J Autom Reasoning (2013) 51:151–196
DOI 10.1007/s10817-012-9256-3

Formalization of Bernstein Polynomials and Applications to Global Optimization

César Muñoz · Anthony Narkawicz

Received: 18 July 2011 / Accepted: 27 June 2012 / Published online: 18 July 2012

Abstract This paper presents a formalization in higher-order logic of a practical 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.

Keywords Formal verification · Non-linear arithmetic · Global optimization · Bernstein polynomials · Interactive theorem proving

1 Introduction

Many engineering problems require determining whether, given bounds on the variables of a multivariate polynomial, the values obtained by the polynomial always fall within a particular range. These types of problems are called *polynomial global optimization* problems. Global optimization problems appear in critical applications such as air traffic conflict detection and resolution algorithms [23], floating point analysis [14], and uncertainty and reliability analysis of dynamic and control systems [10, 17]. Finding precise bounds for the minimum and maximum values of a function is fundamental to the logical correctness of these applications and, for a safety critical system, this correctness is an important component of a safety case.

C. Muñoz (✉) · A. Narkawicz
NASA Langley Research Center, Hampton, VA 23681, USA
e-mail: Cesar.A.Munoz@nasa.gov

A. Narkawicz
e-mail: Anthony.Narkawicz@nasa.gov

For example, a common problem used as a test for global optimization algorithms is the Heart Dipole problem [37]. 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 $x_8 \in [-1.1, -0.3]$:

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

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

Bernstein polynomials form a well-known technique for global optimization [15, 16] and numerical approximation [24]. They are often called Bézier curves when used in the domain of computer graphics. Bernstein polynomials are used to determine bounds on the range of a multivariate polynomial where each variable lies in a finite interval.

This paper presents a formalization of a representation of Bernstein polynomials in the higher-order logic of the Prototype Verification System (PVS) [33]. Using this representation, an algorithm for global optimization is formalized and verified in PVS. This algorithm is based on a branch and bound technique [37] and a clever data structure for representing polynomials [38]. The formally verified branch and bound algorithm is the foundation of proof strategies for mechanically and automatically finding lower and upper bounds for the minimum and maximum values of a polynomial and for solving simply quantified polynomial inequalities. As far as the authors know, the algorithm presented in this paper is the first algorithm for multivariate global optimization based on Bernstein polynomials that has been *completely* verified in a proof assistant.

The rest of the paper is organized as follows. A general overview of multivariate Bernstein polynomials and their main properties is given in Section 2. The formalization of a polynomial representation and verified algorithms for estimating bounds of the minimum and maximum value of a polynomial are described in Sections 3 and 4, respectively. Automated strategies for solving polynomial global optimization problems in PVS and examples of use are presented in Section 5. Related work is discussed in Section 6. The last section concludes this paper.

The formal development presented in this paper is electronically available from http://shemesh.larc.nasa.gov/people/cam/Bernstein. Instructions can be found in the file top.pvs in the PVS library 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

For readability, this section is presented in a rigorous, but informal, notation similar to that used in mathematics textbooks, for example [24]. In particular, the term *polynomial* refers to a mathematical expression involving a finite sum of powers in multiple variables multiplied by numerical constants.

Formal definitions of the concept presented in this section will be provided in Section 3. All the properties presented in this section have been mechanically verified for those formal definitions. 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 latter are called *theorems*. The formal proofs of these theorems closely follow the proofs of the propositions presented here.

Finite sequences of real numbers in the form $(a_0, \ldots, a_{m-1})$ will be called *tuples*, and such a finite sequence with a known number of elements m will more specifically be called an *m-tuple*. That is, m-tuples are elements of $\mathbb{R}^m$, and every tuple is an m-tuple for a unique natural number m. Similarly, a finite sequence of natural numbers is called an *index*, and such a sequence with m elements is called an *m-index*. Meta-variables of tuples and indices will be typed in **boldface**. The orders $<$ and $\leq$ compare two tuples (respectively indices) with the same number of elements. If m is a natural number and $\boldsymbol{a}$ and $\boldsymbol{b}$ are m-tuples (respectively m-indices), then $\boldsymbol{a} < \boldsymbol{b}$ if and only if $a_j < b_j$ for all natural numbers $j < m$. Moreover, $\boldsymbol{a} \leq \boldsymbol{b}$ if and only if $a_j \leq b_j$ for all natural numbers $j < m$. A *(bounded) m-box*, written $[\boldsymbol{a}, \boldsymbol{b}]$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are m-tuples and $\boldsymbol{a} < \boldsymbol{b}$, denotes the set $\{\boldsymbol{x} \in \mathbb{R}^m \mid \boldsymbol{a} \leq \boldsymbol{x} \leq \boldsymbol{b}\}$ of m-tuples. For $j < m$, the set $[a_j, b_j] = \{x \in \mathbb{R} \mid a_j \leq x \leq b_j\}$ is called the *j-th interval* of $[\boldsymbol{a}, \boldsymbol{b}]$.

The product $\boldsymbol{x}^{\boldsymbol{i}} = \prod_{j=0}^{m-1} x_j^{i_j}$, where $\boldsymbol{i}$ is an m-index and $\boldsymbol{x}$ is an m-tuple of variables over $\mathbb{R}$, is called an *m-variate monomial* of degree $\boldsymbol{i}$. An *m-variate polynomial* of degree at most $\boldsymbol{n}$ is a finite sum of the form

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}, \tag{2}$$

where $c_{\boldsymbol{i}} \in \mathbb{R}$, for $\boldsymbol{i} \leq \boldsymbol{n}$, is called the $\boldsymbol{i}$-th *coefficient* of p. The *degree* of the m-variate polynomial p is the minimum m-index $\boldsymbol{k} \leq \boldsymbol{n}$ such that every coefficient $c_{\boldsymbol{i}} \neq 0$, with $\boldsymbol{i} \leq \boldsymbol{n}$, satisfies $\boldsymbol{i} \leq \boldsymbol{k}$. Note that this does not imply that if $\boldsymbol{k}$ is the degree of p, then $c_{\boldsymbol{k}} \neq 0$. Moreover, Formula (2) does not state that $\boldsymbol{n}$ is the degree of p.

When the dimension m is either known from the context or irrelevant to the discussion, this paper will refer to monomial, polynomial, tuple, index, box, etc., as opposed to m-variate monomial, m-variate polynomial, m-tuple, m-index, m-box, etc.

An m-variate polynomial p can be seen as a function from $\mathbb{R}^m$ into $\mathbb{R}$. The *evaluation* of a polynomial p in a tuple $\boldsymbol{a}$ is the function application $p(\boldsymbol{a})$. The expression "the polynomial p on a box $[\boldsymbol{a}, \boldsymbol{b}]$" refers to the polynomial p whose domain has been restricted to the box $[\boldsymbol{a}, \boldsymbol{b}]$. In this case, the polynomial p will be seen a function from $[\boldsymbol{a}, \boldsymbol{b}]$ into $\mathbb{R}$.

Several properties in this section are given for polynomials on the *unit box* $\mathbb{U}^m = [\boldsymbol{0}^m, \boldsymbol{1}^m]$, where $\boldsymbol{0}^m$ and $\boldsymbol{1}^m$ are m-tuples whose components are all 0 and all 1, respectively. The following proposition states that for any polynomial on an arbitrary box there exists another polynomial on the unit box that attains the same values.

Proposition 1 *Let* $[\boldsymbol{a}, \boldsymbol{b}]$ *be an m-box,* $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$ *be an m-variate polynomial, and* $\sigma : \mathbb{U}^m \to [\boldsymbol{a}, \boldsymbol{b}]$ *be defined by* $\sigma(\boldsymbol{x})_j = a_j + x_j(b_j - a_j)$, *where* $0 \leq j < m$. *For all* $\boldsymbol{x} \in \mathbb{U}^m$, $p(\sigma(\boldsymbol{x})) = p^*(\boldsymbol{x})$, *where* $p^*(\boldsymbol{x}) = \sum_{\boldsymbol{k} \leq \boldsymbol{n}} r_{\boldsymbol{k}} \boldsymbol{x}^{\boldsymbol{k}}$ *and*

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

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

Proof By the binomial theorem,

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

□

2.1 Bernstein Basis Polynomials

The expression *Bernstein polynomial* refers to a polynomial written in the form [24]

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \le \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x}),$$

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

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

The coefficients $\hat{b}_{\boldsymbol{i}}$ are called the Bernstein coefficients of p. The m-variate polynomials $B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ in Formula (3) are called Bernstein *basis* polynomials as they form a basis for the vector space of m-variate polynomials of degree at most $\boldsymbol{n}$. Indeed, as the following proposition states, any polynomial can be written as a polynomial in Bernstein form.

Proposition 2 *Any m-variate polynomial* $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \le \boldsymbol{n}} c_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}}$ *can be written in Bernstein form as* $p(\boldsymbol{x}) = \sum_{\boldsymbol{k} \le \boldsymbol{n}} \hat{b}_{\boldsymbol{k}}\, B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x})$, *where*

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

Proof The *trinomial revision* formula states that for all natural numbers i, k, and n, with $i \leq k \leq n$,

$$\binom{k}{i}\binom{n}{k} = \binom{n}{i}\binom{n-i}{k-i}. \tag{4}$$

Thus, if $\boldsymbol{i}$ and $\boldsymbol{n}$ are m-indices such that $\boldsymbol{i} \leq \boldsymbol{n}$, 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 m-variate monomial $\boldsymbol{x}^{\boldsymbol{i}}$ can be written in Bernstein form as follows.

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

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

2.2 Properties of Bernstein Polynomials

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

Proposition 3 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}} B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form, r be a real number, and $\Re$ be a real order in $\{\leq, <, \geq, >\}$. If $\hat{b}_{\boldsymbol{i}}\ \Re\ r$, for all $\boldsymbol{i} \leq \boldsymbol{n}$, then $p(\boldsymbol{x})\ \Re\ r$, for all $\boldsymbol{x} \in \mathbb{U}^m$.*

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

$$\sum_{i_0=0}^{n_0} B_{n_0,i_0}(x_0) = \sum_{i_0=0}^{n_0} \binom{n_0}{i_0} x_0^{i_0} (1-x_0)^{n_0-i_0} = (x + (1-x))^{n_0} = 1.$$

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

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

Therefore, $p(\boldsymbol{x}) \Re r$. □

By Proposition 3, the values attained by a polynomial on the unit box are bounded by the minimum and maximum Bernstein coefficient of the polynomial.

Corollary 1 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form. For all $\boldsymbol{x} \in \mathbb{U}^m$,*

$$\begin{aligned} \min_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}} &\leq \min_{\boldsymbol{x} \in \mathbb{U}^m} p(\boldsymbol{x}), \\ \max_{\boldsymbol{x} \in \mathbb{U}^m} p(\boldsymbol{x}) &\leq \max_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}. \end{aligned} \tag{5}$$

Another useful property of Bernstein polynomials is that the values of a polynomial at the endpoints of the unit box are Bernstein coefficients of the polynomial. An m-tuple $\boldsymbol{c}$ is an *endpoint* of an m-box $[\boldsymbol{a}, \boldsymbol{b}]$ if and only if either $c_j = a_j$ or $c_j = b_j$, for all $j < m$. The set of endpoints of an m-box $[\boldsymbol{a}, \boldsymbol{b}]$ is denoted $\mathcal{E}_{[\boldsymbol{a},\boldsymbol{b}]}$. Given an m-index $\boldsymbol{n}$, an m-index $\boldsymbol{k}$ is an *endindex* of $\boldsymbol{n}$ if and only if either $k_j = 0$ or $k_j = n_j$, for $j < m$. The set of endindices of $\boldsymbol{n}$ is denoted $\mathcal{I}_{\boldsymbol{n}}$. The following proposition establishes a correspondance between the set of endindices of $\boldsymbol{n}$ and the set of endpoints of $\mathbb{U}^m$.

Proposition 4 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form. If $\boldsymbol{k}$ is an endindex of $\boldsymbol{n}$, i.e., $\boldsymbol{k} \in \mathcal{I}_{\boldsymbol{n}}$, then $p(\boldsymbol{y}) = \hat{b}_{\boldsymbol{k}}$, where $\boldsymbol{y}$ is the endpoint of $\mathbb{U}^m$ defined as follows.*

$$y_j = \begin{cases} 0 & \text{if } q_j = 0, \\ 1 & \text{if } q_j = n_j. \end{cases} \tag{6}$$

Proof Let $\boldsymbol{y}$ be defined as in Formula (6). It can be seen that for all $\boldsymbol{i} \leq \boldsymbol{n}$, with $\boldsymbol{i} \neq \boldsymbol{k}$, $B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{y}) = 0$. Thus, $p(\boldsymbol{y}) = \hat{b}_{\boldsymbol{k}}\, B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{y})$. Since $\binom{n_j}{q_j} = 1$ for all $j < m$, it also follows that $B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{y}) = 1$ and, therefore, $p(\boldsymbol{y}) = \hat{b}_{\boldsymbol{k}}$. □

By Proposition 4, the minimum Bernstein coefficient at an endindex is an upper bound for the minimum value attained by a polynomial on the unit box. Similarly, the maximum Bernstein coefficient at an endindex is a lower bound for the maximum value attained by a polynomial on the unit box.

Corollary 2 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form. For all $\boldsymbol{x} \in \mathbb{U}^m$,*

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{U}^m} p(\boldsymbol{x}) &\leq \min_{\boldsymbol{i} \in \mathcal{I}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}}, \\ \max_{\boldsymbol{i} \in \mathcal{I}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}} &\leq \max_{\boldsymbol{x} \in \mathbb{U}^m} p(\boldsymbol{x}). \end{aligned} \tag{7}$$

By Proposition 1, Corollaries 1 and 2, the minimum and maximum values of an m-variate polynomial $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}}\, \boldsymbol{x}^{\boldsymbol{i}}$ on an m-box $[\boldsymbol{a}, \boldsymbol{b}\,]$ satisfy the inequalities

$$\begin{aligned} \min_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}} \leq \min_{\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}\,]} p(\boldsymbol{x}) \leq \min_{\boldsymbol{i} \in \mathcal{I}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}}, \\ \max_{\boldsymbol{i} \in \mathcal{I}_{\boldsymbol{n}}} \hat{b}_{\boldsymbol{i}} \leq \max_{\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}\,]} p(\boldsymbol{x}) \leq \max_{\boldsymbol{i} \leq \boldsymbol{n}} \hat{b}_{\boldsymbol{i}}, \end{aligned} \tag{8}$$

where $\hat{b}_{\boldsymbol{i}}$ are the Bernstein coefficients of the polynomial $p^*(\boldsymbol{x}) = \sum_{\boldsymbol{k} \leq \boldsymbol{n}} r_{\boldsymbol{k}}\, \boldsymbol{x}^{\boldsymbol{k}}$ and

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

2.3 Subdivision Method

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 bound estimates given by Formula (8) are seldom exact.

Example 1 The Bernstein coefficients of the univariate polynomial $p(x) = 4x^2 - 4x + 1$, which attains its minimum at the point $\frac{1}{2}$ with $p(\frac{1}{2}) = 0$ can be written in Bernstein form as $\binom{2}{0}(1-x)^2 - \binom{2}{1}x(1-x) + \binom{2}{2}x^2$, so it has $\hat{b}_0 = 1$, $\hat{b}_1 = -1$, and $\hat{b}_2 = 1$ as Bernstein coefficients. In this case, $p(x) \geq 0$ for all $x \in [0, 1]$, but it is not true that $\min_{i \leq 2} \hat{b}_i \geq 0$.

The *subdivision* method is a branching technique that significantly improves the bound estimates of the minimum and maximum values of an m-variate polynomial p on an m-box $[\boldsymbol{a}, \boldsymbol{b}\,]$ given by Formula (8). The basic idea is to split $[\boldsymbol{a}, \boldsymbol{b}\,]$ into two boxes by selecting a variable x_j, with $j < m$, and then consider 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 technique can be used recursively to compute arbitrarily precise bounds of the minimum and maximum values of the polynomial on $[\boldsymbol{a}, \boldsymbol{b}\,]$. An important feature of the 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 $\boldsymbol{a}$ `with`$[j{:=}r]$, where $j < m$ and $r \in \mathbb{R}$, denotes the m-tuple that is equal to $\boldsymbol{a}$ in every index, except in j where it has the value r. Since the functions $D^L(x) = \frac{x}{2}$ and $D^R(x) = \frac{x+1}{2}$ are bijections from $[0, 1]$ into $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$, respectively, the Bernstein coefficients of an m-variate polynomial p on the boxes

$[\mathbf{0}^m, \mathbf{1}^m \texttt{ with}[j{:=}\frac{1}{2}]]$ and $[\mathbf{0}^m \texttt{ with}[j{:=}\frac{1}{2}], \mathbf{1}^m]$ are the Bernstein coefficients of the polynomials

$$\begin{aligned} p_j^L(\boldsymbol{x}) &= p\left(\boldsymbol{x} \texttt{ with}\left[j := \frac{x_j}{2}\right]\right), \\ p_j^R(\boldsymbol{x}) &= p\left(\boldsymbol{x} \texttt{ with}\left[j := \frac{x_j+1}{2}\right]\right), \end{aligned} \tag{9}$$

respectively.

An algorithm by de Casteljau [6], based on recursively applying linear interpolations, is often used in global optimization problems to compute the Bernstein coefficients of p_j^L and p_j^R [16]. In this paper, another algorithm is used where the Bernstein coefficients are computed, not by linear interpolation as in de Casteljau's algorithm, but by directly expanding the definitions in Formula (9). Both de Casteljau's algorithm and the method presented below have been implemented in PVS and proved correct for both the univariate and multivariate cases.[1]

Proposition 5 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form. For all $j < m$, $p_j^L(\boldsymbol{x}) = \sum_{\boldsymbol{k}\le\boldsymbol{n}} \hat{b}^L_{\boldsymbol{k}}\, B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x})$, where*

$$\hat{b}^L_{\boldsymbol{k}} = \sum_{r=0}^{k_j} \frac{1}{2^{k_j}} \binom{k_j}{r} \hat{b}_{\boldsymbol{k} \texttt{ with}[j:=r]},$$

and $p_j^R(\boldsymbol{x}) = \sum_{\boldsymbol{k}\le\boldsymbol{n}} \hat{b}^R_{\boldsymbol{k}}\, B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x})$, where

$$\hat{b}^R_{\boldsymbol{k}} = \sum_{r=0}^{n_j-k_j} \frac{1}{2^{n_j-k_j}} \binom{n_j-k_j}{r} \hat{b}_{\boldsymbol{k} \texttt{ with}[j:=n_j-r]}.$$

Proof In the left case, it is noted that for all polynomials $q(\boldsymbol{x}) = B_{\boldsymbol{n},\boldsymbol{i}}(x)$, $q(\boldsymbol{x} \texttt{ with}[j{:=}\frac{x_j}{2}])$ 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 given by Formula (4) 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

$$q\left(\boldsymbol{x} \texttt{ with}\left[j := \frac{x_j}{2}\right]\right) = \sum_{\boldsymbol{i}\le\boldsymbol{k}\le\boldsymbol{i} \texttt{ with}[j:=n_j]} \frac{1}{2^{k_j}}\binom{k_j}{i_j} B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x}). \tag{10}$$

[1]Formulas for the Bernstein coefficients on arbitrary divisions of the unit box are presented in [2].

