NASA/TM–2014-218548

A Formally-Verified Decision Procedure for Univariate Polynomial Computation Based on Sturm's Theorem

Anthony J. Narkawicz and César A. Muñoz
Langley Research Center, Hampton, Virginia

November 2014

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question to help@sti.nasa.gov

- Fax your question to the NASA STI Information Desk at 443-757-5803

- Phone the NASA STI Information Desk at 443-757-5802

- Write to:
  STI Information Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM–2014-218548

A Formally-Verified Decision Procedure for Univariate Polynomial Computation Based on Sturm's Theorem

Anthony J. Narkawicz and César A. Muñoz
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

November 2014

Acknowledgments

Authors would like to thank Leonardo de Moura, Manuel Eberl, and anonymous referees for their comments on previous versions of this document.

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Abstract

Sturm's Theorem is a well-known result in real algebraic geometry that provides a function that computes the number of roots of a univariate polynomial in a semi-open interval. This paper presents a formalization of this theorem in the PVS theorem prover, as well as a decision procedure that checks whether a polynomial is always positive, nonnegative, nonzero, negative, or nonpositive on any input interval. The soundness and completeness of the decision procedure is proven in PVS. The procedure and its correctness properties enable the implementation of a PVS strategy for automatically proving existential and universal univariate polynomial inequalities. Since the decision procedure is formally verified in PVS, the soundness of the strategy depends solely on the internal logic of PVS rather than on an external oracle. The procedure itself uses a combination of Sturm's Theorem, an interval bisection procedure, and the fact that a polynomial with exactly one root in a bounded interval is always nonnegative on that interval if and only if it is nonnegative at both endpoints.

1 Introduction

Problems involving polynomial inequalities appear in applications such as air traffic conflict detection and resolution algorithms [16], floating point analysis [5], and uncertainty and reliability analysis of dynamic and control systems [3, 10]. Solving these problems in a rigorous way is fundamental to the logical correctness of these safety-critical systems. However, formal reasoning about polynomials and other nonlinear functions in an interactive theorem prover is challenging.

Fortunately, significant advances have been made in this area in recent years. In addition to related work described in later sections, the authors developed formalizations in PVS [24] of multivariate Bernstein polynomials [22] and a generic branch and bound algorithm [23], both of which yield well-known numerical approximation methods. These PVS developments include formally-verified semi-decision procedures for checking validity and satisfiability of nonlinear properties involving variables ranging over real intervals. The procedures are integrated into the PVS theorem prover as automated proof-producing strategies such as `bernstein`, which uses Bernstein polynomials, and `interval`, which uses interval arithmetic. To automatically discharge a formula such as

$$x^{120} - 2x^{60} + 1.001 > 0, \tag{1}$$

whenever $x \in [0, 3]$, the user simply needs to invoke one of those strategies in PVS. In particular, the user does not need the insight that the polynomial in Formula (1) is equal to $(x^{60} - 1)^2 + 0.001$. While these strategies are powerful, they inherit the downsides of other numerical approximation methods. For instance, neither of these strategies succeed in discharging Formula (1) when the variable x is unbounded even though the inequality still holds for any real number x. Moreover, in many cases, only approximations of the given formulas are used for reasoning. In general, Bernstein polynomials and other numerical tools like interval arithmetic can compute a tight bound for the range of a polynomial but not the exact range. Thus, the strategies `bernstein` and `interval` cannot prove that

$$x^{120} - 2x^{60} + 1 \geq 0, \tag{2}$$

whenever $x \in [0, 3]$, because 0 is the precise minimum of the polynomial on this interval.

This paper addresses shortcomings of numerical approximation methods for the special case of univariate polynomials. In particular, this paper presents a decision procedure, which is formally verified in PVS, that can be used to check satisfiability and validity of single-variable strict and non-strict polynomial inequalities where the variable is restricted to any interval, even an infinite or an open interval. For example, it can prove Formula (2) for values of x in $[0, 3]$, $(0, 3)$, $(-\infty, 0]$, $[3, \infty)$, $(-\infty, \infty)$, etc.

The decision procedure presented in this paper is based on Sturm's theorem. This is a theorem from real algebraic geometry that can be used to explicitly compute the number of roots of a univariate polynomial in a bounded, semi-open interval $(a, b]$. In the case where the polynomial does not have a multiple root at either a

or b, Sturm's Theorem provides a computable function σ such that the number of roots is equal to $\sigma(a) - \sigma(b)$. Moreover, the function σ can be evaluated at $-\infty$ and ∞, making it possible to explicitly count the number of roots in the intervals $(-\infty, b]$ and (a, ∞). Thus, Sturm's Theorem can be used to prove that a univariate polynomial is always positive (respectively, always negative) on an interval, provided that the polynomial does not have a multiple root at either endpoint. This proof can be done by checking that there are no roots in the interval and that the polynomial is nonnegative (respectively, nonpositive) at the endpoints.

Sturm's Theorem allows many useful results to be proved, but it does not directly apply to problems where an endpoint is a multiple root or where the polynomial bound is exact, e.g., $(x - \frac{1}{3})^2 \geq 0$ for all $x \in (0, \infty)$. This paper expands the functionality of Sturm's Theorem into a decision procedure that addresses both of these cases and applies to any connected interval, open or closed. To deal with endpoints that are multiple roots, the decision procedure finds a small positive number ϵ that is less than the distance between any two roots of the polynomial. It then perturbs by ϵ either endpoint that is a multiple root. The new interval contains the same number of roots as the original. This allows counting roots in any connected interval. Finally, to deal with the fact that Sturm's Theorem can not directly help prove the nonnegativity (respectively, nonpositivity) of a polynomial that has roots in a given interval, a branching algorithm is defined that progressively subdivides the original interval into smaller intervals until each smaller interval contains at most one root of the polynomial. Then nonnegativity (respectively, nonpositivity) is checked by whether or not the function is nonnegative (respectively, nonpositive) at each endpoint of these subintervals. As noted above, the final result is a decision procedure that can be used to prove bounds on any univariate polynomial on any connected interval. This decision procedure, which combines Sturm's Theorem and a branching algorithm, appears to be new to the literature. Sturm's Theorem is used most often to count roots rather than to prove nonnegativity or positivity.

The decision procedure presented in this paper has been formally proved in PVS. This paper presents not only the decision procedure itself, but also a proof-producing strategy in PVS, called `sturm`, for discharging existentially and universally quantified univariate polynomial inequalities. It can handle any such problem, it always terminates, and it is always correct. The soundness of the strategy only relies on the PVS deduction engine. In particular, the strategy does not depend on any trusted external oracle.

The formal development presented in this paper is electronically available in the contribution `Sturm` of the NASA PVS Library.[1] Examples are provided in the theory `Sturm@examples.pvs`. 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.

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

2 Sturm's Theorem

Sturm's Theorem [28] is easily stated. Let p be a univariate polynomial. A function σ is defined on the extended real line $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ by setting $\sigma(x)$ to be equal to the number of sign changes in the sequence

$$p_0(x),\ p_1(x),\ p_2(x),\ \ldots\ , p_m(x), \tag{3}$$

where

$$\begin{aligned} p_0(x) &= p(x), \\ p_1(x) &= p'(x), \\ \forall\, j > 1\ :\ \exists\, c \in \mathbb{R}+\ :\ p_j(x) &= -c \cdot rem(p_{j-2}, p_{j-1})(x), \\ p_m &= 0, \text{and} \\ \exists\, y \in \mathbb{R}\ : p_{m-1}(y) &\neq 0. \end{aligned} \tag{4}$$

Here, $rem(p_{j-2}, p_{j-1})$ is the remainder after polynomial division of p_{j-2} by p_{j-1}. Such a sequence is called a *Sturm sequence* of polynomial p. When counting the number of sign changes in the Sturm sequence, any zeros are ignored. For example, if $m = 7$ and $p_0(x) = 4$, $p_1(x) = -3$, $p_2(x) = -5$, $p_3(x) = 0$, $p_4(x) = 18$, $p_5(x) = -4$, $p_6(x) = -1$ and $p_7(x) = 0$, there are sign changes between $p_0(x)$ and $p_1(x)$, between $p_2(x)$ and $p_4(x)$, and between $p_4(x)$ and $p_5(x)$. In this case, the number of sign changes in the sequence is given by $\sigma(x) = 3$.

