and always computes range information on the left interval first. However, as noted in [18] and [20], there are much more efficient methods for choosing these variables and directions, including several based on derivatives. The function varsel is an input to the algorithm in PVS, so it can facilitate any subdivision scheme. One method that has been implemented in PVS is called MaxVarMinDir . This method chooses the variable for which the range between the first and last Bernstein coefficients, when all other variables are held constant, is greatest.

The parameter omm is used to store the current output of the algorithm. The function between? tests whether the output bmm at the current recursive step can contribute anything to the final output of the function once it is combined. That is,

$$\text{between?}(\text{omm}, \text{bmm}) = (\text{omm.lbmax} \leq \text{bmm.lbmin} \wedge \\ \text{bmm.ubmax} \leq \text{omm.ubmin}).$$

At a given recursive step in the algorithm, if $\text{between?}(\text{omm}, \text{bmm})$ returns true , then the output bmm of the current recursive step will not contribute to the overall output of the function since $\text{between?}(\text{omm}, \text{bmm})$ implies that $\text{combine}(\text{omm}, \text{bmm}) = \text{omm}$.

The function BernMinmax is at the core of other algorithms that solve specific global optimization problems, e.g., finding bounds to the minimum and maximum values of a polynomial, proving a universally quantified polynomial inequality, or checking whether a polynomial inequality is satisfiable or not. Each of these problems has a different termination condition. The predicates localex and globalex are used to prune the recursion depending on particular objectives. The predicate localex will be used to exit the algorithm locally and continue to the next recursive step. While both of these predicates are used in the algorithm to simply break recursion locally, the predicate globalex will be chosen so that if recursion breaks because globalex returns true, then every recursion above will also break, effectively resulting in a global exit from the algorithm.

For instance, the algorithm can be set to compute bounds on the range of a polynomial within an arbitrary precision $\epsilon > 0$ of the actual bounds. This can be accomplished by defining the predicates

$$\text{eps_localexit}(\epsilon)(\text{bmm}) = (\text{bmm.lbmax} - \text{bmm.lbmin} \leq \epsilon \wedge \\ \text{bmm.ubmax} - \text{bmm.ubmin} \leq \epsilon), \\ \text{eps_globalexit}(\text{bmm}) = \text{false}.$$

In this case, the parameters localex and globalex are instantiated with $\text{eps_localexit}(\epsilon)$ and eps_globalexit , respectively.

```

Bernstein( $\alpha, n, c, t, m, \mathfrak{R}, \mathbf{a}, \mathbf{b}, d, \text{varsel}$ ) : Outcome =
  let
    bmm = PolyMinmax( $\alpha, n, c, t, m, \mathbf{a}, \mathbf{b}, d, \text{exit}(\mathfrak{R}), \text{counterex}(\mathfrak{R}), \text{varsel}$ )
  in
    if exit( $\mathfrak{R}$ )(bmm) then
      IsTrue
    elseif counterex( $\mathfrak{R}$ )(bmm) then
      if 0  $\mathfrak{R}$  1 then
        Counterexample(bmm.ubvar)
      else
        Counterexample(bmm.lbvar)
      endif
    else
      Unknown
    endif

```

Fig. 3 The function Bernstein

As illustrated in Section 4.4, the function `PolyMinmax` can also be used to decide whether the polynomial p satisfies the inequality $p(\mathbf{x}) \mathfrak{R} 0$ for all \mathbf{x} in a given box $[\mathbf{a}, \mathbf{b}]$. To accomplish that, the following predicates, parametric in \mathfrak{R} , are defined.

```

exit( $\mathfrak{R}$ )(bmm) : boolean =
  if 0  $\mathfrak{R}$  1 then (bmm.ubmax  $\mathfrak{R}$  0) else (bmm.lbmin  $\mathfrak{R}$  0) endif

counterex( $\mathfrak{R}$ )(bmm) : boolean =
  if 0  $\mathfrak{R}$  1 then  $\neg$ (bmm.ubmin  $\mathfrak{R}$  0) else  $\neg$ (bmm.lbmax  $\mathfrak{R}$  0) endif

```

The parameters `localex` and `globalex` of `PolyMinmax` are instantiated with `exit(\mathfrak{R})` and `counterex(\mathfrak{R})`, respectively. Thus, once it can be proved on a subbox that the polynomial inequality is satisfied, i.e., `exit(\mathfrak{R})(bmm)` returns `true` for some `bmm`, the recursion will continue on the next branch of the recursion. On the other hand, if `counterex(\mathfrak{R})(bmm)` returns `true`, the algorithm will exit globally since there is a point where the inequality does not hold.