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

$$p\left(\boldsymbol{x}\ \texttt{with}\left[j := \frac{x_j}{2}\right]\right) = \sum_{\boldsymbol{i}\le\boldsymbol{n}}\left(\hat{b}_{\boldsymbol{i}} \sum_{\boldsymbol{i}\le\boldsymbol{k}\le(\boldsymbol{i}\ \texttt{with}\,[j:=n_j])} \frac{1}{2^{k_j}}\binom{k_j}{i_j} B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x})\right)$$
$$= \sum_{\boldsymbol{k}\le\boldsymbol{n}}\left(\sum_{r=0}^{k_j} \frac{1}{2^{k_j}}\binom{k_j}{r}\hat{b}_{\boldsymbol{k}\ \texttt{with}\,[j:=r]}\right) B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x}).$$

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

$$p\left(\boldsymbol{x}\ \ \texttt{with}\left[j := \frac{x_j+1}{2}\right]\right) = p\left(\boldsymbol{x}\ \texttt{with}\left[j := 1 - \frac{1-x_j}{2}\right]\right)$$
$$= \sum_{\boldsymbol{k}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{k}\ \texttt{with}\,[j:=n_j-k_j]} B_{\boldsymbol{n},\boldsymbol{k}}\left(\boldsymbol{x}\ \texttt{with}\left[j := \frac{1-x_j}{2}\right]\right),\ \text{from definition of } B_{\boldsymbol{n},\boldsymbol{k}}.$$

The proof continues by applying Formula (10) to the case where the variable x_j is replaced by $1 - x_j$,

$$p\left(\boldsymbol{x}\ \ \texttt{with}\left[j := \frac{x_j+1}{2}\right]\right)$$
$$= \sum_{\boldsymbol{k}\le\boldsymbol{n}}\left(\sum_{r=0}^{k_j} \frac{1}{2^{k_j}}\binom{k_j}{r}\hat{b}_{\boldsymbol{k}\ \texttt{with}\,[j:=n_j-r]}\right) B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x}\ \texttt{with}\,[j:=1-x_j])$$
$$= \sum_{\boldsymbol{k}\le\boldsymbol{n}}\left(\sum_{r=0}^{n_j-k_j} \frac{1}{2^{n_j-k_j}}\binom{n_j-k_j}{r}\hat{b}_{\boldsymbol{k}\ \texttt{with}\,[j:=n_j-r]}\right) B_{\boldsymbol{n},\boldsymbol{k}}(\boldsymbol{x}\ \texttt{with}\,[j:=x_j])$$

□

The following proposition, which follows directly from Propositions 3 and 5, enables the use of the subdivision method to improve the accuracy of the estimates for the minimum and maximum values of a polynomial on the unit box given by formulas (5) and (7).

Proposition 6 *Let $p(\boldsymbol{x}) = \sum_{\boldsymbol{i}\le\boldsymbol{n}} \hat{b}_{\boldsymbol{i}}\, B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ be an m-variate polynomial in Bernstein form, r be a real number, and $\Re$ be a real order in $\{\le, <, \ge, >\}$. If $\hat{b}^L_{\boldsymbol{i}} \Re r$ and $\hat{b}^R_{\boldsymbol{i}} \Re r$, for all $\boldsymbol{i} \le \boldsymbol{n}$, then $p(\boldsymbol{x}) \Re r$, for all $\boldsymbol{x} \in \mathbb{U}^m$.*

2.4 Solving Simply Quantified Polynomial Inequalities

A simply quantified polynomial inequality on a bounded box is a first-order proposition of the form $\Box\, \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon\ p(\boldsymbol{x})\ \Re\ r$, where $\Box$ is either a universal quantifier $\forall$ or an existential quantifier $\exists$, $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box, p is an m-variate polynomial of degree at most $\boldsymbol{n}$, $\Re$ is a real order in $\{\le, <, \ge, >\}$, and r is a real number. The real order relation $\neg\Re$ denotes the negated relation of $\Re$, i.e., for all $r_1, r_2 \in \mathbb{R}$, $r_1 \neg\Re\ r_2$ if and only if $\neg(r_1 \ \Re \ r_2)$. The branch and bound procedure in Fig. 1 can be used to check whether the universally quantified polynomial inequality

$$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon\ p(\boldsymbol{x}) \ \Re \ r, \tag{11}$$

holds or not, and if not to find a counterexample. If the procedure states that the polynomial inequality $p^*(\boldsymbol{x}) \ \Re \ r$ holds for all $\boldsymbol{x} \in \mathbb{U}^m$, then, by Proposition 1, Formula (11) holds. If the procedure above states that the polynomial inequality $p^*(\boldsymbol{x}) \ \Re \ r$ does not hold for some $\boldsymbol{x} \in \mathbb{U}^m$, then Formula (11) does not hold for $\boldsymbol{y} \in [\boldsymbol{a}, \boldsymbol{b}]$ 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

$$\exists \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon\ p(\boldsymbol{x}) \ \Re \ r \tag{12}$$

holds, i.e., whether $p(\boldsymbol{x}) \ \Re \ r$ is satisfiable or not, and if so to find a witness, the procedure is used on the universally quantified formula $\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]\colon\ p(\boldsymbol{x}) \ \neg\Re \ r$. If the procedure states that the polynomial inequality $p^*(\boldsymbol{x}) \ \neg\Re \ r$ does not hold for some $\boldsymbol{x} \in \mathbb{U}^m$, then Formula (12) holds for $\boldsymbol{y} \in [\boldsymbol{a}, \boldsymbol{b}]$ defined as $y_j = a_j + x_j \cdot (b_j - a_j)$, for $0 \leq j < m$. If the procedure states that the polynomial inequality $p^*(\boldsymbol{x}) \ \neg\Re \ r$ holds for all $\boldsymbol{x} \in \mathbb{U}^m$, then, Formula (12) does not hold.

The procedure given in Fig. 1 is complete for verifying strict inequalities assuming that the method for selecting a variable for the subdivision method is fair, i.e., that every variable is eventually chosen an infinite number of times in every recursive branch. The completeness result follows from the fact that any continuous function (in this case a polynomial) on a bounded box $[\boldsymbol{a}, \boldsymbol{b}]$ attains a minimum on that box. With a fair subdivision method, the bounds on the maximum and minimum values of the polynomial converge to those values as the recursion goes to infinity.

1. Let $p^*(\boldsymbol{x}) = \sum_{\boldsymbol{k} \leq \boldsymbol{n}} r_{\boldsymbol{k}} \, \boldsymbol{x}^{\boldsymbol{k}}$, where

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

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

Fig. 1 Branch and bound procedure for solving universally quantified polynomial inequalities

In the case of non-strict inequalities, the method is not complete. Indeed, it does not terminate in cases such as the polynomial inequality $9x^2 - 6x + 1 \geq 0$ for all $x \in [0, 1]$. The polynomial attains its minimum 0 at the point $\frac{1}{3}$. The point $\frac{1}{3}$ is never a point attained at an endindex during the recursion, so there will always be a small interval on whose interior the function attains its minimum, and the result can not be proved on that interval. However, the method does terminate for the polynomial inequality $4x^2 - 4x + 1 \geq 0$ for all $x \in [0, 1]$, even though the polynomial attains the value 0 at the point $x = \frac{1}{2}$. This is because the subdivision scheme will split the interval $[0, 1]$ exactly at this point, and the inequalities on the resulting sub-intervals will be proved immediately by the Bernstein coefficients. In general, given a polynomial inequality that is not strict, where the polynomial actually attains the value given by the bound on the given box, the method will terminate only if it eventually subdivides exactly at each of the points where the polynomial attains that value.

Due to the subdivision technique used in Step 5, the complexity of the branch and bound procedure described in Fig. 1 is, in the worst case, at least exponential in the number of variables. Neither the completeness result nor the complexity analysis is part of the formal development presented in this paper. However, as the rest of this paper illustrates, the completeness result is not necessary for the development of practical proof producing strategies based on this procedure for verifying simply quantified polynomial inequalities.

2.5 Partially Open and Partially Unbounded Boxes

The branch and bound algorithm described in Section 2.4 can be modified to verify simply quantified polynomial inequalities on boxes that are partially open or partially unbounded. Problems of these types are reduced to problems on bounded boxes.

A *partially open m-box* ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, where $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box and $\boldsymbol{l}, \boldsymbol{u}$ are m-indices, is the set

$$\{\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] \mid \forall j < m \colon a_j \prec_{l_j} x_j \prec_{u_j} b_j\}, \tag{13}$$

where the relation $\prec_k$, for $k \in \mathbb{N}$, is the real order given by $\leq$ when $k = 0$ and $<$ when $k \neq 0$.

Given a bounded m-box $[\boldsymbol{a}, \boldsymbol{b}]$, the natural numbers l_j and u_j, with $j < m$, determine whether the lower bound and upper bound, respectively, of the j-th interval of ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ formed by the real numbers a_j and b_j is closed or open with 0 denoting closed and any other value denoting open. Using this notation, a bounded box $[\boldsymbol{a}, \boldsymbol{b}]$ is a partially open box ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, where $\boldsymbol{l} = \boldsymbol{u} = \boldsymbol{0}^m$. A completely open box $(\boldsymbol{a}, \boldsymbol{b})$ can be defined as ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, where $\boldsymbol{l} = \boldsymbol{u} = \boldsymbol{1}^m$.

The following trivial proposition reduces a universally quantified polynomial inequality on a partially open box to a universally quantified polynomial inequality on a closed box.

Proposition 7 *For all m-variate polynomials p, bounded m-boxes* $[\boldsymbol{a}, \boldsymbol{b}]$*, m-indices* $\boldsymbol{l}, \boldsymbol{u}$*, real numbers r, and real orders* $\Re$ *in* $\{\leq, <, \geq, >\}$*, if* $p(\boldsymbol{x}) \Re r$ *holds for all* $\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]$*, then* $p(\boldsymbol{x}) \Re r$ *holds for all* $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$*.*

The reciprocal implication of Proposition 7 does not hold in general. Therefore, the procedure in Fig. 1 is not complete for verifying polynomial inequalities on partially open boxes. In particular, the procedure does not succeed to verify the formula $\forall x \in (0,1)\colon x^2 > 0$. Furthermore, if the procedure determines that the formula $p(\boldsymbol{x}) \, \Re \, r$ does not hold for $\boldsymbol{y} \in [\boldsymbol{a}, \boldsymbol{b}]$, it does not mean that $\boldsymbol{y}$ is a counterexample to the formula $\forall \boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}} : \; p(\boldsymbol{x}) \, \Re \, r$. However, by updating the information on $\boldsymbol{l}$ and $\boldsymbol{u}$ at every recursive step, it is possible to modify Step 4 of the algorithm in Fig. 1 to return valid counterexamples to universally quantified polynomial inequalities on partially open boxes. That technique is described in the verified algorithm presented in Section 4.

A *partially unbounded m-box* ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$, where $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box and $\boldsymbol{l}, \boldsymbol{u}$ are m-indices such that for $l_j + u_j \leq 3$, for $j < m$, is the set

$$\left\{ \boldsymbol{x} \in \mathbb{R}^m \mid \forall j < m\colon a_j \prec_{l_j} x_j, \text{ if } l_j \leq 1, \text{ and } x_j \prec_{u_j} b_j, \text{ if } u_j \leq 1 \right\}. \tag{14}$$

As in the case of partially open boxes, the natural numbers l_j and u_j, with $j < m$, determine whether the lower bound and upper bound, respectively, of the j-th interval of ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ is closed, open bounded, or open unbounded with 0 denoting closed, 1 denoting open bounded, and any other value denoting open unbounded. The restriction $l_j + u_j \leq 3$, for $j < m$, states that any interval of a partially unbounded box is bounded in at least one end. In the case where $l_j > 1$ (resp. $u_j > 1$), the value of a_j (resp. b_j) is irrelevant in the definition of ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$. However, for technical reasons, whenever the partially unbounded box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ is referred to, it it is still assumed that $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box, i.e., $a_j < b_j$ for all $j < m$.

Bernstein polynomials are generally used as tools to optimize polynomials over bounded boxes. However, any universally quantified polynomial inequality on a partially unbounded box can be reduced to a similar inequality on a bounded box, so Bernstein polynomials can therefore be used for polynomial inequalities on partially unbounded boxes as well. The key insight here is that if $s_0, \ldots, s_{m-1}$ and $q_0, \ldots, q_{m-1}$ are univariate polynomials and $\theta\colon {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}} \to {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ is a function defined by

$$\theta(\boldsymbol{x}) = \left(\frac{s_0(x_0)}{q_0(x_0)}, \ldots, \frac{s_{m-1}(x_{m-1})}{q_{m-1}(x_{m-1})} \right)$$

that is a bijection, then for any polynomial $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$,

$$p^*(\boldsymbol{x}) = (q_0(x_0)^{n_0} \cdots q_{m-1}(x_{m-1})^{n_{m-1}}) \cdot p(\theta(\boldsymbol{x}))$$

is a polynomial. Thus, if $q_j(x_j) > 0$ for all $\boldsymbol{x} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ and $j < m$, then the universally quantified polynomial inequality $\forall \boldsymbol{x} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}\colon p(\boldsymbol{x}) \, \Re \, 0$ is equivalent to the inequality $\forall \boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}\colon p^*(\boldsymbol{x}) \, \Re \, 0$, the latter of which is on a bounded, partially open box. Since θ is a bijection, if $\boldsymbol{c} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ is a counterexample to the second inequality, in the sense that $p^*(\boldsymbol{c}) \neg\Re \, 0$ holds, then $\theta^{-1}(\boldsymbol{c}) \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ is a counterexample to the first inequality, in the sense that $p(\theta^{-1}(\boldsymbol{c})) \neg\Re \, 0$ holds. Furthermore, if each polynomial s_j or q_j is of degree at most 1, then p^* can be written as $p^*(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c^*_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$, where the $c^*_{\boldsymbol{i}}$ are real numbers that will be calculated exactly in a later section. What makes these facts useful is that there actually exist such polynomials, with degree at most 1, such that θ is a bijection. This depends on the fact that the box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ is unbounded, in each variable, at no more than one end. The bijection $\theta : {}_{l}[\boldsymbol{a}, \boldsymbol{b}\,]_{\boldsymbol{u}} \to {}_{l}\{\boldsymbol{a}, \boldsymbol{b}\,\}_{\boldsymbol{u}}$ is defined component-wise by

$$\theta(\boldsymbol{x})_j = \begin{cases} x_j & \text{if } l_j \le 1 \text{ and } u_j \le 1, \\ \dfrac{(b_j+1)\cdot x_j - b_j \cdot (a_j+1)}{x_j - a_j} & \text{if } l_j > 1 \text{ and } u_j \le 1, \\ \dfrac{(1-a_j)\cdot x_j + a_j \cdot (b_j - 1)}{b_j - x_j} & \text{if } l_j \le 1 \text{ and } u_j > 1. \end{cases}$$

Note that if θ is bijective, then its inverse is easily computed by using the fact that there is a surjective homomorphism from the group of invertible 2-by-2 matrices onto the group of non-constant functions of the form $x \mapsto \frac{r_1 \cdot x + r_2}{r_3 \cdot x + r_4}$ under composition (the group of "Möbius transformations"), where $r_3 \cdot x + r_4 \neq 0$. The homomorphism maps the matrix $\begin{pmatrix} r_1 & r_2 \\ r_3 & r_4 \end{pmatrix}$ to this function. Further, since it is a homorphism, the inverse of this matrix, which has a well-known formula, maps to the inverse of this function under composition.

The following proposition combines the reasoning above to reduce a given universally quantified polynomial inequality on a partially unbounded box to a universally quantified polynomial inequality on a partially open box.

Proposition 8 *For any m-variate polynomial p on a partially unbounded m-box ${}_{l}\{\boldsymbol{a}, \boldsymbol{b}\,\}_{\boldsymbol{u}}$, and real order $\Re$ in $\{\le, <, \ge, >\}$, there is an m-variate polynomial p^* on the partially open m-box ${}_{l}[\boldsymbol{a}, \boldsymbol{b}\,]_{\boldsymbol{u}}$ such that:*

- *If $p^*(\boldsymbol{x}) \Re 0$ holds for all $\boldsymbol{x} \in {}_{l}[\boldsymbol{a}, \boldsymbol{b}\,]_{\boldsymbol{u}}$ then $p(\boldsymbol{x}) \Re 0$ holds for all $\boldsymbol{x} \in {}_{l}\{\boldsymbol{a}, \boldsymbol{b}\,\}_{\boldsymbol{u}}$,*
- *if $p^*(\boldsymbol{c}) \neg\Re 0$, where $\boldsymbol{c} \in {}_{l}[\boldsymbol{a}, \boldsymbol{b}\,]_{\boldsymbol{u}}$, then there exists $\boldsymbol{c}^* \in {}_{l}\{\boldsymbol{a}, \boldsymbol{b}\,\}_{\boldsymbol{u}}$ that is computable from $\boldsymbol{c}$, such that $p(\boldsymbol{c}^*) \neg\Re 0$.*

Formulas for p^* and $\boldsymbol{c}^*$ will be given in a later section, in the context of specific data structures for representing polynomials.