Note that the evaluation of a polynomial at $-\infty$ or ∞ is equal to $-\infty$ or ∞ depending on the degree of the polynomial and its leading coefficient. Thus, it is possible to count the number of sign changes in the sequence of polynomials even if x is equal to $-\infty$ or ∞. Also, it is worth noting that the introduction of the positive real number c in Formula (4) is not included in most statements of Sturm's Theorem where c is set to 1. However, a specific sequence will be constructed in the algorithm presented later that improves the computational efficiency of the standard method and uses different values for c, an idea credited to de Moura and Passmore [6]. Sturm's Theorem is stated as follows.

Theorem 1 (Sturm)**.** *For $a, b \in \bar{\mathbb{R}}$ with $a < b$, if either $p(a) \neq 0$ or $p'(a) \neq 0$, and if either $p(b) \neq 0$ or $p'(b) \neq 0$, then the number of roots of p in the interval $(a, b]$ is given by $\sigma(a) - \sigma(b)$.*

Numerous proofs of Sturm's Theorem can be found on the internet. The authors would be content to avoid redundancy and leave the proof out of this paper, except for the fact that there is often a gulf between how a theorem is proved on paper and how it is proved in an interactive theorem prover. The intended audience is both mathematicians and users of interactive theorem provers. The proof below breaks the main idea down into basic principles, and the authors hope that it can be replicated easily in another interactive theorem prover. It is not a verbatim copy of the PVS proof of the theorem, but it does use the same approach. This proof is the same in essence as several others that are found online, most notably the one found in Sottile's course notes [27]. The main idea behind the PVS proof is given here, which is inspired by the proof given in [27], although it differs in its key

approach in that it does not use a previous result about Sylvester sequences. The proof that is given here is both more elementary and more detailed.

Proof of Theorem 1. First, note that if either $a = -\infty$ or $b = \infty$, then there is another interval $(a^*, b^*]$ with real numbers as endpoints that is contained in $(a, b]$ with the same number of roots of p as the interval $(a, b]$, and such that for every j, $p_j(a^*)$ has the same sign as $p_j(a)$ and $p_j(b^*)$ has the same sign as $p_j(b)$. Thus, $\sigma_m(a^*) = \sigma_m(a)$ and $\sigma_m(b^*) = \sigma_m(b)$. Hence, it is sufficient to prove the result for the interval $(a^*, b^*]$. This shows that the proof can be reduced to the case where a and b are real numbers (not infinite). Thus, it is henceforth assumed that a and b are simply real numbers with $a < b$.

It is relatively easy to prove in an interactive theorem prover that there is some sequence

$$a = a_0 < a_1 < \cdots < a_k = b$$

of real numbers such that each closed interval $[a_i, a_{i+1}]$ contains at most one point that is the root of any p_j, and that this root is in the interior interval (a_i, a_{i+1}) unless either the root is a and $i = 0$ or the root is b and $i = k$. If Sturm's Theorem can be proved on each semi-open interval $(a_i, a_{i+1}]$, then the main result will trivially follow. Thus, instead of proving Sturm's Theorem when there are multiple roots, it is sufficient to assume that (1) there is at most one point that is a root of any of the p_j's in the closed interval $[a, b]$, and (2) if that root is equal to either a or b then it is not a root of both p and p'.

If there are no roots of any p_j in this closed interval then the intermediate value theorem implies that each $p_j(a)$ has the same sign as $p_j(b)$, so $\sigma(a) - \sigma(b) = 0$. This leaves only the case that there exists exactly one point that is a root of some p_j in $[a, b]$ and where this root can only be either a or b, if it is not a root of both p and p'. Denote this root r. It can be verified from the definition of the sequence $\{p_j\}$ that the following *properties* hold:

1. For all $j < m$, there exists a nonnegative integer k_j and another polynomial g_j such that $p(x) = (x - r)^{k_j} \cdot g_j(x)$ for all x, where $g_j(r) \neq 0$.

2. If $k_0 > 0$, then $k_1 = k_0 - 1$.

3. If $k_0 > 1$, then $k_j = k_0 - 1$ and $p_j(r) = 0$ whenever $1 \leq j < m$.

4. If two consecutive elements of the sequence $\{p_j\}$ both have a root at r, then every p_j has a root at r, $r \neq a$, $r \neq b$, $k_0 > 1$, and $k_j > 0$ for all j.

5. If $p_{m-1}(r) = 0$, then $k_j > 0$ and $p_j(r) = 0$ for all j.

6. If $k_0 \leq 1$, then p_{m-1} never changes sign on $[a, b]$.

7. If $p_j(r) \neq 0$, then $p_j(a)$ and $p_j(b)$ have the same nonzero sign.

8. If $k_0 \leq 1$, $1 \leq j < m$ and $p_j(r) = 0$, then $p_{j-1}(a)$ and $p_{j+1}(b)$ are nonzero and have opposite signs, and $p_{j-1}(a)$ and $p_{j+1}(b)$ are nonzero and have opposite signs.

9. If $k_0 > 1$ and k_0 is even, then $p_0(a)$ and $p_0(b)$ are nonzero and have the same sign, and for $0 < j < m$, $p_j(a)$ and $p_j(b)$ have opposite signs.

10. If $k_0 > 1$ and k_0 is odd, then $p_0(a)$ and $p_0(b)$ are nonzero and have opposite signs, and for $0 < j < m$, $p_j(a)$ and $p_j(b)$ are nonzero and have the same sign.

11. If $r \neq a$ and $r \neq b$, then $p_0(a)$ and $p_1(a)$ are nonzero have opposite signs, and $p_0(b)$ and $p_1(b)$ are nonzero and have the same sign.

12. If $r = a$ and $p_0(a) = 0$, then both $p_1(a)$ and $p_0(b)$ have the same sign as $p_1(b)$, which is nonzero.

13. If $r = b$ and $p_1(b) = 0$, then $p_0(a)$ and $p_1(a)$ are nonzero and have opposite signs, while $p_1(a)$ has the same sign as $p_1(b)$.

For the purpose of the proof, define $\sigma_j(x)$ to be the number of sign changes in the first j terms of the sequence in Formula (3), so that $\sigma = \sigma_m$. The proof is carried out in each case by induction on the subscript j in σ_j.