4.4 Function Bernstein

The function `Bernstein`, defined in Figure 3, has as inputs the data structures representing a polynomial inequality $p(\mathbf{x}) \mathfrak{R} 0$ in a bounded box $[\mathbf{a}, \mathbf{b}]$. It returns an element of type `Outcome` with the values `Unknown`, `IsTrue`, or `Counterexample(c)`, where $\mathbf{c} \in [\mathbf{a}, \mathbf{b}]$.

The correctness property of `Bernstein`, which has been proved in PVS, states that if `Bernstein` returns `IsTrue`, then the inequality $p(\mathbf{x}) \mathfrak{R} 0$ holds

for all $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$. Furthermore, if the function returns `Counterexample(c)`, then the inequality $p(\mathbf{c}) \not\Re 0$ does not hold. The function returns `Unknown` when the polynomial inequality cannot be proved nor disproved for the given maximum depth \mathbf{d} and variable selection method `varsel`.

Theorem 11 *For all $\alpha: \mathbb{N} \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{R}))$, $\mathbf{n}: \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{c}: \mathbb{N} \rightarrow \mathbb{R}$, $t \in \mathbb{N}$, $m \in \mathbb{N}$, $\mathbf{d} \in \mathbb{N}$, and $\text{varsel}: \mathbb{N} \rightarrow [\text{boolean}, \text{below}(m)]$, if $p = \text{evalmulti}(\alpha, \mathbf{n}, \mathbf{c}, t, m)$ then*

1. $\text{Bernstein}(\alpha, \mathbf{n}, \mathbf{c}, t, m, \Re, \mathbf{a}, \mathbf{b}, \mathbf{d}, \text{varsel}) = \text{IsTrue}$ implies

$$\forall \mathbf{x} \in [\mathbf{a}, \mathbf{b}] : p(\mathbf{x}) \Re 0.$$

2. $\text{Bernstein}(\alpha, \mathbf{n}, \mathbf{c}, t, m, \Re, \mathbf{a}, \mathbf{b}, \mathbf{d}, \text{varsel}) = \text{Counterexample}(\mathbf{c})$ implies

$$\mathbf{c} \in [\mathbf{a}, \mathbf{b}] \wedge \neg(p(\mathbf{c}) \Re 0).$$

4.5 Open and Unbounded Intervals

The function `BernMinmax` is a simplified version of the algorithm that is implemented in PVS. The actual PVS function provides support for open, half-open, and, for some types of problems, unbounded intervals.

The mechanism through which the algorithm handles open intervals is a modification of the algorithm `berncoeffsminmax`, which computes range information for the Bernstein coefficients of a polynomial. This modification does not let the fields `lbvar` and `ubvar` of the output, which has type `Outminmax`, to be set unless the resulting point is inside the given interval. Thus, the algorithm can be used to find counterexamples to positivity, nonnegativity, negativity, and nonpositivity statements in open and half-open intervals.

In the case of unbounded intervals, there is a relatively simple result that allows a polynomial positivity (and negativity, etc.) problem on an unbounded interval to be reduced to a slightly stronger problem on a bounded interval. As a simple example, consider the problem of determining whether $p(x) > 0$ for all $x \in (0, \infty)$, where p is a polynomial in one variable. There is a bijective function $(0, 1) \mapsto (0, \infty)$ given by $x \mapsto \frac{1-x}{x}$. Thus, $p(x) > 0$ for all $x > 0$ if and only if $p(\frac{1-x}{x}) > 0$ for all $x \in (0, 1)$. But this second statement is equivalent to $x^n p(\frac{1-x}{x}) > 0$ for all $x \in (0, 1)$, where n is the degree of p . Further, $x^n p(\frac{1-x}{x})$ can be written as a polynomial in x . Thus, determining whether $p(x) > 0$ for all $x \in (0, \infty)$ can be reduced to determining whether $q(x) > 0$ for all $x \in (0, 1)$, where q is the polynomial such that $q(x) = x^n p(\frac{1-x}{x})$.