3 Formalization of Polynomials

A key aspect in any algorithmic application involving multivariate polynomials is the data structure used to represent polynomials. In [41], Zippel identifies three decision points to take into account when choosing a particular polynomial representation:

- Expanded vs. recursive representation.
- Variable sparse vs. variable dense.
- Degree sparse vs. degree dense.

A multivariate polynomial in expanded representation is seen as a list of pairs of exponent vectors and coefficients. In a recursive representation, univariate polynomials are defined such that the coefficients are members of an arbitrary ring structure. Since polynomials form a ring, a polynomial on the variables $x_0, \ldots, x_m$, with $m > 0$, can be recursively represented as a univariate polynomial on x_0, where the coefficients are polynomials on the variables $x_1, \ldots, x_n$. Variable sparse/dense refers to a representation of a polynomial, where each monomial occurring it its expansion excludes (respectively includes) all variables occurring with exponent 0 in that monomial. Degree sparse/dense refers to a representation of a polynomial where each monomial that has a coefficient of 0 in the polynomial is excluded (respectively included) in its representation.

For the formal development of Bernstein polynomials presented in this paper, an expanded and variable dense representation has been chosen that is degree dense for univariate polynomials. An expanded representation is convenient since the order in which variables will be subdivided may be unknown. In a recursive representation, subdividing a variable that is not the outermost variable of the recursive representation can be cumbersome. A variable dense representation is often used with an expanded representation so that the monomials all have the same number of variables. A degree sparse representation allows for a compact expanded representation. Unfortunately, the sparseness of a polynomial, which is the ratio of the number of monomials with non-zero coefficient to the total number of possible monomials, is not preserved by some of the polynomial transformations presented in Section 2.

An expanded, variable dense, and degree dense representation of a polynomial may be memory-wise expensive. Consider the polynomial

$$p(x, y) = x^{99}y^{999} - 3x^{99} - 2xy^{999} + 6x. \tag{15}$$

The total number of possible monomials in p, including all degrees up to $(99, 999)$ for (x, y), is 10^5. Of these monomials, all but four have a zero coefficient. In [38], Smith introduces an expanded and yet compact representation of multivariate polynomials in Bernstein form. In Smith's representation, the Bernstein coefficients of a polynomial in Bernstein form are not explicitly computed. This feature is used in [38] to propose an acceleration technique for optimization algorithms based on Bernstein polynomials. That strategy is not implemented in this paper. Smith's representation is used in the formal development presented here as a general polynomial representation technique. In addition to allowing for lazy computations of coefficients, it is more compact than other representations of multivariate polynomials and often allows proofs of properties about multivariate polynomials to be reduced to proofs for the univariate case.

This section presents a formalization of univariate and multivariate polynomials. The theorems presented in this section have been mechanically verified in PVS. They correspond to the propositions presented in Section 2 and, from a logical point of view, the formal proofs of these theorems follow the proofs of the propositions presented in that section.

3.1 Formalization of Univariate Polynomials

In PVS, univariate polynomials are formalized using a degree dense representation. More precisely, a univariate polynomial p of degree at most n written in either the form $p(x) = \sum_{i=0}^{n} a_i x^i$ or the form $p(x) = \sum_{i=0}^{n} a_i \binom{n}{i} x^i (1-x)^{n-i}$, is represented by the $(n+1)$-tuple $(a_0, \ldots, a_n)$. These two forms are referred to here as *standard form* and *Bernstein form*, respectively. Using this representation, the memory necessary to represent a polynomial p of degree at most n is of the order of $n + 1$.

Since a tuple can correspond to a polynomial in either standard or Bernstein form, it does not uniquely determine the polynomial it represents. Thus, two polynomial evaluation functions are defined on tuples that correspond to these forms. The function `eval` takes as a parameter a tuple $\boldsymbol{a}$ and returns a function on real numbers that corresponds to the polynomial in standard form represented by $\boldsymbol{a}$. It is defined by

$$\texttt{eval}(\boldsymbol{a})(x) \;\equiv\; \sum_{i=0}^{n} a_i x^i,$$

where $x \in \mathbb{R}$ and $\boldsymbol{a}$ is an $(n+1)$-tuple. The function `evalbern` takes as a parameter a tuple $\boldsymbol{a}$ and returns a function on real numbers that corresponds to the polynomial in Bernstein form represented by $\boldsymbol{a}$. It is defined by

$$\texttt{evalbern}(\boldsymbol{a})(x) \;\equiv\; \sum_{i=0}^{n} a_i \binom{n}{i} x^i (1-x)^{n-i},$$

where, as above, $x \in \mathbb{R}$ and $\boldsymbol{a}$ is an $(n+1)$-tuple.

The function `tobern` takes as input a tuple $\boldsymbol{a}$ representing a univariate polynomial p in standard form and returns a tuple, with the same number of elements as $\boldsymbol{a}$, that corresponds to p written in Bernstein form. It is defined by

$$\texttt{tobern}(\boldsymbol{a})_i \;\equiv\; \sum_{k=0}^{i} a_k \frac{\binom{i}{k}}{\binom{n}{k}}, \tag{16}$$

where $\boldsymbol{a}$ is an $(n+1)$-tuple and $i \leq n$.

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

Theorem 1 *For all tuples* $\boldsymbol{a}$ *and real numbers* x,

$$\texttt{eval}(\boldsymbol{a})(x) = \texttt{eval}(\texttt{tobern}(\boldsymbol{a}))(x).$$

The function `tobern` takes as inputs a tuple $\boldsymbol{a}$ representing a univariate polynomial p in standard form, as well as two real numbers a and b. It returns a tuple, with the same number of elements as $\boldsymbol{a}$, that corresponds to a polynomial on the unit interval [0, 1] that attains the same values that p does in the interval $[a, b]$. It is defined by

$$\texttt{translate}(\boldsymbol{a}, a, b)_i \;\equiv\; (b-a)^i \sum_{k=i}^{n} a_k \binom{k}{i} a^{k-i}, \tag{17}$$

where, as above, $\boldsymbol{a}$ is an $(n+1)$-tuple and $i \leq n$. The following theorem has been formally proved in PVS. It is a formal statement of Proposition 1 for the case of univariate polynomials.

Theorem 2 *For all tuples* $\boldsymbol{a}$ *and real numbers* a, b, *and* x,

$$\texttt{eval}(\boldsymbol{a})(a + x(b-a)) = \texttt{eval}(\texttt{translate}(\boldsymbol{a}, a, b))(x).$$

Domain subdivision for univariate polynomials is accomplished by the functions `subdivl` and `subdivr`. These functions take as input a tuple $\boldsymbol{a}$ representing a univariate polynomial p written in Bernstein form. They each return a tuple with the same number of elements as $\boldsymbol{a}$, which corresponds to a polynomial written in Bernstein form. They are defined by

$$\texttt{subdivl}(\boldsymbol{a})_i \equiv \frac{1}{2^i}\sum_{k=0}^{i}\binom{i}{k}a_k,$$

$$\texttt{subdivr}(\boldsymbol{a})_i \equiv \frac{1}{2^{n-i}}\sum_{k=0}^{n-i}\binom{n-i}{k}a_{n-k}, \tag{18}$$

where $\boldsymbol{a}$ is an $(n+1)$-tuple and $i \leq n$.

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

Theorem 3 *For all tuples* $\boldsymbol{a}$ *and real numbers* x*,*

$$\mathit{evalbern}(\mathit{subdivl}(\boldsymbol{a}))(x) = \mathit{evalbern}(\boldsymbol{a})\left(\frac{x}{2}\right),$$

$$\mathit{evalbern}(\mathit{subdivr}(\boldsymbol{a}))(x) = \mathit{evalbern}(\boldsymbol{a})\left(\frac{x+1}{2}\right).$$

3.2 Smith's Representation

Smith's representation is based on the fact that any m-variate polynomial p of degree at most $\boldsymbol{n}$, where $\boldsymbol{n}$ is an m-index, can be written in the form

$$p(\boldsymbol{x}) = \sum_{k=0}^{t-1} q_k \prod_{j=0}^{m-1} p_{k,j}(x_j), \tag{19}$$

where, for $k < t$ and $j < m$, $q_k \neq 0$ and $p_{k,j}$ is a univariate polynomial of degree at most n_j on variable x_j. Indeed, since an m-variate monomial $\boldsymbol{x}^{\boldsymbol{i}}$ has the form $\prod_{j=0}^{m-1} x_j^{i_j}$, an m-variate polynomial p of the form $p(\boldsymbol{x}) = \sum_{\boldsymbol{i}\leq\boldsymbol{n}} c_{\boldsymbol{i}}\boldsymbol{x}^{\boldsymbol{i}}$ has also the form $p(x) = \sum_{k=0}^{t-1} q_k \prod_{j=0}^{m-1} x_j^{i_j}$, where t is the number of monomials with non-zero coefficient in p.

A Smith's representation of a polynomial p written in the form of Formula (19) consists of a t-tuple $\boldsymbol{q} = (q_0, \ldots, q_{t-1})$ and a list of t elements representing each product $\prod_{j=0}^{m-1} p_{k,j}(x_j)$, i.e., the k-th element of the list, with $k < t$, is a list of length m, representing each univariate polynomial $p_{k,j}(x_j)$ in a degree dense form. There is not a unique Smith's representation of a polynomial, and the number of terms t may change for different representations. Further, each univariate polynomial $p_{k,j}$ can be a polynomial in either standard form or Bernstein form.

Example 2 The degree of the 2-variate polynomial p in Formula (15) is (99,999). A Smith's representation of p that corresponds to the form $p(x, y) = x^{99}y^{999} - 3x^{99}y^0 - 2xy^{999} + 6xy^0$ consists of the 4-tuple $\boldsymbol{q} = (1, -3, -2, 6)$ and a list of 4 elements representing the products $x^{99}y^{999}$, $x^{99}y^0$, xy^{999}, and xy^0, respectively. Each element in that list consists of a list of two tuples, one per variable.

– The first element consists of $\mathbf{0}^{100}$ `with` $[99 := 1]$ and $\mathbf{0}^{1000}$ `with` $[999 := 1]$, which corresponds to the degree dense representations of $p_{0,0}(x) = x^{99}$ and $p_{0,1}(y) = y^{999}$, respectively.

- The second element consists of $\mathbf{0}^{100}$ `with` $[99 := 1]$ and $\mathbf{0}^{1000}$ `with` $[0 := 1]$, which corresponds to the degree dense representations of $p_{1,0}(x) = x^{99}$ and $p_{1,1}(y) = 1$, respectively.
- The third element consists of $\mathbf{0}^{100}$ `with` $[1 := 1]$ and $\mathbf{0}^{1000}$ `with` $[999 := 1]$, which corresponds to the degree dense representations of $p_{2,0}(x) = x$ and $p_{2,1}(y) = y^{999}$, respectively.
- The last element consists of $\mathbf{0}^{100}$ `with` $[1 := 1]$ and $\mathbf{0}^{1000}$ `with` $[0 := 1]$, which corresponds to the degree dense representations of $p_{3,0}(x) = x$ and $p_{3,1}(y) = 1$, respectively.

An alternative representation of p based on the form $p(x, y) = (x^{99} - 2x)(y^{999} - 3y^0)$ consists of the 1-tuple $\boldsymbol{q} = (1)$ and a list of one element. That element consists of the tuples $\mathbf{0}^{100}$ `with` $[1 := -2, 99 := 1]$ and $\mathbf{0}^{1000}$ `with` $[0 := -3, 999 := 1]$, which corresponds to the degree dense representation of $x^{99} - 2x$ and $y^{999} - 3y^0$, respectively.

One advantage of this representation of polynomials is that multivariate polynomials are seen as a collection of univariate polynomials. Thus, verifying properties of multivariate polynomials often reduces to proving them for univariate polynomials. This makes Smith's representation of multivariate polynomials appealing for applications in theorem proving.

3.3 Formalization of Multivariate Polynomials

In PVS, multivariate polynomials are represented using Smith's representation. In fact, there is one datatype, each of whose elements can represent either a standard or a Bernstein representation of a polynomial. A pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, where $\boldsymbol{q}$ is a tuple and $\boldsymbol{\alpha}$ is a list, is said to be an *m-variate polynomial pair of degree at most $\boldsymbol{n}$* if the following conditions hold.

- The number of elements in $\boldsymbol{q}$, which is written $|\boldsymbol{q}|$, is equal to the length of $\boldsymbol{\alpha}$, which is written $|\boldsymbol{\alpha}|$.
- The k-th element of $\boldsymbol{\alpha}$, denote $\boldsymbol{\alpha}(k)$, is a list of length m.
- For $j < m$, the j-th element of $\boldsymbol{\alpha}(k)$, denoted $\boldsymbol{\alpha}(k)(j)$, is an $(n_j + 1)$-tuple.

The memory used by such a pair is of the order of $|\boldsymbol{q}| \cdot (1 + \sum_{j=0}^{m-1}(n_j + 1))$.

The functions defined in Section 3.1 are used to define similar functions for the multivariate case. Two evaluation functions are defined on m-variate polynomial pairs of degree at most $\boldsymbol{n}$, corresponding representations of two distinct, unrelated a m-variate polynomials. One of them is a standard representation, and the other is a Bernstein representation. These two evaluation functions are defined as follows.

The evaluation function `evalmulti` takes as input an m-variate polynomial pair of degree at most $\boldsymbol{n}$, $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, and it returns a function on an m-tuple $\boldsymbol{x}$.

$$\texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \equiv \sum_{k=0}^{t-1} q_k \cdot \prod_{j=0}^{m-1} \texttt{eval}(\boldsymbol{\alpha}(k)(j))(x_j),$$

This function corresponds to Smith's representation of a polynomial in standard form.

Similarly, the evaluation function `evalmultibern` takes as input an m-variate polynomial pair of degree at most $\boldsymbol{n}$, $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, and it also returns a function on an m-tuple $\boldsymbol{x}$.

$$\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \equiv \sum_{k=0}^{t-1} q_k \cdot \prod_{j=0}^{m-1} \texttt{evalbern}(\boldsymbol{\alpha}(k)(j))(x_j),$$

This corresponds to Smith's representation of a polynomial in Bernstein form.

The most important property of the evaluation function `evalmulti` is that for any m-variate polynomial $p(\boldsymbol{x})$ of degree at most $\boldsymbol{n}$, there is an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, such that for all $\boldsymbol{x} \in \mathbb{R}^m$, $\texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) = p(\boldsymbol{x})$, in which case $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is said to be a *standard representation* of p. Similarly, the most important property of the evaluation function `evalmultibern` is that for any m-variate polynomial $p(\boldsymbol{x})$ of degree at most $\boldsymbol{n}$, there is an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, such that for all $\boldsymbol{x} \in \mathbb{R}^m$, $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) = p(\boldsymbol{x})$, in which case $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is said to be a *Bernstein representation* of p.

There is function `tomultibern` that takes as input a list $\boldsymbol{\alpha}$ that is part of an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, representing a univariate polynomial written in standard form. The function returns another list $\boldsymbol{\alpha}'$, having the same structure as $\boldsymbol{\alpha}$, such that every $(n_j + 1)$-tuple $\boldsymbol{\alpha}'(k)(j)$, for $k < |\boldsymbol{\alpha}|$ and $j < m$, represents the same univariate polynomial as $\boldsymbol{\alpha}(k)(j)$, but written in Bernstein form, i.e.,

$$\texttt{tomultibern}(\boldsymbol{\alpha}) \equiv \boldsymbol{\alpha}', \text{where } \boldsymbol{\alpha}'(k)(j) = \texttt{tobern}(\boldsymbol{\alpha}(k)(j)), \tag{20}$$

for all $k < |\boldsymbol{\alpha}|$ and $j < m$.

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 4 *For all m-variate polynomial pairs* $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$*, of degree at most* $\boldsymbol{n}$*, and* $\boldsymbol{x} \in \mathbb{R}^m$*,*

$$\textit{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) = \textit{evalmultibern}(\boldsymbol{q}, \textit{tomultibern}(\boldsymbol{\alpha}))(\boldsymbol{x}).$$

There is another a function `translatemulti` that takes as input a list $\boldsymbol{\alpha}$ that is part of an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, representing a univariate polynomial written in standard form, and an m-box $[\boldsymbol{a}, \boldsymbol{b}]$. The function returns another list $\boldsymbol{\alpha}'$, having the same structure as $\boldsymbol{\alpha}$, such that if every $(n_j + 1)$-tuple $\boldsymbol{\alpha}(k)(j)$, for $k < |\boldsymbol{\alpha}|$ and $j < m$, represents a univariate polynomial written in standard form, then $\boldsymbol{\alpha}'(k)(j)$ represents a univariate polynomial written in standard form that in the unit interval [0, 1] attains the same values as $\boldsymbol{\alpha}(k)(j)$ in the interval $[a_j, b_j]$, i.e.,

$$\begin{aligned} &\texttt{translatemulti}(\boldsymbol{\alpha}, \boldsymbol{a}, \boldsymbol{b}) \equiv \boldsymbol{\alpha}', \text{where} \\ &\qquad \boldsymbol{\alpha}'(k)(j) = \texttt{translate}(\boldsymbol{\alpha}(k)(j), a_j, b_j), \end{aligned} \tag{21}$$

for all $k < |\boldsymbol{\alpha}|$ and $j < m$.

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

Theorem 5 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, and for all m-boxes $[\boldsymbol{a}, \boldsymbol{b}\,]$ and tuples $\boldsymbol{x} \in \mathbb{R}^m$,*

$$\texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{y}) = \texttt{evalmulti}(\boldsymbol{q}, \texttt{translatemulti}(\boldsymbol{\alpha}, \boldsymbol{a}, \boldsymbol{b}\,))(\boldsymbol{x}),$$

where $y_j = a_j + x_j \cdot (b_j - a_j)$, for $j < m$.