First consider the case where $r = a$. Then $k_0 \leq 1$ by Property 4. It is proved by induction on j that if $p_j(a) \neq 0$, then $p_t(a)$ and $p_t(b)$ have the same sign and that $\sigma_t(a) - \sigma_t(b) = 0$ for either $t = j$ or $t = j + 1$. The result will follow because $p_{m-1}(a)$ is nonzero by properties 5 and 4, and $\sigma_m = \sigma_{m-1}$. The base case is either $j = 0$ or $j = 1$. If $p_0(a) = 0$, then the base case is $j = 1$, and Property 12 implies that $p_1(a)$ and $p_1(b)$ have the same sign and that $\sigma_1(a) = \sigma_1(b) = 0$. If $p_0(a) \neq 0$, then the base case is $j = 0$, and Property 7 implies that $p_0(a)$ and $p_0(b)$ have the same sign and that $\sigma_0(a) = \sigma_0(b) = 0$. Now by induction, suppose that $p_j(r) \neq 0$. Then, by Property 7, $p_j(a)$ and $p_j(b)$ have the same nonzero sign. If $p_{j-1}(r) \neq 0$, then $p_{j-1}(a)$ and $p_{j-1}(b)$ have the same sign as well, so it follows immediately that $\sigma_j(a) - \sigma_{j-1}(a) = \sigma_j(b) - \sigma_{j-1}(b)$, and the result follows by induction. Conversely, if $p_{j-1}(r) = 0$, then Property 8 implies that $p_{j-1}(a)$ and $p_{j+1}(b)$ are nonzero and have opposite signs, and $p_{j-1}(b)$ and $p_{j+1}(b)$ are nonzero and have opposite signs. It follows immediately that $\sigma_{j+1}(a) = \sigma_{j-1}(a) + 1$ and $\sigma_{j+1}(b) = \sigma_{j-1}(b) + 1$. Furthermore, Property 4 implies that $p_{j+1}(a) \neq 0$. By Property 7, $p_{j+1}(a)$ and $p_{j+1}(b)$ have the same sign. This completes the proof in the case where $r = a$.

The case where $r = b$ follows directly from the cases where $r = a$ and where $a < r < b$. To see this, suppose that $r = b$ and note that there is some $\epsilon > 0$ such that no p_j has a root in the semi-open interval $(b, b + \epsilon]$. By applying the result to the interval $(a, b + \epsilon]$, which contains r in its interior, it can deduced that $\sigma_m(a) - \sigma_m(b+\epsilon) = 1$. Similarly, by applying the result to the interval $(b, b+\epsilon]$, whose greatest lower bound is r, it can deduced that $\sigma_m(b) - \sigma_m(b + \epsilon) = 0$. Combining these results yields $\sigma_m(a) - \sigma_m(b) = 1$. This reduces the argument to the case where $a < r < b$, which is assumed for the remainder of the proof.

First suppose that $k_0 > 0$, which by Property 3 implies that $p_j(r) = 0$ for all j. It is proved by induction on j that $\sigma_j(a) - \sigma_j(b) = 1$ for all $j \geq 1$. For the base case, it follows immediately from Property 11 that $\sigma_1(a) = 1$ and $\sigma_1(b) = 0$, and therefore $\sigma_1(a) - \sigma_1(b) = 1$. Define γ to be 1 if k_0 is even and -1 if k_0 is odd. For the inductive case, note that properties 9 and 10 imply that for all $j \geq 1$, the sign of $\sigma_j(a)$ is equal to $-\gamma$ multiplied by the sign of $\sigma_j(b)$. From this, it can be seen

that $p_{j+1}(a)$ and $p_j(a)$ have opposite sign if and only if $p_{j+1}(b)$ and $p_j(b)$ do as well, implying that $\sigma_{j+1}(a) - \sigma_j(a) = \sigma_{j+1}(b) - \sigma_j(b)$, and the result follows. This reduces the proof to the case where $a < r < b$ and $k_0 \leq 1$.

It is now proved by induction that for all $j \geq 1$, $\sigma_j(a) - \sigma_j(b) = 1$ whenever $p_j(r) \neq 0$. The result follows because Property 6 implies that $p_{m-1}(r) \neq 0$ and $\sigma_m = \sigma_{m-1}$. For the base case, it follows immediately from Property 11 that $\sigma_1(a) = 1$ and $\sigma_1(b) = 0$, and therefore $\sigma_1(a) - \sigma_1(b) = 1$. For the inductive case, suppose that $j > 1$ and that $p_j(r) \neq 0$.

If $p_{j-1}(r) \neq 0$, then Property 7, when applied to both $j-1$ and j, gives that $p_{j-1}(a)$ and $p_{j-1}(b)$ have the same nonzero sign, and that $p_j(a)$ and $p_j(b)$ have the same nonzero sign. From this, it can be seen that $p_{j-1}(a)$ and $p_j(a)$ have opposite sign if and only if $p_{j-1}(b)$ and $p_j(b)$ do as well, implying that $\sigma_j(a) - \sigma_{j-1}(a) = \sigma_j(b) - \sigma_{j-1}(b)$, and the result follows.

Finally, suppose that $p_{j-1}(r) = 0$. Property 4 implies that $p_{j-2}(r)$ and $p_j(r)$ are both nonzero. Property 8 implies that $p_{j-2}(a)$ and $p_j(a)$ are nonzero and have opposite signs and that $p_{j-2}(b)$ and $p_j(b)$ are nonzero and have opposite signs. Thus, $\sigma_j(a) = \sigma_{j-2}(a) + 1$ and $\sigma_j(b) = \sigma_{j-2}(b) + 1$. If $j - 2 \geq 1$, then the inductive hypothesis can be applied to deduce that $\sigma_j(a) - \sigma_j(b) = 1$. This completes the proof except in the case where $j = 2$. However, then $j - 1 = 1$, and so by hypothesis $p_1(r) = 0$. Property 1 would then imply that $k_1 > 0$, and so by Property 2, $k_0 > 1$, a contradiction, since it is assumed in this part of the proof that $k_0 \leq 1$. □

Sturm's Theorem is less efficient than other methods at isolating roots. However, as this paper shows, Sturm's Theorem works well for determining whether the polynomial is always positive, negative, non-zero, etc., on a given interval.

3 Sturm Sequence of Integer Polynomials

An important element in the definition of a decision procedure for polynomial reasoning is a function that explicitly computes the remainder between two polynomials. For the development presented in this paper, pseudo division is used rather than standard polynomial division. This removes the need for dividing coefficients in the algorithm, which means that if the coefficients of the original polynomials are integers, then the coefficients of their remainder after division will also be integers. In fact, the pseudo division algorithm in PVS is defined only for integer polynomials.

The remainder after pseudo division of a polynomial g by a polynomial h is not equal to the remainder after standard division, but is a power of the leading coefficient of h multiplied by the standard remainder. This power can be either positive or negative. In the PVS development, the pseudo remainder is multiplied by a constant so that this multiple is always negative. This ensures that the conditions in Formula (4), defining the polynomial remainder sequence, all hold. Moreover, pseudo division can be costly in terms of memory, because avoiding divisions can cause integer coefficients to become quite large in the pseudo remainder. Thus, in the PVS development, the pseudo remainder of two integer polynomials is also multiplied by the reciprocal of the greatest common division of all its coefficients, which still ensures that the output is an integer polynomial. This multiplication helps to

mitigate coefficient growth, and since the greatest common divisor is positive, it does not affect the conditions in Formula (4). In the PVS development, the function gcd on polynomials is defined such that $\gcd(p)$ is equal to the greatest common divisor of the coefficients of p whenever p is nonzero.

Formally, a *univariate polynomial function* p is a function from real numbers to real numbers such that

$$p(x) \equiv \sum_{i=0}^{n} c_i x^i, \tag{5}$$

where $c_n \neq 0$. The natural number n is the degree of p and real numbers c_i are the coefficients of p. If c_i is an integer for every $i \leq n$ then p will be called an *integer polynomial*. If each such c_i is a rational number, then p will be called a *rational polynomial*. In order to specify functions and properties involving polynomials in an interactive theorem prover, it is necessary to define a data structure to represent them. The development presented in this paper uses a degree-dense representation where a polynomial is a list of numerical coefficients of type T, i.e., the i-th element of the list represents the coefficient of the x^i monomial. The type T can be `real`, for real polynomials, `int`, for integer polynomials, or `rat`, for rational polynomials. Nothing in this paper fundamentally depends on this particular representation. For readability, when there is no ambiguity, lower case letters $f, g, \ldots, p, q, \ldots$ will be used for both the mathematical concept of polynomial, e.g., Formula (5), and the polynomial data structure used in the definition of PVS functions, e.g., Formula (6) below. In the latter case, the expressions $\deg(p)$ and $\texttt{coeff}(p, i)$ refer to the degree of p and the i-th coefficient of the polynomial p, respectively. The same remark applies to p', etc. It denotes the mathematical definition of the derivative of p and the data structure that represents this polynomial.