5 Strategies

The formal development presented in this paper includes the proof strategies `minmax` and `bernstein`, which are based on the PVS functions `PolyMinmax` and `Bernstein`, respectively. This section illustrates the use of these strategies to solve polynomial global optimization problems.

5.1 Strategy `minmax`

In its simplest form, the strategy `minmax` has as parameter a multivariate polynomial, which is given either as a string representation or as a location of an expression in a proof sequent. The strategy finds the minimum and maximum of the polynomial within a default precision of $\frac{1}{100}$ using maximum depth 100 and the variable selection method `MaxVarMinDir`. Optional strategy parameters allows for the user to provide specific precision, maximum depth, and variable selection method. For instance, given the following proof sequent

```
{-1}  -1 <= x
{-2}  x <= 1
{-3}  -1 <= y
{-4}  y <= 1
      |-----
{1}   4*x^2 - (21/10)*x^4 + (1/3)*x^6 + x*y - 4*y^2 + 4*y^4
      >= -1.4
```

the strategy call `(minmax (! 1 1))`, where `(! 1 1)` points to the lefthand side of the formula `{1}`, i.e., the polynomial $4x^2 - \frac{21}{10}x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$, results in the following sequent

```
{-1} PolyMinmax( $\alpha$ , $n$ , $c$ , $t$ , $m$ , $a$ , $b$ ,100,
      eps_localexit(1/100),eps_globalexit,MaxVarMinDir) =
      (# lbmax := -258761659 / 251658240,
      lbmin := -162446231 / 157286400,
      lbvar := (: -23 / 32, 1 / 16 :)
      ubmax := 97 / 30,
      ubmin := 97 / 30,
      ubvar := (: -1, -1 :) #)
{-2} boxbetween?( $x$ , $a$ , $b$ )
{-3} 4*x^2 - (21/10)*x^4 + (1/3)*x^6 + x*y - 4*y^2 + 4*y^4 =
      evalmulti( $\alpha$ , $n$ , $c$ , $t$ , $m$ )( $x$ )
{-4}  -1 <= x
{-5}  x <= 1
{-6}  -1 <= y
{-7}  y <= 1
      |-----
{1}   4*x^2 - (21/10)*x^4 + (1/3)*x^6 + x*y - 4*y^2 + 4*y^4
      >= -1.4
```

The strategy `minmax` first extracts a multivariate polynomial representation $\langle \alpha, n, c, t, m \rangle$ from the PVS expression $4x^2 - (21/10)x^4 + (1/3)x^6 + xy - 4y^2 + 4y^4$ and proves the equality between the two representations, i.e., formula `{-3}`, where \mathbf{x} is the m -tuple (x, y) . By default, the order of the variables is the order of occurrence in the polynomial expression, but the user can specify a different order using an optional strategy parameter. Then, the strategy extracts the initial variable boxes from the information in the antecedent of

the sequent and proves that $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$, i.e., formula $\{-2\}$, where \mathbf{a} and \mathbf{b} represent the m -tuple $(-1, 1)$. Finally, the strategy evaluates the ground expression $\text{PolyMinmax}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m, \mathbf{a}, \mathbf{b}, \dots)$, i.e., formula $\{-1\}$. Theorem 10 guarantees that for all $x \in [\mathbf{a}, \mathbf{b}]$,

$$\begin{aligned} -\frac{162446231}{157286400} &\leq p(\mathbf{x}) \leq \frac{97}{30}, \\ p\left(-\frac{23}{32}, \frac{1}{16}\right) &= -\frac{258761659}{251658240}, \\ p(-1, -1) &= \frac{97}{30}, \end{aligned}$$

where $p(\mathbf{x}) = 4 * x^2 - (21/10) * x^4 + (1/3) * x^6 + x * y - 4 * y^2 + 4 * y^4$. Hence, in the given variable box, the maximum value of the polynomial is exactly $\frac{97}{30}$ and attained at $(-1, -1)$, and the minimum value of the polynomial satisfies

$$-\frac{162446231}{157286400} \leq \min_{\mathbf{x} \in [\mathbf{a}, \mathbf{b}]} p(\mathbf{x}) \leq -\frac{258761659}{251658240}.$$

It can be checked that

$$-\frac{258761659}{251658240} - \frac{162446231}{157286400} = \frac{5761553}{1258291200} \leq \frac{1}{100}.$$

Therefore, the value of the polynomial at the point $(-\frac{23}{32}, \frac{1}{16})$ is within $\frac{1}{100}$ of the minimum value of the polynomial on the interval.