Since the polynomials p_j^L and p_j^R from Formula (9) only affect the j-th variable of the polynomial p, Smith's representations of these polynomials can be computed by only subdividing the univariate polynomials corresponding to that variable. The functions `subdivlmulti` and `subdivrmulti` take as inputs a list $\boldsymbol{\alpha}$ and a natural number $j < m$. The list $\boldsymbol{\alpha}$ is assumed to come from a an m-variate polynomial pair of degree at most $\boldsymbol{n}$ that is a Bernstein representation of a given polynomial. These functions return, respectively, new lists $\boldsymbol{\alpha}^L$ and $\boldsymbol{\alpha}^R$ that have the same as structure as $\boldsymbol{\alpha}$ and are defined as follows.

$$\texttt{subdivlmulti}(\boldsymbol{\alpha}, j) \equiv \boldsymbol{\alpha}^L, \text{where}$$

$$\boldsymbol{\alpha}^L(k)(i) = \begin{cases} \boldsymbol{\alpha}(k)(i) & \text{if } i \neq j, \\ \texttt{subdivl}(\boldsymbol{\alpha}(k)(j)) & \text{otherwise,} \end{cases} \tag{22}$$

for $k < |\boldsymbol{\alpha}|$ and $i < m$.

$$\texttt{subdivrmulti}(\boldsymbol{\alpha}, j) \equiv \boldsymbol{\alpha}^R, \text{where}$$

$$\boldsymbol{\alpha}^R(k)(i) = \begin{cases} \boldsymbol{\alpha}(k)(i) & \text{if } i \neq j, \\ \texttt{subdivr}(\boldsymbol{\alpha}(k)(j)) & \text{otherwise.} \end{cases} \tag{23}$$

for $k < |\boldsymbol{\alpha}|$ and $i < m$.

The following theorem presents Proposition 5 as it has been proved in PVS for the case of multivariate polynomials. It states that if $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is a Bernstein representation of a multivariate polynomial p, then $\langle \boldsymbol{q}, \texttt{subdivlmult}(\boldsymbol{\alpha}, j) \rangle$ and $\langle \boldsymbol{q}, \texttt{subdivrmult}(\boldsymbol{\alpha}, j) \rangle$ are Bernstein representations of p_j^L and p_j^R, respectively. The proof uses Theorem 3.

Theorem 6 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, and for all natural numbers $j < m$,*

$$\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha}) \left(\boldsymbol{x} \texttt{ with} \left[j := \frac{x_j}{2} \right] \right)$$
$$= \texttt{evalmultibern}(\boldsymbol{q}, \texttt{subdivlmult}(\boldsymbol{\alpha}, j))(\boldsymbol{x}).$$

$$\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha}) \left(\boldsymbol{x} \texttt{ with} \left[j := \frac{x_j + 1}{2} \right] \right)$$
$$= \texttt{evalmultibern}(\boldsymbol{q}, \texttt{subdivrmult}(\boldsymbol{\alpha}, j))(\boldsymbol{x}).$$

3.4 Bernstein Coefficients

Let $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ be an m-variate polynomial pair of degree at most $\boldsymbol{n}$, and suppose that $p(\boldsymbol{x}) = \sum_{\boldsymbol{i} \leq \boldsymbol{n}} c_{\boldsymbol{i}} \boldsymbol{x}^{\boldsymbol{i}}$ is a polynomial of degree at most $\boldsymbol{n}$ in such that $p(\boldsymbol{x}) =$

$\texttt{evalmultibern}(\boldsymbol{q},\boldsymbol{\alpha})(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^m$. The function `multicoeff`, defined below, computes the coefficient $c_{\boldsymbol{i}}$, for $\boldsymbol{i} \leq \boldsymbol{n}$.

$$\texttt{multicoeff}(\boldsymbol{q},\boldsymbol{\alpha},\boldsymbol{i}) \equiv \sum_{k=0}^{|\boldsymbol{\alpha}|-1} q_i \prod_{j=0}^{m-1} \boldsymbol{\alpha}(k)(j)_{i_j}.$$

Similarly, if p is in Bernstein form and $\boldsymbol{i} \leq \boldsymbol{n}$, then $\texttt{multicoeff}(\boldsymbol{q},\boldsymbol{\alpha},\boldsymbol{i})$ is the Bernstein coefficient $\hat{b}_{\boldsymbol{i}}$ of p, i.e., the coefficient of the Bernstein basis polynomial $B_{\boldsymbol{n},\boldsymbol{i}}(\boldsymbol{x})$ (Section 2).

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

Theorem 7 *For all m-variate polynomial pairs $\langle \boldsymbol{q},\boldsymbol{\alpha}\rangle$, of degree at most $\boldsymbol{n}$, real orders $\Re \in \{\leq, <, \geq, >\}$, $r \in \mathbb{R}$, and $\boldsymbol{x} \in \mathbb{U}^m$, if for all m-indices $\boldsymbol{i} \leq \boldsymbol{n}$, $\texttt{multicoeff}(\boldsymbol{q},\boldsymbol{\alpha},\boldsymbol{i}) \,\Re\, r$, then*

$$\texttt{evalmultibern}(\boldsymbol{q},\boldsymbol{\alpha})(\boldsymbol{x}) \,\Re\, r.$$

The function $\texttt{endpoint}(\boldsymbol{a},\boldsymbol{b})$, where $[\boldsymbol{a},\boldsymbol{b}]$ is an m-box, translates an m-index to an m-tuple in $[\boldsymbol{a},\boldsymbol{b}]$. It is defined as follows.

$$\texttt{endpoint}(\boldsymbol{a},\boldsymbol{b})(\boldsymbol{i})_j \equiv \begin{cases} a_j & \text{if } i_j = 0, \\ b_j & \text{otherwise,} \end{cases} \tag{24}$$

where $\boldsymbol{i}$ is an m-index and $j < m$. The range of the function $\texttt{endpoint}(\boldsymbol{a},\boldsymbol{b})$ is the set of endpoints of $[\boldsymbol{a},\boldsymbol{b}]$ as defined in Section 2.2. This function establishes a correspondence between the set of endindices of $\boldsymbol{n}$ and the set of endpoints of $[\boldsymbol{a},\boldsymbol{b}]$. The following theorem is the formal version of Proposition 4. It states that the function `multicoeff` can be used to compute values of a polynomial.

Theorem 8 *For all m-variate polynomial pairs $\langle \boldsymbol{q},\boldsymbol{\alpha}\rangle$, of degree at most $\boldsymbol{n}$, and all endindices $\boldsymbol{k}$ of $\boldsymbol{n}$,*

$$\texttt{multicoeff}(\boldsymbol{q},\boldsymbol{\alpha},\boldsymbol{k}) = \texttt{evalmultibern}(\boldsymbol{q},\boldsymbol{\alpha})(\texttt{endpoint}(\boldsymbol{0}^m,\boldsymbol{1}^m)(\boldsymbol{k})). \tag{25}$$

3.5 Partially Open and Partially Unbounded Boxes

Let $\boldsymbol{l},\boldsymbol{u}$ be m-indices and $[\boldsymbol{a},\boldsymbol{b}]$ be an m-box. The predicate $\texttt{open}(\boldsymbol{l},\boldsymbol{u},\boldsymbol{a},\boldsymbol{b})$ on m-tuples characterizes the elements in $\mathbb{R}^m$ that are in the partially open box ${}_{\boldsymbol{l}}[\boldsymbol{a},\boldsymbol{b}]_{\boldsymbol{u}}$, i.e.,

$$\texttt{open}(\boldsymbol{l},\boldsymbol{u},\boldsymbol{a},\boldsymbol{b})(\boldsymbol{x}) \equiv \boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a},\boldsymbol{b}]_{\boldsymbol{u}}, \tag{26}$$

where $\boldsymbol{x} \in \mathbb{R}^m$. The predicate $\texttt{openindex}(\boldsymbol{l},\boldsymbol{u})$ on m-indices is defined as follows.

$$\begin{aligned} \texttt{openindex}(\boldsymbol{l},\boldsymbol{u})(\boldsymbol{k}) \equiv \forall j < m\colon & k_j \neq 0, \text{ if } l_j \neq 0, \text{ and} \\ & k_j = 0, \text{ if } u_j \neq 0, \end{aligned} \tag{27}$$

where $\boldsymbol{k} \in \mathbb{N}^m$. It is easy to check, by unfolding the definitions, that an endindex $\boldsymbol{k}$ of $\boldsymbol{n}$ that satisfies $\texttt{openindex}(\boldsymbol{l}, \boldsymbol{u})(\boldsymbol{k})$, produces an endpoint of $[\boldsymbol{a}, \boldsymbol{b}]$ that satisfies $\texttt{open}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})$.

Lemma 1 *For all m-indices $\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{n}$, m-boxes $[\boldsymbol{a}, \boldsymbol{b}]$, and m-indices $\boldsymbol{k} \in \mathcal{I}_n$, if $\texttt{openindex}(\boldsymbol{l}, \boldsymbol{u})(\boldsymbol{k})$ then $\texttt{open}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(\texttt{endpoint}(\boldsymbol{a}, \boldsymbol{b})(\boldsymbol{k}))$.*

Lemma 1 is used with Theorem 8 to find points in a partially open box evaluation satisfy a given polynomial inequality.

Let $\boldsymbol{l}$ and $\boldsymbol{u}$ be m-indices, and let $[\boldsymbol{a}, \boldsymbol{b}]$ be an m-box. Then the predicate $\texttt{unbounded}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})$ on m-tuples characterizes the elements in $\mathbb{R}^m$ that are in the partially unbounded box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$, i.e.,

$$\texttt{unbounded}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(\boldsymbol{x}) \equiv \boldsymbol{x} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}, \tag{28}$$

where $\boldsymbol{x} \in \mathbb{R}^m$.

Proposition 8 states that a polynomial inequality on a partially unbounded box can be translated into a polynomial inequality on a partially open box. This is accomplished through a function $\texttt{tr\_mob}$ (for "translate Möbius"), which takes as inputs real numbers A, B, C, D with either $C \neq 0$ or $D \neq 0$, and a tuple $\boldsymbol{a}$ representing a polynomial p in standard form. It returns a tuple of real numbers of the same length as $\boldsymbol{a}$ that represents the polynomial that is equal to $(C \cdot x + D)^n \cdot p((A \cdot x + B)/(C \cdot x + D))$ for all real numbers x such that $C \cdot x + D \neq 0$, where $\boldsymbol{a}$ is an $(n+1)$-tuple. The function $\texttt{tr\_mob}$ is defined as follows for $j < m$.

$$\texttt{tr\_mob}(A, B, C, D, \boldsymbol{a})_d$$
$$\equiv \sum_{i=0}^{n} a_i \cdot \sum_{k=\max(0,d-i)}^{\min(d,n-i)} A^{d-k} \cdot B^{k-d+i} \cdot C^k \cdot D^{n-k-i} \cdot \binom{n-i}{k} \cdot \binom{i}{d-k}.$$

The following lemma gives the key property for this function.

Lemma 2 *If A, B, C, D, and x are real numbers such that $C \cdot x + D \neq 0$ and $\boldsymbol{a}$ is a $(n+1)$-tuple of real numbers, then*

$$\texttt{eval}(\texttt{tr\_mob}(A, B, C, D, \boldsymbol{a}))(x) = (C \cdot x + D)^n \cdot \texttt{eval}(\boldsymbol{a})\left(\frac{A \cdot x + B}{C \cdot x + D}\right).$$

The translation of a polynomial inequality on a partially unbounded box to an inequality on a partially open box is accomplished through functions $\texttt{translatebound}$ and $\texttt{counterexbound}$, which depend on the function $\texttt{tr\_mob}$. The function $\texttt{translatebound}$ is defined by

$$\texttt{translatebound}(\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}) \equiv \boldsymbol{\alpha}', \text{ where } \boldsymbol{\alpha}'(k)(j)$$
$$= \begin{cases} \boldsymbol{\alpha}(k)(j) & \text{if } l_j \leq 1 \text{ and } u_j \leq 1 \\ \texttt{tr\_mob}(b_j + 1, -b_j \cdot (a_j + 1), 1, -a_j, \boldsymbol{\alpha}(k)(j)) & \leq \text{if } l_j > 1 \text{ and } u_j \leq 1 \\ \texttt{tr\_mob}(1 - a_j, a_j \cdot (b_j - 1), -1, b_j, \boldsymbol{\alpha}(k)(j)) & \text{if } l_j \leq 1 \text{ and } u_j > 1, \end{cases}$$

where $\boldsymbol{l}, \boldsymbol{u}$ are m-indices, $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box, $k < |\boldsymbol{\alpha}|$, $j < m$, $l_j + u_j \leq 3$, and $\boldsymbol{\alpha}$ is part of an m-variate polynomial pair of degree at most $\boldsymbol{n}$. Similarly,

$$\begin{aligned} &\texttt{counterexbound}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(\boldsymbol{x})_j \\ &\equiv \begin{cases} x_j & \text{if if } l_j \leq 1 \text{ and } u_j \leq 1 \\ (-a_j \cdot x + b_j \cdot (a_j + 1))/(-x + b_j + 1) & \text{if if } l_j > 1 \text{ and } u_j \leq 1 \\ (b_j \cdot x - a_j \cdot (b_j - 1))/(x + 1 - a_j) & \text{if if } l_j \leq 1 \text{ and } u_j > 1, \end{cases} \end{aligned}$$

where $\boldsymbol{l}, \boldsymbol{u}$ are m-indices, $[\boldsymbol{a}, \boldsymbol{b}]$ is an m-box, $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, $j < m$, and $l_j + u_j \leq 3$. It can be proved that the function $\texttt{counterexbound}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})$ is a bijection between ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ and ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$.

The following theorem formalizes Proposition 8.

Theorem 9 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, m-indices $\boldsymbol{l}, \boldsymbol{u}$, with $l_j + u_j \leq 3$ for $j < m$, m-boxes $[\boldsymbol{a}, \boldsymbol{b}]$, real orders $\Re \in \{\leq, <, \geq, >\}$, and $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, since* `counterexbound` *is bijective, if for all $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$,*

$$\texttt{evalmulti}(\boldsymbol{q}, \texttt{translatebound}(\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}))(\boldsymbol{x}) \ \Re\ 0,$$

then for all $\boldsymbol{y} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$, $\texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{y}) \ \Re\ 0$. *Furthermore, if*

$$\texttt{evalmulti}(\boldsymbol{q}, \texttt{translatebound}(\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}))(\boldsymbol{x}) \ \neg\Re\ 0$$

then

$$\texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\texttt{counterexbound}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(\boldsymbol{x})) \ \neg\Re\ 0.$$

3.6 Note About Formalization in PVS

The formalization of polynomials described in this section uses the pre-defined PVS types `nat` and `reals`, for natural and real numbers, respectively, and defined types for m-tuples and lists. In higher-order proof assistants such as PVS, there are many ways in which these types of structures can be defined. The PVS development presented in this paper uses functional terms to represent both m-tuples and lists. More precisely, an m-tuple $\boldsymbol{q}$ is formalized in PVS as a function `q` from `nat` into `real` such that $\texttt{q}(j) = q_j$, for $j < m$, and $\texttt{q}(j) = 0$, for $j \geq m$. Furthermore, a list $\boldsymbol{\alpha}$ of t elements, used in the representation of an m-variate polynomial of degree at most $\boldsymbol{n}$, is represented by a function `A` of type `nat` $\rightarrow$ `nat` $\rightarrow$ `nat` $\rightarrow$ `real` such that $\texttt{A}(k)(j)(i) = \boldsymbol{\alpha}(k)(j)_i$, if $k < |\boldsymbol{\alpha}|$, $j < m$, and $i \leq n_j$. Otherwise, $\texttt{A}(k)(j)(i) = 0$.

It is emphasized that nothing in this paper fundamentally depends on the concrete data types used to represent m-tuples and lists. However, the authors have found that a functional representation is convenient in PVS. For instance, by using this formalization, the degree dense representation of univariate polynomials presented here corresponds to the existing formalization of univariate polynomials available as part of the NASA Libraries.[2] The PVS prelude library includes a type `list` defined as an Abstract Data Type. An advantage of a functional type over `list` is that the overwriting operator `WITH`, extensively used in this development, is available

[2]http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html

for functions but not for terms of type `list`. Furthermore, functions in PVS can be partially applied and the development presented here takes advantage of this feature. For example, A$(k)(j)(i)$ has the type `real` and represents the coefficient of the degree i of the univariate polynomial corresponding to the variable j in the k-th term of the list A; A$(k)(j)$ has the type `nat` $\rightarrow$ `real`, which is the type of an m-tuple, and represents the univariate polynomial corresponding to the variable j in the k-th term of the list A; and so forth. The use of functions with the unbounded domain `nat` rather than a bounded domain simplifies the formal development by avoiding generation of Type Correctness Conditions (TCCs) on expressions involving access to m-tuples and lists. Those TCCs are typically easy to discharge but they are an additional nuance on an already complex development. The main drawback of this functional representation is that since the actual lengths of m-tuples and lists do not appear in the data-structures, they have to be explicitly carried out in all definitions.

In the paper, the notation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, which represents a multivariate polynomial pair, is used for notational convenience. However, the pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is not explicitly represented as a datatype in PVS. Hence, the PVS formalization of polynomials described in this section is not a deep embedding of multivariate polynomials. In other words, the formal development does not define operations such as addition, multiplication, etc. that manipulate objects such as $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ as if they were multivariate polynomials.

4 Formally Verified Branch and Bound Algorithm

Using a proof assistant with a sophisticated proof-scripting language, such as PVS, and a formal development of Bernstein polynomials, such as the formalization presented in Section 3, it is possible to implement the algorithm in Fig. 1, Section 2.4, as a proof rule for verifying simply quantified polynomial inequalities.[3] This approach was initially taken by the authors, where the branch and bound procedure was written in the proof-scripting language provided by PVS [3]. The major advantage of this approach is that since proof-scripting languages preserve the logical consistency of theorem provers, tactics do not have to be proved correct. By construction, proofs built by tactics are correct. However, a tactic that implements the branch and bound procedure yields proofs that mimic the recursive structure of the method. In other words, if the branch and bound procedure requires n splits to prove a particular polynomial inequality, a proof that follows the recursive structure of the procedure will have n cases, one case per split. Since n may be large, this approach produces long proofs and it is very inefficient for practical use.

An alternative approach, based on computational reflection [19], is presented in this section. In this case, the algorithmic components of the branch and bound procedure are written as PVS functions using the PVS specification language. These functions act on the structures defined in Section 3 for representing multivariate polynomials. The correctness properties of the functions have been mechanically verified in PVS. The function `bernMinmax`, described in Section 4.1, estimates bounds of the minimum and maximum values of a polynomial on a partially open unit box, where the polynomial is represented in Bernstein form. The function

[3]In some procedural theorem provers, proof rules are called *tactics*. In PVS, proof rules are called *strategies*.