The adjusted pseudo remainder can be calculated for two integer polynomials g and h as follows.

$$\begin{array}{l}
\texttt{adjusted_rem}(g, h) \equiv \\
\quad \texttt{let } r = \texttt{pseudo_rem}(g, h), \\
\qquad d = \gcd(p) \texttt{ in} \\
\quad \texttt{if coeff}(h, \deg(h)) > 0 \ \vee \ \texttt{mod}(\deg(g) - \deg(h) + 1, 2) = 0 \\
\qquad \texttt{then } -r/d \\
\qquad \texttt{else } r/d \texttt{ endif}.
\end{array} \tag{6}$$

Here, $\texttt{pseudo\_rem}(g, h)$ is the remainder after pseudo division of g by h. Its definition is common enough that it is left out here, although it can be found in the PVS development. Also, note that $\texttt{coeff}(h, \deg(h))$ is the leading coefficient of h, i.e., the nonzero coefficient with the highest degree. It can be verified that since g and h both have integer coefficients, the polynomial $\texttt{adjusted\_rem}(g, h)$ does as well.

The function `comp_rem_seq`, defined below, computes the remainder sequence for any two integer polynomials g and h such that the degree of h is less than the degree of g. It has a list ℓ of integer polynomials as an input, which is also used as

an accumulator to store the sequence that is recursively computed by the function.

$$\begin{array}{l}\texttt{comp\_rem\_seq}(g,h,\ell) \equiv \\ \quad \texttt{if length}(\ell) = 0 \texttt{ then comp\_rem\_seq}(g,h,\texttt{cons}(g,\ell)) \\ \quad \texttt{elsif } \deg(\texttt{head}(\ell)) = 0 \texttt{ then } \ell \\ \quad \texttt{elsif length}(\ell) = 1 \texttt{ then comp\_rem\_seq}(g,h,\texttt{cons}(h,\ell)) \\ \quad \texttt{else let } p = \texttt{adjusted\_rem}(\texttt{head}(\texttt{tail}(\ell)),\texttt{head}(\ell)) \texttt{ in} \\ \quad\quad \texttt{comp\_rem\_seq}(g,h,\texttt{cons}(p,\ell)) \\ \quad \texttt{endif}.\end{array} \tag{7}$$

The function `sturm_seq`, defined below, computes a Sturm sequence of an integer polynomial p.

$$\texttt{sturm\_seq}(p) \equiv \texttt{comp\_rem\_seq}(p, p', \phi), \tag{8}$$

where ϕ refers to the empty list. A curious reader may have noticed that `sturm_seq` computes a Sturm sequence in reverse order, i.e., if $\ell = \texttt{sturm\_seq}(p)$, p_m in Formula (3) corresponds to $\texttt{head}(\ell)$, p_{m-1} corresponds to $\texttt{head}(\texttt{tail}(\ell))$, and so on. It can be easily checked that the sign changes of ℓ and $\texttt{reverse}(\ell)$ are equal.

4 Decision Procedure for Integer Polynomials

This section presents a decision procedure for computing the *sign* of an integer polynomial p, i.e., its positivity, nonpositivity, negativity, nonnegativity, or nonzero property, on a nonempty interval, which may or may not have infinite endpoints. This decision procedures depends on a function that explicitly computes the number of roots of p in that interval. One immediately obvious problem with using the function σ from Sturm's Theorem to define such function is that it will not work when either the lower bound or the upper bound of the interval is a multiple root of the polynomial, i.e., when the polynomial and its derivative both have roots at that point.

The problem of multiple roots at the endpoints of an interval can be addressed by perturbing such bounds outward by a small amount so that the new interval contains exactly the same number of roots as the original but has endpoints that are not multiple roots. To see how such a perturbation can be computed, let r be a root of p, so $p(r) = 0$. A function can be defined with p and r as inputs that computes the width of a small interval around r that contains no other roots of p. The first step is to compute the degree of the first successive derivative of p that does not vanish at r, which is accomplished with the following recursive function.

$$\begin{array}{rl}\texttt{md}(p,r) \equiv & \texttt{if } \deg(p) = 0 \ \vee \ p(r) \neq 0 \texttt{ then } \deg(p) \\ & \texttt{else md}(p',r) \texttt{ endif}.\end{array} \tag{9}$$

The function `md` is well defined since the degree of the polynomial that is passed as parameter strictly decreases at each recursive call.

By using Taylor's Theorem, p is approximated in a small neighborhood of r by $p^{(n-\texttt{md}(p,r))}(r) \cdot (x-r)^{n-\texttt{md}(p,r)}$. The error is bounded by a constant times

$(x-r)^{n-\mathtt{md}(p,r)+1}$, where $p^{(n-\mathtt{md}(p,r))}(r)$ is the $(n-\mathtt{md}(p,r))$-th derivative of p at r. The interval around r containing no other roots is determined by computing a neighborhood of r on which this derivative is always nonzero. Since it is always nonzero, it can be proved by induction and the mean value theorem that all lesser derivatives vanish only at r on this neighborhood. This neighborhood is computed as follows. First, the following function is defined that takes as inputs polynomial p, $x \in \mathbb{R},$, and $\epsilon \in \mathbb{R} > 0$, and returns a $\delta \in \mathbb{R} > 0$ so that any input within δ of x will have value less than ϵ from x when p is applied to both points.

$$\begin{aligned}
&\mathtt{pcc}(p,x,\epsilon) \equiv \\
&\quad \mathtt{let}\ n = \deg(p), \\
&\qquad c = \max_{i=0}^{n} |\mathtt{coeff}(p,i)| + 1\ \mathtt{in} \\
&\quad \mathtt{if}\ n = 0\ \mathtt{then}\ 1/2 \\
&\quad \mathtt{else}\ \min\left(\frac{\epsilon}{c}\left(1 + \sum_{i=1}^{n}\sum_{j=1}^{i}\binom{i}{j-1}|x|^{j-1}\right), \frac{1}{2}\right) \\
&\quad \mathtt{endif}.
\end{aligned} \tag{10}$$

Using the function `pcc`, it is now possible to define a radius around the point r in which the polynomial p has no other roots. This radius is computed with the function `root_rad`.

$$\begin{aligned}
\mathtt{root_rad}(p,r) \equiv\ &\mathtt{let}\ n = \deg(p)\ \mathtt{in} \\
&\mathtt{pcc}(p^{(n-\mathtt{md}(p,r))}, r, |p^{(n-\mathtt{md}(p,r))}(r)|).
\end{aligned} \tag{11}$$

Note that the function `root_rad` can be used to compute the number of roots in any interval, not simply an interval without a multiple root at either endpoint. To see this by example, note that if $(a,b]$ is an interval of real numbers such that b is a multiple root of p but a is not, then the interval $(a, b+\mathtt{root\_rad}(p,b)/2]$ contains the same number of roots as $(a,b]$ but neither endpoint is a multiple root.