5.2 Strategy `bernstein`

The strategy `bernstein` automatically discharges PVS sequents having one of the following forms, where p is a multivariate polynomial expression on variables \mathbf{x} , $[\mathbf{a}, \mathbf{b}]$ is a bounded box, $\mathfrak{R} \in \{<, \leq, >, \geq\}$, and $r \in \mathbb{R}$,

1. $\vdash \forall \mathbf{x} \in \mathbb{R}^m : \mathbf{x} \in [\mathbf{a}, \mathbf{b}] \implies p(\mathbf{x}) \mathfrak{R} r$.
2. $\mathbf{x} \in [\mathbf{a}, \mathbf{b}] \vdash p(\mathbf{x}) \mathfrak{R} r$.
3. $\vdash \exists \mathbf{x} \in \mathbb{R}^m : \mathbf{x} \in [\mathbf{a}, \mathbf{b}] \wedge p(\mathbf{x}) \mathfrak{R} r$.

The strategy `bernstein` does not require any parameters, but, as in the case of the strategy `minmax`, optional strategy parameters allow for specific variable orders, maximum depths, and variable selection methods. Similar to the strategy `minmax`, `bernstein` extracts from the sequent a polynomial representation $\langle \boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m \rangle$ and a variable box $[\mathbf{a}, \mathbf{b}]$ that satisfy $p(\mathbf{x}) - r = \text{evalmulti}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x})$ and $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$.

In the case of a universally quantified sequent, i.e., sequent forms 1 and 2, the ground expression `Bernstein`($\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m, \mathfrak{R}, \mathbf{a}, \mathbf{b}, \mathbf{d}, \text{varsel}$) is evaluated, where \mathbf{d} and `varsel` are given maximum depth and variable selection method, respectively. If the outcome `IsTrue` is computed, then Theorem 11 is used to discharge the sequent. If the outcome `Counterexample`(\mathbf{c}) is computed, then it is reported that the polynomial inequality does not hold for $\mathbf{x} = \mathbf{c}$.

In the case of an existentially quantified sequent, i.e., sequent form 3, the ground expression `Bernstein($\alpha, \mathbf{n}, \mathbf{c}, t, m, \neg \mathfrak{R}, \mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{vartsel}$)` is evaluated. If the outcome `Counterexample(\mathbf{c})` is computed, then sequent is discharged by instantiating the existential variable \mathbf{x} with \mathbf{c} . If the outcome `IsTrue` is computed, then it is reported that the polynomial inequality does not hold for any $\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$.

5.3 Examples

The rest of this section presents several examples of global optimization theorems that can be automatically discharged with the strategy `Bernstein`. These examples are taken from [20] and were originally drawn from [22], where new exit conditions and methods for range subdivision are tested on particular problems. These polynomials are typical test problems for global optimization algorithms since standard tricks, such as initially eliminating certain variables, will not typically work with these problems. Thus, these problems are designed to push global optimization problems to their limits. The polynomials and the domains of the associated variables are given below.

– **Schwefel:**

$$\text{schwefel}(x_1, x_2, x_3) = (x_1 - x_2^2)^2 + (x_2 - 1)^2 + (x_1 - x_3^2)^2 + (x_3 - 1)^2,$$

where $x_1, x_2, x_3 \in [-10, 10]$.

– **3-Variable Reaction Diffusion:**

$$\text{rd}(x_1, x_2, x_3) = -x_1 + 2x_2 - x_3 - 0.835634534 x_2(1 + x_2),$$

where $x_1, x_2, x_3 \in [-5, 5]$.

– **Caprasse's System**

$$\text{caprasse}(x_1, x_2, x_3, x_4) = -x_1 x_3^3 + 4x_2 x_3^2 x_4 + 4x_1 x_3 x_4^2 + 2x_2 x_4^3 + 4x_1 x_3 + 4x_3^2 - 10x_2 x_4 - 10x_4^2 + 2,$$

where $x_1, x_2, x_3, x_4 \in [-0.5, 0.5]$.