`polyMinmax`, described in Section 4.2, estimates bounds of the minimum and maximum values of polynomial on a partially open box, where the polynomial is represented in standard form. The function `poly_rel`, described in Section 4.3, solves a simply quantified polynomial inequality on a partially unbound box, where the polynomial is represented in standard form. These functions are used in PVS strategies, described in Section 5, that formally and automatically solve polynomial global optimization problems.

Notation

The functions described in this section use record types. Record type declarations take the form $\mathtt{T} \equiv \mathtt{a}_1 : \mathtt{T}_1 \;\times\; \cdots \times \mathtt{a}_n : \mathtt{T}_n$, where $\mathtt{a}_i$, with $1 \leq i \leq n$, is a field of $\mathtt{T}$. Field access is performed using the dot operator, i.e., if $\mathtt{r}$ is a record of type $\mathtt{T}$, $\mathtt{r.a}_i$, with $1 \leq i \leq n$, has the type $\mathtt{T}_i$ and representes the value of $\mathtt{r}$ in the field $\mathtt{a}_i$. As in the case of tuples, the `with` operator overrides record fields, i.e., given a field $\mathtt{a}_i$ of $\mathtt{r}$ and an expression e of type $\mathtt{T}_i$, $\mathtt{r}\ \mathtt{with}\,[\mathtt{a}_i := e]$ represents the record $\mathtt{s}$ that satisfies $\mathtt{s.a}_i = e$ and for all $1 \leq j \leq n$, with $j \neq i$, $\mathtt{s.a}_j = \mathtt{r.a}_j$.

4.1 Function `bernMinmax`

The core component of the branch and bound procedure in Section 2.4 is the recursive function `bernMinmax` presented in Fig. 2. It is used to calculate the information about the range of of a polynomial p in Bernstein form on the unit box. Rather than having the polynomial p, which is not a formally defined object, as an input, it has as basic parameters an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of degree at most $\boldsymbol{n}$, and m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$ that represent the partially open box ${}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$. The intent is that the pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is a Bernstein representation of p.

In contrast to the procedure in Fig. 1, the function `bernMinmax` always terminates. Termination is enforced by having a maximum recursion depth $\mathtt{D} \in \mathbb{N}$ and the current recursion depth $\mathtt{d} \in \mathbb{N}$, which satisfy the invariant $\mathtt{d} \leq \mathtt{D}$. Additional inputs includes a function `varsel`, which determines the variable to subdivide at each iteration and the direction to explore first, predicates `localex` and `globalex` on the output type, which cause the algorithm to exit locally and globally, respectively, and an accumulative parameter `omm` of the same type as the output value, which is used to prune some branches of the recursion. These inputs are described in Section 4.1.5. The inputs `D`, `varsel`, `localex`, `globalex`, and $\boldsymbol{q}$ never change during the recursion. To emphasize this fact, they are written as parameters of `bernMinmax` using PVS Curry notation.

The function `bernMinmax` returns a record of type

$$\begin{aligned}\mathtt{Outminmax} \;\equiv\; & \mathtt{lbmin}\colon \mathbb{R} \times \mathtt{lbmax}\colon \mathbb{R} \times \mathtt{lbvar}\colon \mathbb{R}^m \cup \{\bot\} \\ & \times \mathtt{ubmin}\colon \mathbb{R} \times \mathtt{ubmax}\colon \mathbb{R} \times \mathtt{ubvar}\colon \mathbb{R}^m \cup \{\bot\}.\end{aligned}$$

The fields `lbmin`, `lbmax`, `ubmin`, and `ubmax` are all real numbers. The fields `lbvar` and `ubvar` are either m-tuples or a special value $\bot$, which represents a null value. Elements of this type stores information about the range of an m-variate polynomial on a partially open box:

- `lbmin`: a minimum estimate for the lower bound,
- `lbmax`: a maximum estimate for the lower bound, if such a estimate is found.

```
bernMinmax(D, varsel, localex, globalex, $\boldsymbol{q}$)($\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}$, d, omm) : Outminmax $\equiv$
  let
    bmm = berncoeffsminmax($\boldsymbol{q}, \boldsymbol{\alpha}$)
  in
    if d = D $\vee$ localex(bmm) $\vee$ between?(omm, bmm) $\vee$ globalex(bmm) then
      bmm
    else
      let
        (left, $j$) = varsel($\boldsymbol{q}, \boldsymbol{\alpha}$, d),
        sl = subdivlmulti($\boldsymbol{\alpha}, j$),
        sr = subdivrmulti($\boldsymbol{\alpha}, j$),
        ($\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2$) = if left then (sl, sr) else (sr, sl) endif,
        ($\boldsymbol{l}_1, \boldsymbol{l}_2$) = if left then ($\boldsymbol{l}, \boldsymbol{l}$ with $[j := 0]$) else ($\boldsymbol{l}$ with $[j := 0], \boldsymbol{l}$) endif,
        ($\boldsymbol{u}_1, \boldsymbol{u}_2$) = if left then ($\boldsymbol{u}$ with $[j := 0], \boldsymbol{u}$) else ($\boldsymbol{u}$ with $[j := 0], \boldsymbol{u}$) endif,
        $\sigma$ = if left then $\lambda x.x/2$ else $\lambda x.(x+1)/2$ endif,
        omm$_1$ = combine(omm, bmm),
        bmm$_1$ = bernMinmax(D, varsel, localex, globalex, $\boldsymbol{q}$)($\boldsymbol{\alpha}_1, \boldsymbol{l}_1, \boldsymbol{u}_1$, d + 1, omm$_1$)
      in
        if globalex(bmm$_1$) then
          combine(update(bmm$_1$, $\sigma$, $j$), bmm)
        else
          let
            omm$_2$ = combine(omm$_1$, bmm$_1$),
            bmm$_2$ = bernMinmax(D, varsel, localex, globalex, $\boldsymbol{q}$)($\boldsymbol{\alpha}_2, \boldsymbol{l}_2, \boldsymbol{u}_2$, d + 1, omm$_2$),
            bmmleft = if left then bmm$_1$ else bmm$_2$ endif,
            bmmright = if left then bmm$_2$ else bmm$_1$ endif
          in
            combine(update(bmmleft, $\lambda x.x/2, j$),
                    update(bmmright, $\lambda x.(x+1)/2, j$))
        endif
    endif
```

Fig. 2 The function `bernMinmax`

- `lbvar`: a point where the polynomial attains the value `lbmax`, if such a point if found,
- `ubmin`: a minimum estimate for the upper bound, if such a estimate is found,
- `ubvar`: a point in the where the polynomial attains the value `ubmin`, if such a point is found,
- `ubmax`: a maximum estimate for the upper bound.

This null value is never returned when ${}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$ is a bounded box, i.e., when $\boldsymbol{l} = \boldsymbol{u} = \mathbf{0}^m$. However, for arbitrary m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$, it may be possible that no appropriate values for `lbmax`, `lbvar`, `ubmin`, and `ubvar` are found. The function `bernMinmax` keeps the invariant that `lbvar` $= \bot$ if and only if `ubvar` $= \bot$. Furthermore, if `lbvar` $=$ `ubvar` $= \bot$, the values of `lbmax` and `ubmin` are meaningless.

4.1.1 Lines 2–6: Base Case

Let $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ be an m-variate polynomial pair of degree at most $\boldsymbol{n}$, as defined in Section 3.3. The function `berncoeffsminmax` in Line 3 of Fig. 2 iterates the function `multicoeff` over all possible m-indices $\boldsymbol{i} \leq \boldsymbol{n}$ and computes an element of `Outminmax` whose fields satisfy the following properties.

$$\texttt{lbmin} = \min_{\boldsymbol{i} \leq \boldsymbol{n}} \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i}),$$

$$\texttt{ubmax} = \max_{\boldsymbol{i} \leq \boldsymbol{n}} \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i}).$$

The values of the rest of the fields depend on the set O of endindices of $\boldsymbol{n}$ defined as $\{\boldsymbol{i} \in \mathcal{I}_{\boldsymbol{n}} \mid \texttt{openindex}(\boldsymbol{l}, \boldsymbol{u})(\boldsymbol{i})\}$. It is noted that when $\boldsymbol{l} = \boldsymbol{u} = \mathbf{0}^m$, $O = \mathcal{I}_{\boldsymbol{n}} \neq \emptyset$. However, for arbitrary m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$, the set O may be empty. If O is empty, $\texttt{lbmax} = \texttt{ubmax} = 0$ and $\texttt{lbvar} = \texttt{ubvar} = \perp$. Otherwise, let $\boldsymbol{i}_{\min}$ and $\boldsymbol{i}_{\max}$ be m-indices in O where the minimum value $\min_{\boldsymbol{i} \in O} \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i})$ and the maximum value $\max_{\boldsymbol{i} \in O} \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i})$, respectively, are reached. In this case,

$$\texttt{lbmax} = \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i}_{\min}),$$

$$\texttt{lbvar} = \texttt{endpoint}(\mathbf{0}^m, \mathbf{1}^m)(\boldsymbol{i}_{\min}),$$

$$\texttt{ubmin} = \texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{i}_{\max}),$$

$$\texttt{ubvar} = \texttt{endpoint}(\mathbf{0}^m, \mathbf{1}^m)(\boldsymbol{i}_{\max}).$$

By Theorem 7, for all $\boldsymbol{x} \in \mathbb{U}^m$, $\texttt{lbmin} \leq \texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \leq \texttt{ubmax}$. Since $O \subseteq \mathcal{I}_{\boldsymbol{n}}$, the following properties follow from Theorem 8 when $\texttt{lbmax} \neq \perp$ (equivalently, $\texttt{ubmin} \neq \perp$),

$$\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\texttt{lbvar}) = \texttt{lbmax},$$

$$\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\texttt{ubvar}) = \texttt{ubmin}.$$

As noted in Section 3.5, for all $\boldsymbol{k} \in O$, the m-tuple $\texttt{endpoint}(\mathbf{0}^m, \mathbf{1}^m)(\boldsymbol{k})$ satisfies the predicate $\texttt{open}(\boldsymbol{l}, \boldsymbol{u}, \mathbf{0}^m, \mathbf{1}^m)$. Therefore, $\texttt{lbvar} \in {}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$ and $\texttt{ubvar} \in {}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$.

Let `bmm` be an element of type `Outminmax` defined as in Line 3 of Fig. 2. Then $\texttt{bmm.lbmin} \leq \texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \leq \texttt{bmm.lbmax}$ for all $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$. Furthermore, when both $\texttt{bmm.lbvar} \neq \perp$ and $\texttt{bmm.ubmin} \neq \perp$ hold, $\min_{\boldsymbol{x} \in_{\boldsymbol{l}} [\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}} \texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \in [\texttt{bmm.lbmin}, \texttt{bmm.lbmax}]$, and $\max_{\boldsymbol{x} \in_{\boldsymbol{l}} [\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}} \texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \in [\texttt{bmm.ubmin}, \texttt{bmm.ubmax}]$.

4.1.2 Lines 7–14: Subdivision

If the condition in Line 5 is `false`, the function `varsel` selects a natural number $j < m$, representing a variable to subdivide, and a Boolean value `left`, representing a direction ($\texttt{left} = \texttt{true}$ is left and $\texttt{left} = \texttt{false}$ is right). Thus, if the m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is a Bernstein representation of the polynomial p, then the functions `subdivlmulti` and `subdivrmulti`, defined in Section 3.3, are used to compute Bernstein representations of polynomials p_j^L and p_j^R as defined by Formula (9) in Section 2.3. By Theorem 6, $\langle \boldsymbol{q}, \boldsymbol{\alpha}_1 \rangle$ and $\langle \boldsymbol{q}, \boldsymbol{\alpha}_2 \rangle$ are m-variate polynomial pairs of degree at most $\boldsymbol{n}$ that are Bernstein representations of the polynomials p_1 and p_2, respectively, where

$$p_1(\boldsymbol{x}) = \begin{cases} p_j^L(\boldsymbol{x}) & \text{if } \texttt{left} = \texttt{true}, \\ p_j^R(\boldsymbol{x}) & \text{otherwise.} \end{cases} \qquad p_2(\boldsymbol{x}) = \begin{cases} p_j^R(\boldsymbol{x}) & \text{if } \texttt{left} = \texttt{true}, \\ p_j^L(\boldsymbol{x}) & \text{otherwise.} \end{cases}$$

The m-indices $\boldsymbol{l}_1, \boldsymbol{u}_1$ and $\boldsymbol{l}_2, \boldsymbol{u}_2$ represent, according to the value of `left`, the left or right partially open boxes that result from dividing the j-th interval of the box ${}_{\boldsymbol{l}}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}}$ in two halves.

4.1.3 Lines 15–32: Recursive Calls

The function `berncoeffsminmax` is recursively called with the parameters $\boldsymbol{\alpha}_1, \boldsymbol{l}_1, \boldsymbol{u}_1, \mathrm{d}+1$, and omm_1, which is an accumulative parameter explained in Section 4.1.5. It returns an element bmm_1 of `Outminmax` that represents range information of the polynomials p_1 on the partially open box ${}_{\boldsymbol{l}_1}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}_1}$. Since the m-tuples by bmm_1`.lbvar` and bmm_1`.ubvar` are computed for a unit box, the j-th element of those m-tuples must be translated back to the corresponding half intervals. This translation is accomplished by the function defined by

$$\begin{aligned} \texttt{update}(\mathrm{bmm}, \sigma, j) \;\equiv\; & \mathrm{bmm}\ \texttt{with}\,[\texttt{lbvar} := \texttt{lbvar}\ \texttt{with}\,[j := \sigma(\mathrm{bmm}.\texttt{lbvar}_j)], \\ & \texttt{ubvar} := \texttt{ubvar}\ \texttt{with}\,[j := \sigma(\mathrm{bmm}.\texttt{ubvar}_j)]], \end{aligned}$$

where bmm is an element of type `Outminmax`, σ is a function of type $\mathbb{R} \to \mathbb{R}$, and $j < m$ is a natural number.

If the condition in Line 19 is `true`, the function `update` is used with the actual parameters bmm_1 (defined in Line 17), σ (defined in Line 15), and j (defined in Line 9). Since the fields `lbmin` and `lbmax` in the element of type `Outminmax` returned by the function `update` are only correct for one of the half intervals, more conservative bound estimates should be computed for the whole interval. This is accomplished by the function `combine` on elements omm_1 and omm_2 of type `Outminmax` that returns an element of type `Outminmax` that satisfies

$$\texttt{lbmin} = \min(\mathrm{omm}_1.\texttt{lbmin}, \mathrm{omm}_2.\texttt{lbmin}),$$

$$\texttt{lbmax} = \begin{cases} \mathrm{omm}_1.\texttt{lbmax} & \text{if } \mathrm{omm}_2.\texttt{lbmax} = \bot, \\ \mathrm{omm}_2.\texttt{lbmax} & \text{if } \mathrm{omm}_1.\texttt{lbmax} = \bot, \\ \min(\mathrm{omm}_1.\texttt{lbmax}, \mathrm{omm}_2.\texttt{lbmax}) & \text{otherwise,} \end{cases}$$

$$\texttt{lbvar} = \begin{cases} \mathrm{omm}_1.\texttt{lbvar} & \text{if } \texttt{lbmax} = \mathrm{omm}_1.\texttt{lbvar}, \\ \mathrm{omm}_2.\texttt{lbvar} & \text{otherwise,} \end{cases}$$

$$\texttt{ubmin} = \begin{cases} \text{omm}_1.\texttt{ubmin} & \text{if } \text{omm}_2.\texttt{ubmin} = \bot, \\ \text{omm}_2.\texttt{ubmin} & \text{if } \text{omm}_1.\texttt{ubmin} = \bot, \\ \max(\text{omm}_1.\texttt{ubmin}, \text{omm}_2.\texttt{ubmin}) & \text{otherwise}, \end{cases}$$

$$\texttt{ubvar} = \begin{cases} \text{omm}_1.\texttt{ubvar} & \text{if } \texttt{ubmin} = \text{omm}_1.\texttt{ubmin}, \\ \text{omm}_2.\texttt{ubvar} & \text{otherwise}, \end{cases}$$

$$\texttt{ubmax} = \max(\text{omm}_1.\texttt{ubmax}, \text{omm}_2.\texttt{ubmax}),$$

The element of type `Outminmax` returned in Line 20 therefore has bound estimates for $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x})$, where $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\mathbf{1}^m, \mathbf{0}^m]_{\boldsymbol{u}}$.

If the condition in Line 19 is `false`, the function `berncoeffsminmax` is recursively called for a second time with the parameters $\boldsymbol{\alpha}_2, \boldsymbol{l}_2, \boldsymbol{u}_2, \texttt{d}+1$, and omm_2, which is an accumulative parameter explained in Section 4.1.5. The element bmm_2 represents range information of the polynomial p_2 on the box ${}_{\boldsymbol{l}_2}[\mathbf{0}^m, \mathbf{1}^m]_{\boldsymbol{u}_2}$. In Lines 28 and 29, the j-th component of the fields `lbvar` and `ubvar` of bmm_1 and bmm_2 are translated back to their corresponding half interval and then, the resulting elements of type `Outminmax` are combined into a new element of type `Outminmax` that has bound estimates for $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x})$, where $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\mathbf{1}^m, \mathbf{0}^m]_{\boldsymbol{u}}$.

4.1.4 Correctness

The correctness property of the function `bernMinmax` states that it computes correct bound estimates for $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x})$, where $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\mathbf{1}^m, \mathbf{0}^m]_{\boldsymbol{u}}$. Thus, if the m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is a Bernstein representation of a polynomial p, then the function `bernMinmax` computes range information for p on the unit box as well. 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. Lemma 1, Theorems 7, and 8 are used to prove the base case. The inductive case is discharged by Theorem 6.