The next step in the development is to define a function called `roots_cl_int` with two extended real numbers a and b as inputs, with $a < b$, along with an integer polynomial p. The function returns the number of roots in the closed interval $[a,b]$. The other input to this function is a list ℓ of polynomials, which in practice is set to $\mathtt{sturm\_seq}(p)$.

$$\begin{aligned}
&\mathtt{roots_cl_int}(p,a,b,\ell) \equiv \\
&\quad \mathtt{let}\ a^* = \mathtt{if}\ a = -\infty\ \vee\ p(a) \neq 0\ \vee\ p'(a) \neq 0\ \mathtt{then}\ a \\
&\qquad\qquad \mathtt{else}\ a - \mathtt{root_rad}(p,a)/2\ \mathtt{endif} \\
&\qquad b^* = \mathtt{if}\ b = \infty\ \vee\ p(b) \neq 0\ \vee\ p'(b) \neq 0\ \mathtt{then}\ b \\
&\qquad\qquad \mathtt{else}\ b + \mathtt{root_rad}(p,b)/2\ \mathtt{endif} \\
&\qquad c\ = \mathtt{if}\ a \neq -\infty\ \wedge\ p(a) = 0\ \wedge\ p'(a) \neq 0\ \mathtt{then}\ 1 \\
&\qquad\qquad \mathtt{else}\ 0\ \mathtt{endif} \\
&\quad \mathtt{in}\ \mathtt{sigma}(\ell, a^*) - \mathtt{sigma}(\ell, b^*) + c.
\end{aligned} \tag{12}$$

The definition of `roots_cl_int` uses the function `sigma`, which implements the function σ from §2. This function takes as inputs a finite list ℓ of polynomials and an extended real number r, and returns the number of sign changes in that list when each element is evaluated at r. The introduction of the number c in the definition of `roots_cl_int` addresses the limitation in Sturm's Theorem, which only gives a way to count the number of roots in a half open interval that does not include its lower bound. The term c adjusts this number based on whether the lower bound is equal to the newly computed a^* and is also a root of p but not of its derivative. Finally, it should also be noted that the numbers a and b in the definition of `roots_cl_int` are extended real numbers and therefore may be equal to $-\infty$ or ∞. There is no built-in support for extended reals in PVS. Hence, a simple data structure for representing these numbers has been defined. Using this data structure, the evaluation of a polynomial p at an extended real $r \in \{-\infty, \infty\}$ is defined as $\pm\infty$, where the sign of the infinite number is given by $\text{sign}(\texttt{coeff}(p, \deg(p)))$, if $\deg(p)$ is even, and $\text{sign}(r) \cdot \text{sign}(\texttt{coeff}(p, \deg(p)))$, otherwise. Furthermore, it is assumed that $-\infty$ is smaller than any other extended real number and ∞ is greater than any other extended real number. No other operations are defined on infinite extended real numbers.

The following theorem states the correctness of the function `roots_cl_int`. It has been formally proved in PVS, where it is called `roots_closed_int_def`.

Theorem 2. *Let $a, b \in \bar{\mathbb{R}}$, with $a < b$, p be an integer polynomial, and $S = \{r \in \mathbb{R} \mid a \leq r \leq b \text{ and } p(r) = 0\}$. If $N = \texttt{roots\_cl\_int}(p, a, b, \texttt{sturm\_seq}(p))$ then $N \geq 0$ and there exists a bijection $\beta\colon \{0, 1, \ldots, N-1\} \to S$.*

The next step in the PVS development is to use the function `roots_cl_int` to define a function `roots_interval` that computes the number of roots of an integer polynomial in any interval, whether it is closed, open, half open and half closed, unbounded, and whether or not the polynomial has a multiple root at an endpoint of the interval. The precise definition of this function is omitted from this description, because it is straightforward to define it directly in terms of the function `roots_cl_int`. The reader is referred to the PVS development Sturm in the NASA PVS Library for the precise definition of this function. Basically, the number of roots in the closure of the given interval is computed using `roots_cl_int`, and this number is then adjusted upward or downward depending on whether the lower bound or the upper bound of the interval is a root of the polynomial, and whether the interval itself actually contains this bound. The function `roots_interval` takes as inputs an integer polynomial p, two extended real numbers a and b as inputs, with $a < b$, as well as two Boolean values `cont_lb` and `cont_ub`, which refer to whether the interval contains its lower or upper bound, respectively. That is, if $a = -10$, $b = 7$, `cont_lb` = `true`, and `cont_ub` = `false`, then the corresponding half open interval is $[-10, 7)$. The following theorem has been proved in PVS and can be found in the PVS development with the name `number_roots_interval_def`.

Theorem 3. *Let $a, b \in \bar{\mathbb{R}}$, with $a < b$, p be an integer polynomial, and*

$$S = \{r \in \mathbb{R} \mid a \leq r \leq b \text{ and } p(r) = 0 \text{ and } (r = a \implies \texttt{cont\_lb} = \texttt{true}) \text{ and } (r = b \implies \texttt{cont\_ub} = \texttt{true})\}.$$

If $N = \texttt{roots\_interval}(p, a, b, \texttt{cont\_lb}, \texttt{cont\_ub}, \texttt{sturm\_seq}(p))$ then $N \geq 0$ and there exists a bijection $\beta \colon \{0, 1, \ldots, N-1\} \to S$.

It can now be noted that if a polynomial is always positive on a given interval, it is trivial to prove that this is true using the function `roots_interval`. All that must be checked is that the function `roots_interval` returns 0, so that the polynomial has no roots on the interval, and that it is positive at any fixed point in the interval, such as $(a+b)/2$ in the case where $a, b \in \mathbb{R}$. Thus, the difficulty when determining whether the polynomial p satisfies $p(r)\ R\ 0$ for all r in a given interval, when R is a relation in $\{>, <, \neq, \geq, \leq\}$ lies in the case when R is nonstrict, i.e., when $R \in \{\geq, \leq\}$. Moreover, if a decision procedure can be defined for when R is $\geq$, then it can be defined for when R is $\leq$ as well by just replacing p with $-p$. Thus, the next step is to define a specific decision procedure that determines whether an integer polynomial is always nonnegative on a bounded closed interval. The PVS function `nonneg_int` takes as inputs an integer polynomial p, real numbers x and y (not extended real numbers), and a list ℓ of polynomials, which in practice is set to $\texttt{sturm\_seq}(p)$. It returns a Boolean, which is equal to `true` if and only if $p(r) \geq 0$ for r in the closed, bounded interval $[x, y]$.

The function `nonneg_int` works by recursively subdividing the interval $[x, y]$ into left and right halves, until each subinterval is small enough that it contains at most one root of the polynomial p. Then, the polynomial p is evaluated at each endpoint of each subinterval. One important fact that is used in the verification of this function is that a continuous function with at most one root in a closed, bounded interval is always nonnegative on that interval if and only if it is nonnegative at its endpoints. This result follows directly from the intermediate value theorem. The recursive function `nonneg_int` is defined below.

$$\begin{aligned}
&\texttt{nonneg_int}(p, x, y, \ell) \equiv \\
&\quad \texttt{if } x > y \texttt{ then true} \\
&\quad \texttt{elsif } x = y \texttt{ then } p(x) \geq 0 \\
&\quad \texttt{elsif roots_cl_int}(p, x, y, \ell) \leq 1 \texttt{ then } p(x) \geq 0 \ \wedge\ p(y) \geq 0 \qquad (13) \\
&\quad \texttt{else nonneg_int}(p, x, (x+y)/2, \ell) \wedge \\
&\qquad\quad \texttt{nonneg_int}(p, (x+y)/2, y, \ell) \\
&\quad \texttt{endif}.
\end{aligned}$$