– **Adaptive Lotka-Volterra System:**

$$\text{lv}(x_1, x_2, x_3, x_4) = x_1 x_2^2 + x_1 x_3^2 + x_1 x_4^2 - 1.1x_1 + 1,$$

where $x_1, x_2, x_3, x_4 \in [-2, 2]$.

Problem	k_1	k_2
Schwefel	-0.00000000058806	0.00000000058806
Reaction Diffusion	-36.7126907	-36.7126
Caprasse	-3.1801	-3.18009
Lotka-Volterra	-20.801	-20.799
Butcher	-1.44	-1.439
Magnetism	-0.25001	-0.2499
Heart Dipole	-1.7435	-1.7434

Table 1 Constants k_1 and k_2 for Global Optimization Theorems

– **Butcher’s Problem:**

$$\text{butcher}(x_1, x_2, x_3, x_4, x_5, x_6) = x_6x_2^2 + x_5x_3^2 - x_1x_4^2 + x_4^3 + x_4^2 - \frac{1}{3}x_1 + \frac{4}{3}x_4,$$

where $x_1 \in [-1, 0]$, $x_2 \in [-0.1, 0.9]$, $x_3 \in [-0.1, 0.5]$, $x_4 \in [-1, -0.1]$, $x_5 \in [-0.1, -0.05]$, and $x_6 \in [-0.1, -0.03]$.

– **7-Variable Magnetism:**

$$\text{magnetism}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 - x_1,$$

where $x_1, x_2, x_3, x_4, x_5, x_6, x_7 \in [-1, 1]$.

– **Heart Dipole:**

$$\text{heart}(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = -x_1x_6^3 + 3x_1x_6x_7^2 - x_3x_7^3 + 3x_3x_7x_6^2 - x_2x_5^3 + 3x_2x_5x_8^2 - x_4x_8^3 + 3x_4x_8x_5^2 - 0.9563453,$$

where $x_1 \in [-0.1, 0.4]$, $x_2 \in [0.4, 1]$, $x_3 \in [-0.7, -0.4]$, $x_4 \in [-0.7, 0.4]$, $x_5 \in [0.1, 0.2]$, $x_6 \in [-0.1, 0.2]$, $x_7 \in [-0.3, 1.1]$, and $x_8 \in [-1.1, -0.3]$.

For each one of these problems, the following types of theorems are proved for some $k_1, k_2 \in \mathbb{R}$.

– **Theorem p.forall:** $\forall \mathbf{x} \in \mathbb{R}^m : \mathbf{x} \in [\mathbf{a}, \mathbf{b}] \implies p(\mathbf{x}) \geq k_1$.

– **Theorem p.exists:** $\exists \mathbf{x} \in \mathbb{R}^m : \mathbf{x} \in [\mathbf{a}, \mathbf{b}] \wedge p(\mathbf{x}) \leq k_2$.

The constants k_1 and k_2 are chosen such that $k_2 - k_1 < \epsilon$, where ϵ is a small positive number. Hence, these theorems imply that both k_1 and k_2 are estimates of the global minimum of the polynomial p in the box $[\mathbf{a}, \mathbf{b}]$, within a precision of ϵ . Table 1 shows the constants k_1 and k_2 for each problem.

Each of the theorems for the problems listed in Table 1 can be proved in PVS using the proof strategy (**bernstein**). Table 2 shows proof times (in seconds) for each theorem in a MacBook Pro 2.4 GHz Inter Core 2 Duo, 8 GB of memory. In the case of the universally quantified theorems, a considerable amount of time is spent in the verification of the equivalence $p(\mathbf{x}) - k_1 = \text{evalmulti}(\boldsymbol{\alpha}, \mathbf{n}, \mathbf{c}, t, m)(\mathbf{x})$. This step requires many symbolic manipulations

Problem	p_forall (sec)		p_exists (sec)
	Full	W/O Equiv.	
Schwefel	10.23	3.18	1.27
Reaction Diffusion	3.11	0.17	0.21
Caprasse	11.44	1.25	0.01
Lotka-Volterra	4.75	0.23	0.24
Butcher	19.83	0.47	0.43
Magnetism	160.44	82.87	1.71
Heart Dipole	79.68	26.14	14.94