Theorem 10 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of degree at most $\boldsymbol{n}$, m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$, $\texttt{D} \in \mathbb{N}$, $\texttt{d} \in \mathbb{N}$ with $\texttt{d} \leq \texttt{D}$, predicates `localex` and `globalex`, functions `varsel`, and elements omm and bmm of type `Outminmax` such that*

$$bmm = \texttt{bernMinmax}(\texttt{D}, \texttt{varsel}, \texttt{localex}, \texttt{globalex}, \boldsymbol{q})(\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \texttt{d}, omm),$$

the following properties hold

1. $bmm.\texttt{lbmin} \leq \texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \leq bmm.\text{ubmax}$, *for all* $\boldsymbol{x} \in \mathbb{U}^m$,
2. $bmm.\texttt{lbvar} \in {}_{\boldsymbol{l}}[\mathbf{1}^m, \mathbf{0}^m]_{\boldsymbol{u}}$ *and* $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(bmm.\texttt{lbvar}) = bmm.\texttt{lbmax}$, *if* $bmm.\texttt{lbvar} \neq \bot$, *and*
3. $bmm.\texttt{ubvar} \in {}_{\boldsymbol{l}}[\mathbf{1}^m, \mathbf{0}^m]_{\boldsymbol{u}}$ *and* $\texttt{evalmultibern}(\boldsymbol{q}, \boldsymbol{\alpha})(bmm.\texttt{ubvar}) = bmm.\texttt{ubmin}$, *if* $bmm.\texttt{ubvar} \neq \bot$.

It is noted that Theorem 10 holds for all possible values of the input parameters `varsel`, `localex`, `globalex`, and omm. These parameters are added for generality and efficiency reasons. They are explained in Section 4.1.5. Furthermore, since all functions in PVS are total, it is implicit in this theorem that the function `bernMinmax` always terminates.

4.1.5 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 function `varsel` takes as inputs an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, of degree at most $\boldsymbol{n}$, and a recursion depth. It returns a pair $(\texttt{left}, \texttt{var})$, where `left` is a Boolean value and $\texttt{var} < m$. 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 $\texttt{varsel}(\boldsymbol{q}, \boldsymbol{\alpha}, \texttt{d}) = (\texttt{true}, \texttt{mod}(m, \texttt{d}))$, which alternates the variables at each recursive call and always computes range information on the left interval first. The function `varsel` is an input to the algorithm in PVS, so it can facilitate any subdivision scheme.

One method for variable selection that has been implemented in PVS is called `MaxVarMinDir`. This function iterates through the variables, and for each $j < m$ computes the Bernstein coefficients $\texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \mathbf{0}^m)$ and $\texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \mathbf{0}^m\ \texttt{with}\,[j{:=}n_j])$, which by Theorem 8 in Section 3.4 are equal to the (Bernstein) evaluations of the polynomial at the points $\mathbf{0}^m$ and $(\mathbf{0}^m\ \texttt{with}\,[j{:=}1])$, respectively. These are the values at two different corner points of the unit box $\mathbb{U}^m$, where only the value of the j-th variable is different at the two points. The function `MaxVarMinDir` picks a variable for which these two function values have the greatest difference. The intention is that this is likely to choose the variable for which there is the most variation in the values of the polynomial on the box, when all of the other variables are fixed. The boolean value returned by the function `MaxVarMinDir`, along with this variable, depends on whether the algorithm is computing the maximum or the minimum of the polynomial. For instance, if the algorithm is computing the minimum, then the boolean value will be set to `true`, indicating subdividing left before right, precisely when the coefficient $\texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \mathbf{0}^m)$ is no greater than the coefficient $\texttt{multicoeff}(\boldsymbol{q}, \boldsymbol{\alpha}, \mathbf{0}^m\ \texttt{with}\,[j{:=}n_j])$. This is because it is more likely that the minimum value of the polynomial is attained on the left half of the interval. When computing the maximum, the boolean value would similarly be set to `false` in this example.

The function `MaxVarMinDir` represents a simple, intuitive method for choosing the variable for subdivision. The state of the art in variable and direction selection methods for subdividing Bernstein polynomials is more advanced than this method, however. As noted in [31] and [37], there are more efficient methods for choosing these variables and directions, including several based on derivatives. The reason that these methods have not been implemented in PVS is that they require that all the Bernstein coefficients of the polynomial are stored in memory and then analyzed by making many comparisons between them, and currently the PVS implementation does not store any of this information in memory. There are two issues that would arise from storing this information in memory. The first is that it would cause efficiency problems, and the second is that the algorithms would have to be redesigned, and the formal proofs would become more complicated. It is possible that these issues are not entirely prohibitive, so this topic will be analyzed in future work.

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,

$$\begin{aligned}\texttt{between?}(\text{omm}, \text{bmm}) \equiv\ & \text{omm}.\texttt{lbvar} \neq \bot \wedge \text{omm}.\texttt{lbmax} \leq \text{bmm}.\texttt{lbmin} \\ & \wedge \text{omm}.\texttt{ubvar} \neq \bot \wedge \text{bmm}.\texttt{ubmax} \leq \text{omm}.\texttt{ubmin}.\end{aligned}$$

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

The function bernMinmax is at the core of other algorithms that solve specific global optimization problems, e.g., finding bounds for 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. The predicate globalex will be used to force termination of the algorithm when a given condition is satisfied.

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

$$\begin{aligned}\texttt{eps\_localexit}(\epsilon)(\text{bmm}) \equiv\ & \text{bmm}.\texttt{lbvar} \neq \bot \wedge \text{bmm}.\texttt{lbmax} - \text{bmm}.\texttt{lbmin} \leq \epsilon \\ & \wedge \text{bmm}.\texttt{ubvar} \neq \bot \wedge \text{bmm}.\texttt{ubmax} - \text{bmm}.\texttt{ubmin} \leq \epsilon, \\ \texttt{eps\_globalexit}(\text{bmm}) \equiv\ & \texttt{false}.\end{aligned}$$

In this case, the parameters localex and globalex are instantiated with eps_localexit(ϵ) and eps_globalexit, respectively. Another useful instantiation of these parameters is presented in Section 4.3.

4.2 Function polyMinmax

The function polyMinmax in Fig. 3 can be used to compute range information for polynomial in standard form on an arbitrary partially open box $_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}\,]_{\boldsymbol{u}}$. However, as for the function bernMinmax, it does not directly have p, which is not a formally

```
01 : polyMinmax(D, varsel, localex, globalex, $\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$) : Outminmax ≡
02 :   let
03 :     $\boldsymbol{\alpha}_1$ = translatemulti($\boldsymbol{\alpha}, \boldsymbol{a}, \boldsymbol{b}$),
04 :     $\boldsymbol{\alpha}_2$ = tomultibern($\boldsymbol{\alpha}_1$),
05 :     omm = bernMinmax(D, varsel, localex, globalex, $\boldsymbol{q}$)($\boldsymbol{\alpha}_2, \boldsymbol{l}, \boldsymbol{u}$, 0, Emptymm)
06 :   in
07 :     omm with [lbvar := denormalize($\boldsymbol{a}, \boldsymbol{b}$)(omm.lbvar),
08 :               ubvar := denormalize($\boldsymbol{a}, \boldsymbol{b}$)(omm.ubvar)]
```

Fig. 3 The function polyMinmax

defined object, as an input. Rather, it has as parameters an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of degree at most $\boldsymbol{n}$, and m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$ that represent the partially open box ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$. The intent is that the pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ is a standard representation of p. The algorithm proceeds in four steps as follows.

Step 1 (Line 3): Use `translatemulti` to translate the standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of p on $[\boldsymbol{a}, \boldsymbol{b}]$ to a standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha}_1 \rangle$ of a polynomial p_1 on the unit box $\mathbb{U}^m$, such that p and p_1 attain the same values on their respective boxes.

Step 2 (Line 4): Use `tomultibern` to translate the standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha}_1 \rangle$ of p_1 to a Bernstein representation $\langle \boldsymbol{q}, \boldsymbol{\alpha}_2 \rangle$ of p_1. The constant element `Emptymm` of type `Outminmax` is defined such that all the numerical fields are 0 and the m-tuples are $\perp$.

Step 3 (Line 5): Apply `bernMinmax` to compute an element `omm` of `Outminmax` that gives range information for `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x})$ (equivalently $p_1(\boldsymbol{x})$), where $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{0}^m, \boldsymbol{1}^m]_{\boldsymbol{u}}$.

Step 4 (Lines 7–8): Translate the fields `lbvar` and `ubvar` of `omm` from ${}_{\boldsymbol{l}}[\boldsymbol{0}^m, \boldsymbol{1}^m]_{\boldsymbol{u}}$ back to ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$. This is accomplished by the function `denormalize`$(\boldsymbol{a}, \boldsymbol{b})$ that maps ${}_{\boldsymbol{l}}[\boldsymbol{0}^m, \boldsymbol{1}^m]_{\boldsymbol{u}}$ to ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ component-wise. It is defined such that

$$\texttt{denormalize}(\boldsymbol{a}, \boldsymbol{b})(\boldsymbol{x})_j \equiv a_j + x_j \cdot (b_j - a_j),$$

for $j < m$ and $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{0}^m, \boldsymbol{1}^m]_{\boldsymbol{u}}$.

The following correctness property of the function `polyMinmax` has been proved in PVS. The proof uses Theorems 4 and 5, and the correctness property of `bernMinmax` (Theorem 10).

Theorem 11 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of degree at most $\boldsymbol{n}$, m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$, m-boxes $[\boldsymbol{a}, \boldsymbol{b}]$, $D \in \mathbb{N}$, predicates `localex` and `globalex`, functions `varsel`, and elements `omm` of type `Outminmax` such that*

$$omm = \texttt{polyMinmax}(D, \texttt{varsel}, \texttt{localex}, \texttt{globalex}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}),$$

the following properties hold

1. *omm.`lbmin` $\leq$ `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \leq$ omm.`ubmax`, for all $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$,*
2. *omm.`lbvar` $\in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ and `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha})$(omm.`lbvar`) = omm.`lbmax` whenever omm.`lbvar` $\neq \perp$, and*
3. *omm.`ubvar` $\in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ and `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha})$(omm.`ubvar`) = omm.`ubmin` whenever omm.`ubvar` $\neq \perp$.*

4.3 Function `poly_rel`

The function `poly_rel`, defined in Fig. 4, uses the function `polyMinmax` to decide whether the polynomial p satisfies the inequality $p(\boldsymbol{x}) \ \Re\ 0$ for all $\boldsymbol{x}$ in a given partially unbounded box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$. As for the functions `bernMinmax` and `polyMinmax`, it does not have p as an input since it is not a formally defined object. Rather, it has an m-variate polynomial pair $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ as input, and the intent is that this pair is a standard representation of p.

```
01 : poly_rel(D, varsel, $\boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \Re$) : Outcome $\equiv$
02 :   let
03 :     $\boldsymbol{\alpha}_1$ = translatebound($\boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$),
04 :     omm = polyMinmax(D, varsel, localtrue($\Re$), counterex($\Re$), $\boldsymbol{q}, \boldsymbol{\alpha}_1, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$)
05 :   in
06 :     if localtrue($\Re$)(omm) then
07 :       IsTrue
08 :     elsif counterex($\Re$)(omm) then
09 :       if $0 \; \Re \; 1$ then
10 :         counterexbound($\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$)(omm.ubvar)
11 :       else
12 :         counterexbound($\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$)(omm.lbvar)
13 :       endif
14 :     else
15 :       Unknown
16 :     endif
```

Fig. 4 The function `poly_rel`

The function `poly_rel` has as inputs data structures representing a polynomial inequality on a partially unbounded box. It returns an element of the disjunct type

$$\texttt{Outcome} \equiv \mathbb{R}^m \cup \{\texttt{Unknown}, \texttt{IsTrue}\},$$

representing three possible kinds of outcomes of the function. The algorithm proceeds in 3 steps as follows.

Step 1 (Line 3): Use `translatemulti` to translate the standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of p on the partially unbounded box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$ to a standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha}_1 \rangle$ of a polynomial p_1 on the partially open box ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, such that if the inequality is satisfied for p_1, it is also satisfied for p.

Step 2 (Line 4): Apply `polyMinmax` to compute an element omm of `Outminmax` that gives range information for p_1 on ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$. The parameters `localex` and `globalex` of `polyMinmax` are instantiated with `localtrue`($\Re$) and `counterex`($\Re$), respectively. These predicates, which are parametric on the real order relation $\Re$, are defined as follows.

$$\begin{aligned}
\texttt{localtrue}(\Re)(\text{bmm}) &\equiv \texttt{if}\ 0\ \Re\ 1\ \texttt{then}\ \text{bmm.}\texttt{ubmax}\ \Re\ 0 \\
&\ \texttt{else}\ \text{bmm.}\texttt{lbmin}\ \Re\ 0\ \texttt{endif}, \\
\texttt{counterex}(\Re)(\text{bmm}) &\equiv \texttt{if}\ 0\ \Re\ 1\ \texttt{then}\ \text{bmm.}\texttt{ubvar} \neq \bot \wedge \text{bmm.}\texttt{ubmin}\ \neg\Re\ 0 \\
&\ \texttt{else}\ \text{bmm.}\texttt{lbvar} \neq \bot \wedge \text{bmm.}\texttt{lbmax}\ \neg\Re\ 0\ \texttt{endif},
\end{aligned}$$

where bmm is an element of type `Outminmax`.

Step 3 (Lines 6–16): Use the information in omm to determine wether the inequality `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha}_1)(\boldsymbol{x})\ \Re\ 0$ (equivalently $p_1(\boldsymbol{x})\ \Re\ 0$) holds for all $\boldsymbol{x} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$, in which case it returns the value `IsTrue`, or there exists $\boldsymbol{c} \in {}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$ such that `evalmulti`$(\boldsymbol{q}, \boldsymbol{\alpha}_1)(\boldsymbol{c})\ \neg\Re\ 0$ (equivalently $p_1(\boldsymbol{c})\ \neg\Re\ 0$), in which case it returns `counterexbound`$(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(\boldsymbol{c})$. If no determination can be made from omm, the algorithm returns the value `Unknown`.

The functions `localtrue`($\Re$) and `counterex`($\Re$) are passed as the parameters `localex` and `globalex`, respectively, to the recursive function `bernMinmax`. Thus, once it is found in a recursive step that the polynomial inequality holds on a subbox, i.e., `localtrue`($\Re$)(bmm) returns `true` for some bmm, the recursion will continue on the next branch. On the other hand, if `counterex`($\Re$)(bmm) returns `true`, the function `bernMinmax` will exit globally since there is a point where the inequality does not hold.

The following correctness property of `poly_rel` has been proved in PVS. The proof uses Theorem 9 and the correctness property of `polyMinmax` (Theorem 11).

Theorem 12 *For all m-variate polynomial pairs $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of degree at most $\boldsymbol{n}$, m-indices $\boldsymbol{l}$ and $\boldsymbol{u}$, with $l_j + u_j \leq 3$ for all $j < m$, m-boxes $[\boldsymbol{a}, \boldsymbol{b}]$, $D \in \mathbb{N}$, real order relations $\Re \in \{<, \leq, >, \geq\}$, and functions* `varsel`*,*

1. $\texttt{poly\_rel}(D, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \Re) = \texttt{IsTrue}$ *implies*

$$\forall \boldsymbol{x} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}} : \texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) \; \Re \; 0.$$

2. $\texttt{poly\_rel}(D, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \Re) = \boldsymbol{c}$*, with* $\boldsymbol{c} \in \mathbb{R}^m$*, implies*

$$\boldsymbol{c} \in {}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}} \wedge \texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{c}) \; \neg\Re \; 0.$$

5 Strategies

The formal development presented in this paper includes strategies that solve polynomial global optimization problems in PVS. These strategies apply the functions `minmax` and `bernstein`, described in Section 4, and their correctness properties to construct proofs using a computational reflection approach. The strategies not only yield proofs of constant length, but are effective in practical non-trivial problems.

The computational reflection approach used in this paper, which is not exclusive to PVS, can be illustrated as follows. Assume that a proof assistant user wants to prove

$$\forall \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] : p(\boldsymbol{x}) > -\frac{17434}{10000}, \tag{29}$$

where $\boldsymbol{a}$ is the 8-tuple $(-\frac{1}{10}, \frac{4}{10}, -\frac{7}{10}, -\frac{7}{10}, \frac{1}{10}, -\frac{1}{10}, -\frac{3}{10}, -\frac{11}{10})$, $\boldsymbol{b}$ is the 8-tuple $(\frac{4}{10}, 1, -\frac{4}{10}, \frac{4}{10}, \frac{2}{10}, \frac{2}{10}, \frac{11}{10}, -\frac{3}{10})$ and $p(\boldsymbol{x})$ is the 8-variate polynomial in Formula (1), written as a real number expression involving numerical rational constants, variables in $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$, and the operations addition, subtraction, multiplication, and exponentiation where the exponent is a numerical natural constant. Instead of a direct proof of that formula, the approach in Fig. 5 is used.

In a theorem prover with a sophisticated strategy language, such as PVS, all the steps in Fig. 5 can be mechanized. Since the polynomial $p(\boldsymbol{x})$ is written as a real number expression in the specification language, Step 1 requires introspective capabilities in the strategy language, i.e., the ability to observe expressions in the specification language as data in the strategy language. Once a representation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of p is found, Step 2 can be accomplished by unfolding the definitions in `eval`$(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x})$. Since the expression in Formula (30) is ground, Step 3 can be efficiently executed using a ground evaluator in a theorem prover that supports this feature. Otherwise, this step

1. Find a standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ of the polynomial $p(\boldsymbol{x}) + \frac{17434}{10000}$.
2. Prove that the proposition $\texttt{eval}(\boldsymbol{q}, \boldsymbol{\alpha})(\boldsymbol{x}) = p(\boldsymbol{x}) + \frac{17434}{10000}$ holds for all $\boldsymbol{x}$ in $[\boldsymbol{a}, \boldsymbol{b}]$.
3. Given a concrete natural number `D`, e.g., 100, and a concrete function `varsel`, e.g., `MaxVarMinDir`, check that the ground expression

$$\texttt{poly\_rel}(\texttt{D}, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{0}^m, \boldsymbol{0}^m, \boldsymbol{a}, \boldsymbol{b}, >) \tag{30}$$

   evaluates to `IsTrue`.
4. Use the correctness theorem of `poly_rel` to deduce Formula (29).

Fig. 5 Computational reflection approach

can be accomplished by unfolding all definitions in the expression. Step 4 is a simple application of Theorem 12.

The PVS strategy `bernstein`, described in Section 5.1, implements the approach described in Fig. 5 to solve simply quantified polynomial inequalities on partially unbound boxes. The PVS strategy `minmax`, described in Section 5.2, uses a similar computational reflection approach, through the function `polyMinmax`, to find bound estimates to a given precision of the minimum and maximum values of polynomials on partially open boxes.