The function `nonneg_int` is a decision procedure for nonnegativity on closed, bounded intervals. However, it should be noted that a polynomial is always nonnegative on any given interval if and only if it is nonnegative on the closure of that interval. For instance, p is always nonnegative on the open interval $(0, 1)$ if and only if it is nonnegative on the closed interval $[0, 1]$. Thus, the function `nonneg_int` can be used

as a decision procedure on any bounded interval, even if it is open. To extend this function to unbounded intervals, a number is computed that can be guaranteed to bound all of the roots of the integer polynomial p. This bound is used to reduce any unbounded interval to a bounded interval on which the polynomial is nonnegative if and only if it is nonnegative on the original unbounded interval.

$$\texttt{root\_bound}(p) \;\equiv\; \max\left(2,\; \sum_{i=0}^{\deg(p)-1} \frac{|\texttt{coeff}(p,i)|}{|\texttt{coeff}(p,\deg(p))|}\right). \tag{14}$$

It can be proved that any root of p must lie in the bounded interval

$$(-\texttt{root\_bound}(p), \texttt{root\_bound}(p)).$$

The function `nonneg_int` can now be used to define a function `always_nonneg` that takes as inputs an integer polynomial p and two extended (so possibly unbounded) real numbers a and b with $a < b$. It returns `true` precisely when p is always nonnegative on (a, b), which, as noted above, is equivalent to p being nonnegative on the closure of this interval. Thus, this function does not need inputs that determine whether the interval contains its endpoints, since that information is irrelevant to nonnegativity.

$$\begin{array}{l}
\texttt{always_nonneg}(p,a,b) \equiv \\
\quad \texttt{let } \ell = \texttt{sturm_seq}(p), \\
\qquad M = \texttt{root_bound}(p) \texttt{ in} \\
\quad \texttt{if } a \neq -\infty \;\wedge\; b \neq \infty \texttt{ then } \texttt{nonneg_int}(p,a,b,\ell) \\
\quad \texttt{elsif } b \neq -\infty \texttt{ then } \texttt{nonneg_int}(p,\min(-M,b-1),b,\ell) \\
\quad \texttt{elsif } a \neq \infty \texttt{ then } \texttt{nonneg_int}(p,a,\max(M,a+1),\ell) \\
\quad \texttt{else } \texttt{nonneg_int}(p,-M,M,\ell) \\
\quad \texttt{endif}.
\end{array} \tag{15}$$

The following theorem has been proved in PVS and can be found in the PVS development under the name `always_nonnegative_def`.

Theorem 4. *If $a, b \in \bar{\mathbb{R}}$, with $a < b$, and p is an integer polynomial, then*

$$\textit{always\_nonneg}(p,a,b) = \textit{true}$$

if and only if every real number x, with $a \leq x \leq b$, satisfies $p(x) \geq 0$.

Note again that the continuity of p implies that this theorem is true if either (or both) of the $\leq$ signs is replaced with a $<$ sign.

The decision procedure that determines whether $p(x)\, R\, 0$ for all x in a given interval, and where R is a relation in $\{>, <, \neq, \geq, \leq\}$, can now be defined as follows. If R is either $\geq$ or $\leq$, then the function `always_nonneg` is used for either p or $-p$ (respectively). Otherwise, it is simply checked that the function `roots_interval` returns 0 on the interval and that a given point r in the interval satisfies $p(r)\, R\, 0$.

The correctness of the procedure then follows from the intermediate value theorem. As above, the interval here is represented by two extended real numbers a and b, as well as two Boolean values `cont_lb` and `cont_ub`, which refer to whether the interval contains a and b, respectively. The point r can be chosen in a number of ways. For this development, it is simply assumed that there is a function `choose` with inputs a, b, `cont_lb`, and `cont_ub` as inputs that picks a point in this interval. In the actual PVS development, r is defined as the midpoint of the non-empty proper interval $[c, d]$, where c and d are real numbers and $c < d$. The numbers c, d are defined as follows. If a and b are both real numbers then $c = a$ and $d = b$; if both a and b are infinite numbers, $c = -1$ and $d = 1$; if a is a real number and b is $+\infty$, then $c = a$ and $d = a + 1$; otherwise, $c = b - 1$ and $d = b$. The input p to the function `compute_poly_sat`, defined below, must be an integer polynomial.

$$\begin{array}{l}\texttt{compute\_poly\_sat}(p, a, b, \texttt{cont\_lb}, \texttt{cont\_ub}, R) \equiv \\ \quad \texttt{if } R = (\geq) \texttt{ then always\_nonneg}(p, a, b) \\ \quad \texttt{elsif } R = (\leq) \texttt{ then always\_nonneg}(-p, a, b) \\ \quad \texttt{else roots\_interval}(p, a, b, \texttt{cont\_lb}, \texttt{cont\_ub}, \texttt{sturm\_seq}(p)) \\ \qquad \wedge\ R(p(\texttt{choose}(p, a, b, \texttt{cont\_lb}, \texttt{cont\_ub})), 0) \\ \quad \texttt{endif}.\end{array} \tag{16}$$

The following theorem has been formally proved in PVS and can be found in the PVS development under the name `poly_sat_correct`.

Theorem 5. *If p is a nonzero integer polynomial, a and b are extended real numbers, and R is a relation in $\{>, <, \neq, \geq, \leq\}$, then*

$$\textit{compute\_poly\_sat}(p, a, b, \textit{cont\_lb}, \textit{cont\_ub}, R) = \textit{true}$$

if and only if $p(x)\, R\, 0$ for all real numbers x in I, where I is the interval corresponding to a, b and the Boolean values `cont_lb` and `cont_ub`.

Example 1. *If $a = 0$, $b = 3$, `cont_lb` = `false`, and `cont_ub` = `false`, then the corresponding interval is I is $(0, 3)$. If p is the integer polynomial in §1 given by $x^{120} - 2x^{60} + 1$, then $p(x) \geq 0$ for all x in I. In fact, p factors as $(x^{60} - 1)^2$. It can be checked that*

$$\begin{array}{l}\textit{compute\_poly\_sat}(p, 0, 3, \textit{false}, \textit{false}, \geq) = \textit{true}, \\ \textit{compute\_poly\_sat}(p, 0, 3, \textit{false}, \textit{false}, >) = \textit{false}.\end{array} \tag{17}$$

5 Decision Procedure for Rational Polynomials

If the polynomial p has integer coefficients and $R \in \{>, <, \neq, \geq, \leq\}$, the function `compute_poly_sat`, defined in §4, decides whether or not $p(x)\, R\, 0$ holds on any given interval. Thus, if p has *rational* coefficients, the relation $p(x)\, R\, 0$ can be decided on any given interval by multiplying this relational formula by the product of the denominators of coefficients in p and then using the procedure `compute_poly_sat` on the resulting polynomial.

In order to multiply the polynomial p by the product of all of its coefficients' denominators, a function must be defined that computes the denominator of a given coefficient. In a theorem prover like PVS, where rational numbers are built-in, defining such as function is not straightforward. For example, in PVS, the terms $\frac{1}{2}$ and $\frac{2}{4}$ are indistinguishable. However, there are several ways in which a function that computes the numerator and denominator of a rational number can be defined in PVS. For example, the PVS function that computes the rational decomposition uses a recursive function called `compute_pos_rat` that continually simplifies a positive rational number, making its numerator and denominator smaller and removing the integer part at each steps. In the final recursive call, the function is called on an integer, making the answer trivial. The function takes a positive rational number and return as a pair of positive integers, the first being the numerator and the second being the denominator.

$$\begin{array}{l}\texttt{compute\_pos\_rat}(r) \equiv \\ \quad \texttt{if } \texttt{floor}(r) = r \texttt{ then } (\texttt{floor}(r), 1) \\ \quad \texttt{elsif } r > 1 \texttt{ then let } (a, b) = \texttt{compute\_pos\_rat}(r - \texttt{floor}(r)) \texttt{ in} \\ \quad\quad (b \cdot \texttt{floor}(r) + a, b) \\ \quad \texttt{else let } (a, b) = \texttt{compute\_pos\_rat}(1/r) \texttt{ in } (b, a) \texttt{ endif}.\end{array} \tag{18}$$

The function `compute_pos_rat` can then be used to define two other functions `numerator` and `denominator`, which are simply given by the first and second componet of the output of `compute_pos_rat`. It has been proved in PVS that if r is any positive rational number, then $r = \texttt{numerator}(r)/\texttt{denominator}(r)$. A challenging part of the proof is showing that the function `compute_pos_rat` terminates. In PVS, showing termination requires that the function be given a measure, which is a function on the inputs of the function that returns a natural number and strictly decreases in value every time the function is called recursively. In PVS, the measure function on r is defined by

$$\texttt{pos\_rat\_meas}(r) \;\equiv\; \texttt{if } r < 1 \texttt{ then } 10^g \texttt{ else } 10^g - 1 \texttt{ endif}, \tag{19}$$

where g is the least integer such that there exists positive rational numbers a and b with $r = a/b$ and $g = a + b$.