Table 2 Proof Times for Global Optimization Theorems

in PVS and is therefore slow. Thus, the bottleneck in proof speed for these theorems in PVS is not the execution of the algorithm `PolyMinMax` but rather verifying that the polynomial representation is correct. The first column in the section `p_forall` shows the total time to prove the theorem, and the second column shows the proof time without checking the equivalence of the polynomial representations. In the case of the existential theorem, the equivalence does not need to be proved. This is because the algorithm `PolyMinMax` provides points, `lbvar` and `ubvar`, where the polynomial attains the values `lbmax` and `ubmin`, respectively. Thus, the final step in the existential proofs is an instantiation with a particular choice of variables for which the inequality holds, and so the equivalence is never proved.

6 Conclusion

This paper presented a set of formally verified algorithms for global optimization of multivariate polynomials. These algorithms, which are based on recent Bernstein polynomial techniques, are the building blocks of proof strategies for automatically finding upper and lower polynomial bounds and solving simply quantified multivariate polynomial inequalities. For multivariate polynomial global optimization, the verification technique presented in this paper is superior to techniques based on quantifier elimination.

One algorithm for variable selection for domain subdivision in the branching and bounding scheme chooses the variable for which the range between the first and last Bernstein coefficients, when all other variables are held constant, is greatest. However, as noted in [18] and [20], there are more efficient methods for choosing these variables that have not been implemented, including several based on derivatives. The function `varsel` is an input to the algorithm in PVS, so it can facilitate any new subdivision scheme. There are other heuristics that can be used to increase the efficiency of optimization algorithms based on Bernstein polynomials, which are also described in papers such as [18] and [20]. Some of these heuristics reduce to pruning strategies that can be implemented through the `local_exit` and `global_exit` parameters of the function `BernMinMax`. Since Theorem 9 holds for all possible inputs

of the function, the correctness property of `BernMinmax` is not affected by any particular instantiation of these parameters.

From an algorithmic point of view, the performance of `BernMinmax` can still be improved by using additional data structures that cache values involved in the computation of `subdivlmulti` and `subdivrmulti`. The definition of these data structures is not difficult but they require modifications to the formalization that add complexity to an already technically complex proof. These enhancements are left for future work.

As explained in Section 5.3, a bottleneck in the logical steps performed by the strategy `bernstein` is the equivalence proof between a polynomial expression in the PVS language and its formal representation. This proof requires several symbolic manipulations that in many cases are slower than the actual computation of the function `BernMinmax`. Future work will look into developing strategies inspired by symbolic execution techniques to improve the performance of these types of equivalence proofs.

Despite all the possible improvements, it is important to note that the performance of the algorithms presented in this paper cannot be compared with similar algorithms implemented in a programming language such as C++ with floating point computations. For instance, the tool `RealPaver` [10], which solves non-linear constraint satisfaction problems over the real numbers using branch-and-prune interval analysis techniques, reduces a problem like Heart Dipole from Section 5.3 in a few seconds. There are several reasons for this. The first is that PVS, being a specification language, is not optimized for computation. The second is that numerical computations in PVS are performed with infinite-precision rational arithmetic. This is much more costly than computations with floating point numbers in a programming language. Therefore, unlike a programming language, there is an absolute guarantee that the results in PVS are correct and that floating point arithmetic errors have not affected the outcome. Finally, as opposed to `RealPaver` and similar constraint solvers, the algorithms and strategies presented in this paper are backed by formal theorems. Indeed, the proof of every proposition discharged with the strategy `bernstein` can be unfolded in a proof that uses only basic deductive steps in PVS.

The mention of interval analysis in the previous paragraph is not coincidence. Interval arithmetic is another well-known technique for global optimization [16]. Indeed, the branch-and-prune techniques used in interval analysis are very similar in spirit to the subdivision method used in algorithms based on Bernstein polynomials. An interval arithmetic library is available in PVS [5] as part of the PVS NASA Libraries (<http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html>). That library includes strategies for computing precise numerical approximations of real number expressions. Those strategies do not perform well on polynomials with multiple variables, but they handle real-valued functions such as logarithm, exponential, square root, and trigonometric functions. Future work will integrate the Bernstein formal development presented in this paper into the PVS interval arithmetic library. This integration will take advantage of the efficient computation of bounds for

multivariate polynomials and the larger set of expressions supported through interval arithmetic.