5.1 Strategy `bernstein`

The strategy `bernstein` implements the approach in Fig. 5. It automatically discharges PVS sequents having one of the following forms

1. $\vdash \forall x_1, \ldots, x_m \colon X_1 \wedge \ldots \wedge X_m \implies p(x_1, \ldots, x_m) \,\Re\, r$,
2. $X_1, \ldots, X_m \vdash p(x_1, \ldots, x_m) \,\Re\, r$,
3. $\vdash \exists x_1, \ldots, x_m \colon X_1 \wedge \ldots \wedge X_m \wedge p(x_1, \ldots, x_m) \,\Re\, r$,

where

- $x_1, \ldots, x_m$ is a collection of variables of type `real`,
- for $1 \leq j \leq m$, X_j denotes a Boolean expression of one of the forms $a_j \prec_{l_j} x_j$, $x_j \prec_{u_j} b_j$, $a_j \prec_{l_j} x_j \prec_{u_j} b_j$, or $|x_j| \prec_{u_j} b_j$, a_j and b_j are numerical rational constants, and $\prec_{l_j}$, $\prec_{u_j}$, are real orders in $\{<, \leq\}$.
- $p(x_1, \ldots, x_m)$ denotes a real expression involving numerical rational constants, variables in $x_1, \ldots, x_m$, and the operations addition, subtraction, multiplication, and exponentiation where the exponent is a numerical natural constant.
- $\Re$ is a real order in $\{<, \leq, >, \geq\}$, and
- r is a numerical rational constant.

Sequents of the form 1 and 2 are called *universal* and sequents of the form 3 are called *existential*. A sequent of the form 1 can be reduced to the form 2 by skolemizing the quantified variables.

The strategy `bernstein` does not require any parameters, but optional strategy parameters allow for the specification of a maximum depth `D` (the default is $\texttt{D} = 100$) and variable selection method `varsel` (the default is $\texttt{varsel} = \texttt{MaxVarMinDir}$). First, the strategy builds from the Boolean expressions denoted by X_j, with

$1 \leq j \leq m$, PVS expressions representing a partially unbounded interval ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$, i.e., m-indices $\boldsymbol{l}, \boldsymbol{u}$ and m-tuples $\boldsymbol{a}, \boldsymbol{b}$ defined as follows.

- If X_j has the form $a'_j \prec_{l'_j} x_j$, then $a_j = a'_j$, $b_j = a'_j + 1$, $l_j = l'_j$, and $u_j = 2$.
- If X_j has the form $x_j \prec_{u'_j} b'_j$, then $a_j = b'_j - 1$, $b_j = b'_j$, $l_j = 2$, and $u_j = u'_j$.
- If X_j has the form $a'_j \prec_{l'_j} x_j \prec_{u'_j} b'_j$, then $a_j = a'_j$, $b_j = b'_j$, $l_j = l'_j$, and $u_j = u'_j$.
- If X_j has the form $|x_j| \prec_{u'_j} b'_j$, then $a_j = -b'_j$, $b_j = b'_j$, and $l_j = u_j = u'_j$.

Then, the strategy extracts a standard representation $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$ from the real expression denoted by $p(x_1, \ldots, x_m) - r$. This part is the most complex function in the strategy, it is 250 lines of strategy code and represents one third of the whole strategy development. It is a parser of PVS real expressions that builds an m-tuple $\boldsymbol{q}$ and a list $\boldsymbol{\alpha}$. The function, which was developed by B. Di Vito (NASA), does not assume any particular polynomial normal form. In particular, real expressions such as `(x-y)^2`, `(x-y)*(x-y)`, and `x*x-2*x*y+y*y` are all parsed into a pair of PVS expressions $\boldsymbol{q}$ and $\boldsymbol{\alpha}$ representing the 2-variable polynomial $x^2 - 2xy + y^2$.

It is noted that the correctness of the strategy is not compromised by the constructions of $\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{q}$, and $\boldsymbol{\alpha}$. Indeed, for the strategy to succeed in the case of a universal sequent, the goals

$$X_1, \ldots, X_m \vdash \texttt{unbounded}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(x_1, \ldots, x_m), \tag{31}$$

and

$$\vdash \texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(x_1, \ldots, x_m) = p(x_1, \ldots, x_m) - r, \tag{32}$$

are discharged by unfolding the definition of `unbounded` and `evalmulti`, respectively. Furthermore, the strategy uses the PVS ground evaluator to check whether the goal

$$\vdash \texttt{poly\_rel}(\texttt{D}, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \Re) = \texttt{IsTrue}$$

holds or not. If this is the case, Theorem 12 is applied and the proof of the sequent succeeds. If the expression $\texttt{poly\_rel}(\texttt{D}, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \Re)$ evaluates to an m-tuple $\boldsymbol{c}$, the strategy fails with a message informing that the counterexample $\boldsymbol{c}$ has been found.

In case of an existential sequent, the strategy evaluates the ground expression $\texttt{poly\_rel}(\texttt{D}, \texttt{varsel}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}, \neg\Re)$. If the expresion evaluates to `IsTrue`, the strategy fails with a message informing that the polynomial inequality does not hold for any point in the partially unbounded box ${}_{\boldsymbol{l}}\{\boldsymbol{a}, \boldsymbol{b}\}_{\boldsymbol{u}}$. If the expression evaluates to an m-tuple $\boldsymbol{c}$, the strategy instantiates the existential quantifier with the m-tuple $\boldsymbol{c}$ and discharges the goal

$$\vdash X_1 \wedge \ldots \wedge X_m \wedge p(\boldsymbol{c}) \, \Re \, r, \tag{33}$$

using the ground evaluator. In the case of an existential sequent, neither Formula (31) nor Formula (32) need to be discharged.

In all cases, if the ground evaluation of `poly_rel` returns `Unknown`, the strategy fails with a message informing that it was not possible to make a determination using the given maximum depth.

5.2 Strategy `minmax`

The strategy `minmax` can be applied to sequents having the form

$$\Gamma, X_1, \ldots, X_m \vdash \Delta,$$

where

- Γ and Δ denote arbitrary sets of Boolean expressions involving some variables $x_1, \ldots, x_m$ of type `real`,
- for $1 \leq j \leq m$, X_j denotes a Boolean expression of one of the forms $a_j \prec_{l_j} x_j \prec_{u_j} b_j$, or $|x_j| \prec_{u_j} b_j$, a_j and b_j are numerical rational constants, and $\prec_{l_j}, \prec_{u_j}$ are real orders in $\{<, \leq\}$.

The strategy has as parameter a real expression $p(x_1, \ldots, x_m)$, which is given either as a string or as sequent reference location [13] to an expression in Γ or Δ. As The expression $p(x_1, \ldots, x_m)$ must only involve numerical rational constants, variables in $x_1, \ldots, x_m$, and the operations addition, subtraction, multiplication, and exponentiation where the exponent is a numerical natural constant. The strategy computes bound estimates for the minimum and maximum values of $p(x_1, \ldots, x_m)$ within a given precision $\epsilon > 0$ (default value is), for the variable ranges given by $X_1 \wedge \ldots \wedge X_m$. Optional parameters of the strategy set the precision ϵ, maximum recursion depth `D`, and variable selection method `varsel` to values different from the defaults $\frac{1}{100}$, 100, and `MaxVarMinDir`, respectively.

As in the case of the strategy `bernstein`, the strategy `minmax` builds m-indices $\boldsymbol{l}, \boldsymbol{u}$ and m-tuples $\boldsymbol{a}, \boldsymbol{b}$ from the Boolean expressions denoted by X_i, for $1 \leq i \leq m$. In this case, since the forms of these expressions is more restricted than for the strategy `bernstein`, the objects $\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}$, represent the partially open box ${}_{\boldsymbol{l}}[\boldsymbol{a}, \boldsymbol{b}]_{\boldsymbol{u}}$. The goal

$$\vdash \texttt{open}(\boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b})(x_1, \ldots, x_m) \iff X_1 \wedge \ldots \wedge X_m,$$

is discharged by unfolding the definition of `open`. Furthermore, an m-tuple $\boldsymbol{q}$ and a list $\boldsymbol{\alpha}$ are built from $p(x_1, \ldots, x_m)$. The goal

$$\vdash \texttt{evalmulti}(\boldsymbol{q}, \boldsymbol{\alpha})(x_1, \ldots, x_m) = p(x_1, \ldots, x_m).$$

is discharged by unfolding the definition of `evalmulti`. Then, the strategy evaluates the expression

$$\texttt{polyMinmax}(\texttt{D}, \texttt{varsel}, \texttt{eps\_localexit}(\epsilon), \texttt{eps\_globalexit}, \boldsymbol{q}, \boldsymbol{\alpha}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{a}, \boldsymbol{b}).$$

The result of this evaluation is a PVS expression `omm` denoting a record of type `Outminmax`. The strategy adds the following formulas to Γ:

1. $\texttt{omm.lbmin} \leq p(x_1, \ldots, x_m)$,
2. $p(x_1, \ldots, x_m) \leq \texttt{omm.ubmax}$,
3. $p(\texttt{omm.lbvar}) = \texttt{omm.lbmax}$, if $\texttt{omm.lbvar} \neq \bot$, and
4. $p(\texttt{omm.ubvar}) = \texttt{omm.ubmin}$, if $\texttt{omm.ubvar} \neq \bot$.

These additional formulas are discharged by the application of Theorem 11. It is noted that the strategy does not always guarantee $\texttt{omm.lbmax} - \texttt{omm.lbmin} \leq \epsilon$ and $\texttt{omm.ubmax} - \texttt{omm.ubmin} \leq \epsilon$, since it is possible that the recursive function `bernMinmax` reaches the maximum depth before that precision is achieved.

5.3 Examples

The rest of this section presents several examples of global optimization theorems that can be automatically discharged with the strategy `poly_rel`. These examples are taken from [37] and were originally drawn from [40], 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:**

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

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

– **3-Variable Reaction Diffusion:**

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

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

– **Caprasse's System:**

$$\begin{aligned}\texttt{caprasse}(x_1, x_2, x_3, x_4) = & -x_1x_3^3 + 4x_2x_3^2x_4 + 4x_1x_3x_4^2 + 2x_2x_4^3 + 4x_1x_3 \\ & + 4x_3^2 - 10x_2x_4 - 10x_4^2 + 2,\end{aligned}$$

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

– **Adaptive Lotka-Volterra System:**

$$\texttt{lv}(x_1, x_2, x_3, x_4) = x_1x_2^2 + x_1x_3^2 + x_1x_4^2 - 1.1x_1 + 1,$$

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

– **Butcher's Problem:**

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

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:**

$$\begin{aligned}\texttt{magnetism}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = & x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 \\ & + 2x_7^2 - x_1,\end{aligned}$$

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

– **Heart Dipole:**

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

Table 1 Constants k_1 and k_2 for global optimization problems

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

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`: $\forall \boldsymbol{x} \in \mathbb{R}^m : \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] \implies p(\boldsymbol{x}) \geq k_1$.
- **Theorem** `p_exists`: $\exists \boldsymbol{x} \in \mathbb{R}^m : \boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}] \wedge p(\boldsymbol{x}) \leq k_2$.

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 $[\boldsymbol{a}, \boldsymbol{b}]$, 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 Formula (32). 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 discharging Formula (32). As noted before, existential sequents do not require that formula to be discharged. Therefore, this is not an issue for existential theorems.

Formula (32) involves variables $x_1 \ldots x_n$. Hence, it can not be checked using a ground evaluator. Since PVS does not feature an efficient symbolic evaluator, it is discharged by fully unfolding the definition of `evalmulti`. This approach requires many symbolic manipulations and, for some of these theorems, it is the bottleneck in proof speed.

Table 2 Proof times (sec) for global optimization problems

Problem	`p_forall`		`p_exists`
	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

6 Related Work

Simply quantified multivariate polynomial inequalities belong to the category of *non-linear arithmetic* problems, i.e., polynomial arithmetic problems that are not restricted to the linear case. Tarski proved that the first-order theory of real numbers with addition, subtraction, multiplication, and less-than allows quantifier elimination [39]. Hence, non-linear arithmetic is decidable assuming that the truth value of expressions involving only constants can be computed. Tarski's quantifier elimination procedure is not elementary recursive, which makes it impractical for an actual implementation. A procedure with double exponential complexity called *Cylindrical Algebraic Decomposition* (CAD) was proposed by Collins [9]. Sophisticated implementations of the CAD procedure are available in the Redlog system[4] and in the QEPCAD library.[5]

In the context of interactive theorem proving, McLaughlin and Harrison present a proof-producing implementation in HOL Light of a quantifier elimination procedure due to Hörmander [26]. As Tarski's original method, Hörmander's procedure cannot be bounded by a tower of exponential functions. A formalization in Coq of a quantifier elimination procedure that is closer to CAD but still not elementary recursive is presented in [8]. Cohen and Mahboubi hope that the formalization in [8] will lead to the verification of a CAD algorithm specified, but not completely verified, in Coq [25]. MetiTarski [1] and RAHD (Real Algebra in High Dimensions) [36] are specialized theorem provers for the theory of real closed fields. MetiTarski is designed to prove universally quantified inequalities involving real-valued functions such as trascendental functions. RAHD combines several decision methods for the existential theory of real closed fields. Both systems use a CAD procedure for quantifier elimination among many other proof strategies.

Table 3 reports run times in seconds of different quantifier elimination tools on the problems listed in Section 5.3. These tools are not all installed on the same machine, but all machines have a similar configuration. For this reason, these times should be used as relative indicators rather than as absolute times. A blank entry for a given problem and tool means that the problem was not solved by the tool in 5 minutes. The columns $\text{Redlog}_{\texttt{rlqe}}$ and $\text{Redlog}_{\texttt{rlcad}}$ refer to two different quantifier elimination methods implemented in Redlog (Free CSL version), 10-Mar-11. The first method is a specialized method for polynomials where each quantified variable has at most degree 2. However, several heuristics are used to handle the case of polynomials with higher degrees. The second method is CAD. The next column corresponds to QEPCAD Version B 1.54, 15 Apr 2010. The last column refers to MetiTarski 1.8 (built 18 Feb 2011), which was only tried on universally quantified problems. It is noted that the problem Butcher causes QEPCAD to abort with the message "Failure occurred in: GCSI (final check) Reason for the failure: Too few cells reclaimed." Since MetiTarski uses QEPCAD, this error is reflected in MetiTarski, which also signals an error for this problem. The column Kodiak refers to the run time of an

[4]http://redlog.dolzmann.de

[5]http://www.usna.edu/cs/~qepcad/B/QEPCAD.html

Table 3 Times (sec) of quantifier elimination procedures on global optimization problems

Problem	Kodiak	Redlog$_{\texttt{rlqe}}$	Redlog$_{\texttt{rlcad}}$	QEPCAD	Metit
Schwefel (∀)	0.94	0.49		0.84	0.11
Schwefel (∃)	0.28	138.9		0.91	(n/a)
Reaction diffusion (∀)	0.0	0.34	0.37	0.01	0.09
Reaction diffusion (∃)	0.0	0.34	0.35	0.01	(n/a)
Caprasse (∀)	0.29	1.75		6.54	0.16
Caprasse (∃)	0.31	15.06		6.88	(n/a)
Lotka-Volterra (∀)	0.1	0.36	0.45	0.01	0.1
Lotka-Volterra (∃)	0.0	0.35	0.4	0.01	(n/a)
Butcher (∀)	0.2	0.42		(abort)	(abort)
Butcher (∃)	0.2	0.36		(abort)	(n/a)
Magnetism (∀)	73.54	0.67	0.36	0.18	0.54
Magnetism (∃)	0.32	0.42	0.36	0.35	(n/a)
Heart dipole (∀)	7.36				
Heart dipole (∃)	3.7				(n/a)

implementation in C, using the GNU library for arbitrary precision GMP,[6] of the verified algorithms presented in this paper.

The most recent version of PVS (5.0) has a strategy `rahd`, which is an early implementation of the RAHD method. In contrast to `bernstein`, the strategy `rahd` is not implemented in the logic of PVS. In particular, it is not supported by a proof tree of basic PVS proof rules. The strategy is supposed to use the library QEPCAD. However, according to the PVS developers, this feature is currently disabled. The strategy `rahd` proves the universal theorem of the Reaction Diffusion problem in 0.34 second. In all the other problems, it either terminates without proving the theorems or does not terminate within 5 minutes. The most recent standalone version of RAHD was not tried.[7] The authors also tried the quantifier elimination procedure implemented in HOL Light, but since it did not return within 5 minutes in any of the problems, it is not reported in the table.

From this limited benchmark, it seems that for these kinds of problems the Bernstein method competes well against state-of-the-art quantifier elimination procedures. In particular, none of these procedures was able to discharge the Heart Dipole theorems.This is not surprising as it is generally accepted that quantifier elimination is only practical for a small number of variables. For example, Passmore remarks in his thesis that he has “never succeeded in using pure P-CAD on a nonlinear problem in more than 10 variables” and that, in particular, “standard P-CAD implementations such as QEPCAD run out of resources on relatively small problems in 5 or 6 variables” [35]. However, the comparison of the PVS strategy `bernstein` to these quantifier elimination tools is not completely fair. On one hand, some of these tools handle a richer set of formulas, e.g., mixed quantification, Boolean operators, non-polynomial inequalities, etc. On the other hand, `bernstein` is a proof-producing strategy entirely implemented in the logic of PVS, while those tools are implemented using efficient programming languages.

[6]http://gmplib.org

[7]http://code.google.com/p/rahd

Another approach to solving non-linear arithmetic problems of the form

$$\forall \boldsymbol{x} \in \mathbb{R}^m : \; p(\boldsymbol{x}) \geq 0 \tag{34}$$

consists of finding $q_0, \ldots, q_n$ polynomials such that $p(\boldsymbol{x}) = \sum_{i=0}^{n} q_i(\boldsymbol{x})^2$. Finding this polynomial decomposition is sufficient to prove Formula (34), since for $i \leq n$, $q_i(\boldsymbol{x})^2 \geq 0$. Such problems are referred to as *Sum of Squares* problems and have been studied for more than a century.