The final step before defining a decision procedure for rational polynomials is to define a function that takes as an input a rational polynomial p and returns an integer polynomial that is a positive multiple of the first. This integer polynomial is obtained by multiplying p by the product of the denominators of its coefficients. Each coefficient supplies one integer to this product. If c is one of the coefficients, then this number is 1 if c is zero and it is $|\texttt{denominator}(c)|$ otherwise. A function `rat2poly` is defined in PVS that takes a rational polynomial and coverts it to an integer polynomial. This process works by recursively considering each coefficient of the polynomial. At each step, it multiplies the polynomial by the denominator of the coefficient in question, and it also stores the current greatest common divisor of all resulting integer coefficients that it has simplified so far in the recursion. At the end of the recursion, all of the coefficients are divided by this greatest common

divisor to simplify the answer. The fact that `rat2poly`(p) is a positive multiple of p is specified in the output type of the function `rat2poly`. Using this function, the decision procedure for rational polynomials is defined below.

$$\begin{aligned}&\texttt{compute\_poly\_sat\_rat}(p, a, b, \texttt{cont\_lb}, \texttt{cont\_ub}, R) \equiv \\ &\quad \texttt{compute\_poly\_sat}(\texttt{rat2poly}(p), a, b, \texttt{cont\_lb}, \texttt{cont\_ub}, R).\end{aligned} \tag{20}$$

The following theorem has been formally proved in PVS and is found in the PVS development under the name `poly_sat_rat_correct`.

Theorem 6. *Let p be a nonzero rational polynomial, $a, b \in \bar{\mathbb{R}}$, and R be a relation in $\{>, <, \neq, \geq, \leq\}$.*

$$\textit{compute\_poly\_sat\_rat}(p, a, b, \textit{cont\_lb}, \textit{cont\_ub}, R) = \textit{true}$$

if and only if $p(x)\, R\, 0$ for all real numbers x in I, where I is the interval corresponding to a, b and the Boolean values `cont_lb` and `cont_ub`.

Example 2. *If $a = -\infty$, $b = 3$, `cont_lb` = `false`, and `cont_ub` = `false`, then the corresponding interval I is $(-\infty, 3)$. If p is the rational polynomial in §1 given by $x^{120} - \frac{2}{3}x^{60} + \frac{1}{9}$, then $p(x) \geq 0$ for all x in I. In fact, p factors as $(x^{60} - \frac{1}{3})^2$. It can be seen that*

$$\begin{aligned}&\textit{compute\_poly\_sat\_rat}(p, -\infty, 3, \textit{false}, \textit{false}, \geq) = \textit{true}, \\ &\textit{compute\_poly\_sat\_rat}(p, -\infty, 3, \textit{false}, \textit{false}, >) = \textit{false}.\end{aligned} \tag{21}$$

6 Automated Strategy

The decision procedure implemented by the function `compute_poly_sat_rat` and its correctness property (Theorem 6) can be used to discharge, in a methodical way, PVS sequents involving univariate polynomial inequalities.

In a nutshell, a PVS sequent is a logical judgement of the form $\Gamma \vdash \Delta$, where Γ, called the antecedent, and Δ, called the consequent, are lists of logical formulas. The intuitive meaning of a sequent is that from the conjunction of formulas in Γ, the disjunction of formulas in Δ can be deduced. PVS proof commands are logical rules that transforms a sequent into a set of sequents, with the objective of producing sequents where the formula `false` appears in the antecedent or the formula `true` appears in the consequent. When all sequents generated by a proof command are of one of these forms, the initial sequent is *discharged*. Hence, a *proof* in PVS can be represented by a tree of proof commands that discharge an initial sequent.

The following example describes a PVS proof of the polynomial inequality given in Example 2 using the development presented in §5.

Example 3. *Consider the sequent below displayed in a vertical layout. Formulas in a PVS sequent are numbered using the notation $\{n\}$, where $n < 0$ in the antecedent and $n > 0$ in the consequent. In this sequent, x is a free real variable (actually, a Skolem real constant in PVS terminology).*

$$\begin{array}{l}\{-1\}\ x < 3 \\ \vdash \\ \{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0\end{array}$$

1. *The first step in the PVS proof is to execute a proof command that instantiates Theorem* `poly_sat_rat_correct` *with* $a = -\infty$, $b = 3$, `cont_lb` $=$ `false`, `cont_ub` $=$ `false`, *and* $p =$ `p`, *where* `p` *is a list of rational numbers that is* 0 *everywhere except in the positions* 0, 60, *and* 120, *where it has the values* $\frac{1}{9}$, $-\frac{2}{3}$, *and* 1, *respectively. It should be noted that* `p` *is a PVS data structure that represents the polynomial* $p(x) = x^{120} - \frac{2}{3}x^{60} + \frac{1}{9}$. *The notation* $\mathtt{p}(x)$ *will be used to represent the evaluation of the polynomial represented by the data structure* `p` *on* x. *That proof command yields the following sequent.*

$$\begin{array}{l}
\{-1\}\ \mathtt{compute_poly_sat_rat}(\mathtt{p}, -\infty, 3, \mathtt{false}, \mathtt{false}, \geq) \iff \\
\qquad (\forall(x\colon \mathbb{R})\colon x \in (-\infty, 3) \implies \mathtt{p}(x) \geq 0) \\
\{-2\}\ x < 3 \\
\vdash \\
\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0
\end{array}$$

2. *Next, a proof rule that evaluates*

$$\mathtt{compute\_poly\_sat\_rat}(\mathtt{p}, -\infty, 3, \mathtt{false}, \mathtt{false}, \geq)$$

is executed. This evaluation, which only involves computable functions and ground terms, is efficiently performed by the PVS ground evaluator. The following sequent is obtained.

$$\begin{array}{l}
\{-1\}\ \mathtt{true} \iff (\forall(x\colon \mathbb{R})\colon x \in (-\infty, 3) \implies \mathtt{p}(x) \geq 0) \\
\{-2\}\ x < 3 \\
\vdash \\
\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0
\end{array}$$

3. *Sequent formula* $\{-1\}$ *can be easily reduced using propositional simplification to*

$$\begin{array}{l}
\{-1\}\ \forall(x\colon \mathbb{R})\colon x \in (-\infty, 3) \implies \mathtt{p}(x) \geq 0 \\
\{-2\}\ x < 3 \\
\vdash \\
\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0
\end{array}$$