References

1. Akbarpour, B., Paulson, L.C.: MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning* 44(3), 175–205 (2010)
2. Archer, M., Di Vito, B., Muñoz, C. (eds.): *Design and Application of Strategies/Tactics in Higher Order Logics*. No. NASA/CP-2003-212448, NASA, Langley Research Center, Hampton VA 23681-2199, USA (September 2003)
3. Bertot, Y., Guillhot, F., Mahboubi, A.: A formal study of Bernstein coefficients and polynomials. Tech. Rep. inria-005030117, INRIA (July 2010)
4. Crespo, L.G., Muñoz, C.A., Narkawicz, A.J., Kenny, S.P., Giesy, D.P.: Uncertainty analysis via failure domain characterization: Polynomial requirement functions. In: *Proceedings of European Safety and Reliability Conference*. Troyes, France (September 2011)
5. 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)
6. de Dinechin, F., Lauter, C., Melquiond, G.: Certifying the floating-point implementation of an elementary function using Gappa. *IEEE Transactions on Computers* 60(2), 242–253 (February 2011)
7. Garloff, J.: Convergent bounds for the range of multivariate polynomials. In: *Proceedings of the International Symposium on interval mathematics on Interval mathematics 1985*. pp. 37–56. Springer-Verlag, London, UK (1985)
8. Garloff, J.: The Bernstein algorithm. *Interval Computations* 4, 154–168 (1993)
9. Garloff, J.: Application of Bernstein expansion to the solution of control problems. *Reliable Computing* 6, 303–320 (2000)
10. Granvilliers, L., Benhamou, F.: RealPaver: An interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software* 32(1), 138–156 (March 2006)
11. Harrison, J.: *Metatheory and reflection in theorem proving: A survey and critique*. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK (1995), available on the Web as <http://www.cl.cam.ac.uk/~jrh13/papers/reflect.dvi.gz>
12. Kuchar, J., Yang, L.: A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems* 1(4), 179–189 (December 2000)
13. Lorentz, G.G.: *Bernstein Polynomials*. Chelsea Publishing Company, New York, N.Y., second edn. (1986)
14. Mahboubi, A.: Implementing the cylindrical algebraic decomposition within the Coq system. *Mathematical Structures in Computer Science* 17(1), 99–127 (February 2007)
15. McLaughlin, S., Harrison, J.: A proof-producing decision procedure for real arithmetic. In: Nieuwenhuis, R. (ed.) *Proceedings of the 20th International Conference on Automated Deduction*, proceedings. *Lecture Notes in Computer Science*, vol. 3632, pp. 295–314 (2005)
16. Moa, B.: *Interval Methods for Global Optimization*. Ph.D. thesis, University of Victoria (2007)
17. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) *Proceeding of the 11th International Conference on Automated Deduction*. *Lecture Notes in Artificial Intelligence*, vol. 607, pp. 748–752. Springer (June 1992)
18. P. S. V. Nataraj, M.A.: A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *International Journal of Automation and Computing* 4(4), 342 (2007), http://www.ijac.net:8080/Jwk\ijac/EN/abstract/article_506.shtml
19. Passmore, G.O., Jackson, P.B.: Combined decision techniques for the existential theory of the reals. In: Dixon, L. (ed.) *Proceedings of Calculemus/Mathematical Knowledge Managment*. pp. 122–137. No. 5625 in LNAI, Springer-Verlag (2009)

20. Ray, S., Nataraj, P.S.: An efficient algorithm for range computation of polynomials using the Bernstein form. *Journal of Global Optimization* 45, 403–426 (November 2009), <http://portal.acm.org/citation.cfm?id=1644158.1644172>
21. Smith, A.P.: Fast construction of constant bound functions for sparse polynomials. *J. of Global Optimization* 43, 445–458 (March 2009)
22. Verschelde, J.: The PHC pack, the database of polynomial systems. Tech. rep., University of Illinois, Mathematics Department, Chicago, IL (2001)
23. Zumkeller, R.: *Global Optimization in Type Theory*. Ph.D. thesis, École Polytechnique Paris (2008)