Parrillo proposes a method to find such a sum of squares decomposition of a polynomial using Linear Matrix Inequalities (LMI's), which are solved using semidefinite programming [34]. Harrison implemented this method in HOL Light [20] in a procedure that comes with the standard distribution of HOL Light. An advantage of this method with respect to CAD is that the algorithm that computes the decomposition on sum of squares can be used in a proof-producing procedure without being trusted, i.e., it does not need to be verified. The fact that the decomposition is correct is checked by the theorem prover. However, the software that solves the LMI problem does use floating point computations and can be susceptible to related errors. Thus, the coefficients in the polynomials q_i may not be exact. Harrison proposes a heuristic method to adjust the original decomposition to rational coefficients that works in several cases. The authors are also aware of recent developments on SOS methods that work on polynomials with rational coefficients [22, 29].

The authors have tried the `REAL_SOS` procedure available in HOL Light on the universal theorems listed in Section 5.3. That procedure instantly solves the universal case of the problem Magnetism. The universal case of the problem Schwefel is solved, but only when no bounds are given. In all the other cases, the procedure does not terminate within 5 min. It has been reported that the Coq theorem prover also implements this procedure.[8] The authors did not try it, but expect similar results as the documentation states that Coq's SOS tactic is based on HOL Light's procedure.

Numerical approximation methods have been tried before in the context of interactive theorem proving. A PVS proof-producing strategy called `numerical` is presented in [11]. This strategy, which is available as part of the PVS NASA Libraries, solves universally quantified formulas involving variables in a bounded box and real-valued functions. The strategy uses a branch-and-bound method called *interval splitting*, but relies on interval arithmetic rather than on Bernstein polynomials. The authors tried `numerical`, with default parameters, on the universal case of the problems listed in Section 5.3, and it succeeds to discharge Schwefel in 2.79 s; it fails in all the other problems. A similar tactic called `interval` is available in Coq [27]. The Coq tactic is more efficient than the PVS strategy as it uses a formalized floating-point arithmetic rather than rational arithmetic. In any case, the well-known variable dependency problem of interval arithmetic, due to the fact that interval arithmetic is only semi-distributive, make interval arithmetic, without additional optimization techniques, impractical for solving high-dimension multivariate polynomial inequalities.

A sophisticated implementation of interval arithmetic is provided by the tool RealPaver [18]. The input to RealPaver is a constraint satisfaction problem (CSP), i.e., set of constraints involving variables and real-valued functions. The output is a set of

[8]http://coq.inria.fr/refman/Reference-Manual026.html#@default870

boxes containing solutions to the constraints. This tool is particularly interesting as it uses floating-point numbers to correctly bound the interval computations. Therefore, the output of RealPaver is guaranteed to be correct, i.e., the union of the boxes returned by RealPaver precisely contain all the solutions to the CSP. In particular, if no boxes are returned, the problem has no solutions. RealPaver is a powerful tool, but it is a constraint solver rather than a theorem prover, and using the tool in a theorem prover would require the user to trust two things: the logical soundness of the theorem prover, and the validity of the answer produced by RealPaver. However, using an algorithm that is implemented and verified in a theorem prover directly to solve problems in that same theorem prover only requires the user to trust the logical soundness of the prover.

Solving a universally quantified polynomial inequality in RealPaver requires expressing the problem as a CSP with no solution and looking for an empty box as the result. By using this technique, the universal case of the Heart Dipole problem is solved by RealPaver in 0.280 s. Existential problems cannot be solved this way, since there is no way to express disjunctive constraints in the language and the fact that a box is returned does not mean that a solution is found on it. Moreover, RealPaver may use some heuristics that do not preserve completeness. This is indicated by RealPaver with a message stating that the process was not reliable and some solutions may be lost. In that case, the fact that no box is produced as output does not guarantee that the CSP does not have a solution.

The authors have modified the branch-and-bound algorithm in Section 4 such that given a list of polynomial inequalities, which can be interpreted in a conjunctive or disjunctive way, it returns 3 sets of boxes:

- Green boxes whose points are guaranteed to satisfy the constraints.
- Red boxes whose points are guaranteed not to satisfy the constraints.
- Yellow boxes whose points may or may not satisfy the constraints.

For the restricted set of polynomial constraints, this algorithm provides a functionality similar to that of RealPaver. However, since it is written in PVS and uses rational arithmetic instead of floating-point computations, it is not nearly as efficient as RealPaver. In contrast to RealPaver, it computes sets of boxes that under approximate (green boxes) **and** over approximate (union of red and yellow boxes) the region defined by the constraints. This algorithm is used in [10] to approximate, with a high level of accuracy, the failure and safe domains, i.e., red and green boxes, of a system modeled by polynomial inequalities.

Heuristic and incomplete methods have also been used in interactive theorem provers to handle non-linear arithmetic [21]. The PVS proof-producing strategy `field` [30], which is based on an homonymous tactic in Coq [12], works by first removing all divisions of the real number expressions in a sequent and then applying several heuristics such as case analysis, simplification of expressions, application of real order properties, etc. These kinds of strategies work on some practical problems, but they are not general enough to handle multivariate polynomials.

Single and multivariate Bernstein polynomials have been formalized in the Coq theorem prover [4, 43]. In [4], a formalization in Coq of de Casteljau's algorithm for computing Bernstein coefficients of univariate polynomial 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. It is not intended to be used in a reflective way as the algorithm for multivariate polynomials presented in this paper. The closest work to the one presented here, and the inspiration for this work, is the formalization in Coq of multivariate Bernstein polynomials given by Zumkeller in [43]. That work also includes strategies for solving global optimization problems based on a branch-and-bound algorithm. In his thesis, Zumkeller gives a short overview of the branch-and-bound method, but does not provide any technical details of its implementation. The thesis uses a recursive formalization of multivariate polynomials, but it is unknown to the authors if the same representation is used in the algorithm. The correctness of that algorithm is not formally proved in Coq. Unfortunately, that development seems to be abandoned and, according to the Coq developers, it is not supported in the current version of Coq. Hence, the authors were unable to compare the development presented here to that presented in [43].

7 Conclusion

This paper presented a set of formally verified algorithms for global optimization of multivariate polynomials. These algorithms, which are based on Bernstein polynomial techniques, are the building blocks of PVS strategies for automatically finding upper and lower polynomial bounds and solving simply quantified multivariate polynomial inequalities. For the limited set of problems presented here, the PVS strategies compare well to state-of-the-art implementations of quantifier elimination procedures, such as those available in RedLog and QEPCAD, and to other tools based on SOS, such as the the one available in HOL Light.

The proof of a formula that uses the PVS strategies presented here are guaranteed to be correct in the sense that they are supported by a tree of basic PVS proof rules that completely discharges the original statement. These proof rules do not make use of any trusted external oracle. A key step in this proof is the use of the correctness theorem of a branch-and-bound algorithm, which is fully proved in PVS. The evaluation of this algorithm on ground terms is relatively efficient, since PVS natively supports rational arithmetic. In other words, the fact that $\frac{1}{3} + \frac{1}{3} + \frac{1}{3}$ is equal to 1 does not require a proof. PVS automatically reduces the former expression into the latter one.

Despite the promising results, there is still room for improvement. An obvious one is to implement a better approach for discharging Formula (32), which states that a real number expression $p(x_1, \ldots, x_n)$ is correctly represented by $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$. It could be possible to request that the user enters the formulas using a particular datatype for polynomial expressions, e.g., $\mathtt{P}(x_1, \ldots, x_n)$. In this case, in the spirit of computational reflection, it would be possible to write in PVS a function that translates a representation such as $\mathtt{P}(x_1, \ldots, x_n)$ into $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$. Proving Formula (32) would only require the application of a correctness theorem for such a function. However, if the PVS strategies allow users to specify polynomials as real number expressions in no particular form, it is unavoidable to discharge a formula such as Formula (32), where one side of the equality is a real number expression involving universally quantified variables $x_1, \ldots, x_n$ and the other side is an evaluation function on the given representation and variables $x_1, \ldots, x_n$. In PVS, there is not a particularly efficient way to evaluate expressions containing variables. Expanding the definitions will definitively discharge the formula, assuming that the representation is correct.

However, if the number of sums and products involved in the evaluation function is large, as it is the case for `evalmulti`, this approach is not very efficient.

The authors are looking into a deeper embedding of multivariate polynomials, where the representation $\mathtt{P}(x_1,\ldots,x_n)$ reflects the syntactical structure of $p(x_1,\ldots,x_n)$. In order to do this, the polynomial representation needs to support polynomial operations such as addition, multiplication, and exponentiation. As stated in a previous section, such as representation is called a deep-embedding. In this case, Formula (32) can be discharged by recursively rewriting the component of the expressions. This approach is successfully applied by the interval arithmetic strategy `numerical` to discharge the inclusion relation between a real number expressions and a syntactically similar interval arithmetic expressions [11].

The algorithms in Section 4 have a function `varsel` as a parameter. This parameter is instantiated by the strategies in Section 5 with the function `MaxVarMinDir`. As noted in [31] and [37], there are efficient methods for choosing these variables that have not been implemented, including several based on derivatives. Furthermore, some of heuristics to prune the recursive search tree in the brand-and-bound algorithm can be also implemented through the `local_exit` and `global_exit` parameters of the function `bernMinmax`. Since the correctness proofs of these algorithms hold for all possible inputs of these functions, the correctness of the strategies is not affected by any particular instantiation of these parameters. The problem there is not the verification but the formalization of these methods and heuristics. Some of these techniques require complex dynamic data structures that may be difficult to formalize in the pure functional specification language provided by PVS. Furthermore, 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.

The strategies presented in this paper only handle simply quantified polynomial inequalities. The authors are currently working on an extension of the approach presented here to arbitrary Boolean formulas involving *conditional polynomials*, i.e, expressions of the form `if` $b(\boldsymbol{x})$ `then` $p(\boldsymbol{x})$ `else` $q(\boldsymbol{x})$ `endif`, where $b(\boldsymbol{x})$ is a Boolean expression and p and q are multivariate polynomials on $\boldsymbol{x}$. A generic branch and bound algorithm has been specified and verified, whose inputs and outputs are defined on abstract types. That generic algorithm has been instantiated to work with polynomial representations, e.g., $\langle \boldsymbol{q}, \boldsymbol{\alpha} \rangle$, but also with representations of conditional polynomials. The authors are currently investigating other possible instantiations of this verified branch-and-bound algorithm for paving, as in [10], and for interval analysis.

As mentioned before, interval arithmetic is another well-known technique for global optimization [28]. The interval splitting technique used in interval analysis is very similar to the subdivision method used in algorithms based on Bernstein polynomials. Interval arithmetic does not perform well on high-dimension polynomials, but it handles real-valued functions such as logarithm, exponential, square root, and trigonometric functions. A possible direction of research that combines both interval arithmetic and polynomial approximations of real-valued functions is Taylor models [32]. A Taylor model is an object consisting of an interval and a polynomial that together approximates a function in a given interval. Interval arithmetic and Taylor models are already available in PVS [7]. In order to extend the work presented in this paper to real-valued functions, the authors will follow [42] and explore the use of Taylor models to bound expressions involving real-valued functions, where the polynomial part of a Taylor Model is represented using Bernstein polynomials.

Finally, although PVS has built-in rational arithmetic, rational arithmetic is still expensive compared to floating-point arithmetic. In the context of Taylor models, an interesting idea is to formalize floating-point arithmetic in PVS and use it for representing either polynomials with floating-point coefficients, as in [5], interval arithmetic with floating-point bounds, as in [27], or both. In any case, this will require a major modification to the existing developments in PVS.

References

1. Akbarpour, B., Paulson, L.C.: MetiTarski: an automatic theorem prover for real-valued special functions. J. Autom. Reason. **44**(3), 175–205 (2010)
2. Alford, J.: Translation of Bernstein coefficients under an affine mapping of the unit interval. Technical Memorandum NASA/TM-2012-217557, NASA Langley Research Center (2012)
3. 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 (2003)
4. Bertot, Y., Guilhot, F., Mahboubi, A.: A formal study of Bernstein coefficients and polynomials. Tech. Rep. INRIA-005030117, INRIA (2010)
5. Brisebarre, N., Joldes, M., Martin-Dorel, É., Mayero, M., Muller, J.M., Pasca, I., Rideau, L., Théry, L.: Rigorous polynomial approximation using Taylor models in Coq. In: Goodloe, A., Person, S. (eds.) Proceedings of the NASA Formal Methods Symposium (NFM 2012). Lecture Notes in Computer Science, vol. 7226, pp. 85–99. Springer, Norfolk, US (2012)
6. de Casteljau, P.: Formes à pôles. Hermès (1985)
7. Cháves, F., Daumas, M.: A library of Taylor models for PVS automatic proof checker. Technical Report RR2006-07, École Normale Supérieure de Lyon (2006)
8. Cohen, C., Mahboubi, A.: Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. Logical Methods in Computer Science (LMCS) **8**(1:02), 1–40 (2012)
9. Collins, G.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Second GI Conference on Automata Theory and Formal Languages. Lecture Notes in Computer Science, vol. 33, pp. 134–183. Springer, Kaiserslautern (1975)
10. 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 (2011)
11. Daumas, M., Lester, D., Muñoz, C.: Verified real number calculations: a library for interval arithmetic. IEEE Trans. Comput. **58**(2), 1–12 (2009)
12. Delahaye, D., Mayero, M.: Field, une procédure de décision pour les nombres réels en coq. In: Castéran, P. (ed.) Journées Francophones des Langages Applicatifs (JFLA'01), pp. 33–48. Collection Didactique, INRIA, Pontarlier, France, Janvier (2001)
13. Di Vito, B.: Manip user's guide, version 1.3. Technical Memorandum NASA/TM-2002-211647, NASA Langley Research Center (2002)
14. de Dinechin, F., Lauter, C., Melquiond, G.: Certifying the floating-point implementation of an elementary function using Gappa. IEEE Trans. Comput. **60**(2), 242–253 (2011)
15. 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)
16. Garloff, J.: The Bernstein algorithm. Interval Comput. **4**, 154–168 (1993)
17. Garloff, J.: Application of Bernstein expansion to the solution of control problems. Reliab. Comput. **6**, 303–320 (2000)
18. Granvilliers, L., Benhamou, F.: RealPaver: an interval solver using constraint satisfaction techniques. ACM Trans. Math. Softw. **32**(1), 138–156 (2006)

19. Harrison, J.: Metatheory and reflection in theorem proving: a survey and critique. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK (1995)
20. Harrison, J.: Verifying nonlinear real formulas via sums of squares. In: Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 4732, pp. 102–118. Springer (2007)
21. Hunt, W.A., Jr., Krug, R.B., Moore, J.S.: Linear and nonlinear arithmetic in ACL2. In: Geist, D., Tronci, E. (eds.) Proceedings of Correct Hardware Design and Verification Methods (CHARME). Lecture Notes in Computer Science, vol. 2860, pp. 319–333. Springer, L'Aquila, Italy (2003)
22. Kaltofen, E.L., Li, B., Yang, Z., Zhi, L.: Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. In: Robbiano, L., Abbott, J. (eds.) Approximate Commutative Algebra. Texts and Monographs in Symbolic Computation, Springer Vienna (2010)
23. Kuchar, J., Yang, L.: A review of conflict detection and resolution modeling methods. IEEE Trans. Intell. Transp. Syst. **1**(4), 179–189 (2000)
24. Lorentz, G.G.: Bernstein Polynomials, 2nd edn. Chelsea Publishing Company, New York, N.Y. (1986)
25. Mahboubi, A.: Implementing the cylindrical algebraic decomposition within the Coq system. Math. Struct. Comput. Sci. **17**(1), 99–127 (2007)
26. 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)
27. Melquiond, G.: Proving bounds on real-valued functions with computations. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12–15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 2–17. Springer (2008). doi:10.1007/978-3-540-71070-7_2
28. Moa, B.: Interval methods for global optimization. Ph.D. thesis, University of Victoria (2007)
29. Monniaux, D., Corbineau, P.: On the generation of Positivstellensatz witnesses in degenerate cases. In: Proceedings of Interactive Theorem Proving (ITP). Lecture Notes in Computer Science (2011)
30. Muñoz, C., Mayero, M.: Real automation in the field. Tech. Rep. NASA/CR-2001-211271 Interim ICASE Report No. 39, ICASE-NASA Langley, ICASE Mail Stop 132C, NASA Langley Research Center, Hampton VA 23681-2199, USA (2001)
31. Nataraj, P.S.V., Arounassalame, M.: A new subdivision algorithm for the Bernstein polynomial approach to global optimization. International Journal of Automation and Computing (IJAC) **4**(4), 342–352 (2007)
32. Neumaier, A.: Taylor forms - use and limits. Reliab. Comput. **9**(1), 43–79 (2003)
33. Owre, S., Rushby, J., Shankar, N.: PVS: a prototype verification system. In: Kapur, D. (ed.) Proceeding of the 11th International Conference on Automated Deductioncade. Lecture Notes in Artificial Intelligence, vol. 607, pp. 748–752. Springer (1992)
34. Parrilo, P.A.: Semidefinite programming relaxations for semialgebraic problems. Math. Program. **96**, 293–320 (2003). doi:10.1007/s10107-003-0387-5
35. Passmore, G.O.: Combined decision procedures for nonlinear arithmetics, real and complex. Ph.D. thesis, The Univesity of Edinburgh (2011)
36. 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. LNAI, pp. 122–137, no. 5625. Springer-Verlag (2009)
37. Ray, S., Nataraj, P.S.: An efficient algorithm for range computation of polynomials using the Bernstein form. J. Glob. Optim. **45**, 403–426 (2009)
38. Smith, A.P.: Fast construction of constant bound functions for sparse polynomials. J. Glob. Optim. **43**, 445–458 (2009)
39. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. Univeristy of California Press (1951)
40. Verschelde, J.: The PHC pack, the database of polynomial systems. Tech. rep., Univeristy of Illinois, Mathematics Department, Chicago, IL (2001)
41. Zippel, R.: Effective Polynomial Computation. Kluwer Academic Publishers (1993)
42. Zumkeller, R.: Formal global optimisation with Taylor models. In: Furbach, U., Shankar, N. (eds.) Proceedings of the Third International Joint Conference on Automated Reasoning. Lecture Notes in Computer Science, vol. 4130, pp. 408–422 (2006)
43. Zumkeller, R.: Global Optimization in Type Theory. Ph.D. thesis, École Polytechnique Paris (2008)