4. *Next, the quantified variable* x *in sequent formula* $\{-1\}$ *is instantiated with the Skolem constant* x *appearing in sequent formulas* $\{-2\}$ *and* $\{1\}$. *This instantiation yields the sequent.*

$$\begin{array}{l}
\{-1\}\ x \in (-\infty, 3) \implies \mathtt{p}(x) \geq 0 \\
\{-2\}\ x < 3 \\
\vdash \\
\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0
\end{array}$$

5. *The elimination of the implication in sequent formula* $\{-1\}$ *yields two sequents.*

$$\begin{array}{ll}
\{-1\}\ x < 3 & \{-1\}\ \mathtt{p}(x) \geq 0 \\
\vdash & \{-2\}\ x < 3 \\
\{\ 1\ \}\ x \in (-\infty, 3) & \vdash \\
\{\ 2\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0 & \{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0
\end{array}$$

6. *The two sequents generated in the previous step can be easily discharged by unfolding the definitions of the operations "$\in$" and "$\mathtt{p}(x)$", respectively.*

The PVS proof in Example 3 illustrates a well-known technique in theorem proving known as *computational reflection* [12]. Given an object theory, such as the theory of univariate rational polynomials, a computable target theory is defined such that elements of the object theory are embedded using data structures in the target theory, e.g., formulas $x \in (-\infty, 3)$ and $\mathtt{p}(x) \geq 0$ are embeddings of $x < 3$ and $x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$, respectively. Functions and theorems, such as `compute_poly_sat_rat` and `poly_sat_rat_correct`, that relate both theories enable the proof of formulas in the object theory by evaluating terms in the target theory, e.g., Step 2 in Example 3.

Computational reflection is particularly well-adapted for the development of automated proof strategies in interactive theorem provers. First, it produces proofs whose size is independent of the size of the initial sequent. For instance, the method in Example 3 can be used for discharging inequalities on any univariate polynomial of any degree and any number of monomials. Second, proofs by computational reflection are small since the most involved logical steps are done once for all in the proof of theorems such as `poly_sat_rat_correct`. Finally, proofs by computational reflection are efficient since they depend on computation rather than on deduction.

A PVS *strategy* is a procedure that, when executed by the theorem prover's Lisp engine, produces a PVS proof, i.e., a tree of proof commands. The PVS strategy language includes combinators for sequencing, branching, and backtracking of proof commands. The language also provides mechanisms to inspect the internal representation of PVS syntactic elements within a proof context. For instance, in the PVS specification language, the expression $x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$ is just a Boolean expression. In the strategy language, it is possible to parse the internal Lisp representation of this expression and determine that it is a real order relation whose left hand side is a univariate rational polynomial of degree 120 over variable x with monomials x^{120}, $-\frac{2}{3}x^{60}$, and $\frac{1}{9}$. PVS strategies, as tactics in the LCF-style of interactive theorem provers, preserve the logical consistency of the proof system. That is, any strategy within a proof can be unfolded in a tree of proof commands that only includes basic PVS proof rules.

The development presented in this paper includes a PVS strategy called `sturm` that implements an enhanced version of the method described in Example 3. Using this strategy, the initial sequent in this example is automatically discharged in less than one second. More generally, the strategy `sturm` automatically discharges sequents having one of the following forms

1. $X_1, \ldots, X_m, \Gamma \vdash \underline{p_1(x) \diamond p_2(x)}, \Delta$
2. $X_1, \ldots, X_m, \underline{p_1(x) \diamond' p_2(x)}, \Gamma \vdash \Delta$
3. $\Gamma \vdash \underline{\forall(x\colon T)\colon X_1 \wedge \ldots \wedge X_m \implies p_1(x) \diamond p_2(x)}, \Delta$
4. $\underline{\forall(x\colon T)\colon X_1 \wedge \ldots \wedge X_m \implies p_1(x) \diamond p_2(x)}, \Gamma \vdash \Delta$
5. $\Gamma \vdash \underline{\exists(x\colon T)\colon X_1 \wedge \ldots \wedge X_m \wedge p_1(x) \diamond' p_2(x)}, \Delta$

6. $\underline{\exists(x\colon T)\colon X_1 \wedge \ldots \wedge X_m \wedge p_1(x) \diamond' p_2(x)}, \Gamma \vdash \Delta$

where

- T is a subtype of $\mathbb{R}$,
- Γ and Δ are arbitrary lists of formulas, which are ignored by the strategy,
- $m \geq 0$ and for $1 \leq j \leq m$, X_m denotes a Boolean expression of one of the forms $a \prec x$, $x \prec a$, $|x| \prec a$, or $x \in [a, b]$, where $\prec \in \{<, \leq\}$, and a, b are numerical rational constants,
- $p_1(x)$ and $p_2(x)$ denote univariate rational polynomial such that the polynomial $p(x) = p_1(x) - p_2(x)$ is not a constant.
- $\diamond$ is a real order in $\{<, \leq, >, \geq, \neq\}$ and $\diamond'$ is a real order in $\{<, \leq, >, \geq, =\}$.

The strategy `sturm` works on a formula of interest, which is the underlined formula in the forms above. By default, the strategy assumes that the formula of interest is the first formula in the consequent, but the user can specify a different formula though an optional parameter in the strategy. As specified above, the formula of interest can appear in the antecedent or in the consequent, be nonquantified or quantified, and the quantifier can be existential or universal.

The strategy proceeds as follows. First, it examines the sequent and determines whether or not it has one of the supported forms. If this is not the case, the strategy prints an error message and does nothing else. If the sequent is supported, the strategy parses the formula of interest and constructs a PVS data structure `p` that represents the univariate polynomial $p(x) = p_1(x) - p_2(x)$. This part of the strategy uses a function developed by B. Di Vito (NASA) for parsing PVS polynomial expressions [22]. The parse does not assume any particular polynomial normal form. In particular, real expressions such as `(x-1)^2`, `(x-1)*(x-1)`, and `x*x-2*x+1` are all parsed into the same data structure `p`. Next, the strategy computes PVS data structures for instantiating the variables $a, b,$ `cont_lb`, `cont_ub` in Theorem `poly_sat_rat_correct`. These data structures are constructed by examining the relational formulas X_j, $1 \leq j \leq m$, and, in the case of quantified forms, the type T. It is noted that the correctness of the strategy is not compromised by the constructions of these objects. Indeed, as shown in Example 3, the strategy checks that the semantics of the original expressions and their data structure embeddings coincide.

The instantiation of relation R in Theorem `poly_sat_rat_correct` depends on the form of the sequent. In the case of forms 1, 3, and 4, R is instantiated with $\diamond$. Otherwise, R is instantiated with $\neg\diamond'$. From this point on, the strategy follows the method described in Example 3. Some minor modifications are needed for the cases where the formula of interest is quantified. Sequents of the forms 3 and 6 can be transformed into sequents of the forms 1 and 2 by introducing the quantified variable (this process is called Skolemization, in PVS terminology) and propositional simplification. Sequents of the forms 4 and 5 use the fact that Theorem `poly_sat_rat_correct` is a double implication. In this case, the evaluation

of the function `compute_poly_sat_rat` returns `false` and the proof proceeds accordingly. Finally, if the sequent holds, the strategy succeeds and the sequent is discharged. If the sequent does not hold, the strategy prints a message stating that the sequent is not provable.

Example 4. *Turan's inequality for Legendre's polynomials is a theorem that states that* $p_n(x)^2 > p_{n-1}(x) \cdot p_{n+1}(x)$ *for all* x *such that* $-1 < x < 1$*, where* p_j *is the* j*-th Legendre polynomial (of degree* j*). For instance, the* 10*-th Legendre polynomial is given by* $\frac{1}{256} \cdot (46189x^{10} - 109395x^8 + 90090x^6 - 30030x^4 + 3465x^2 - 63)$*. Turan's inequality for* $n = 10$ *easily reduces to showing that*

$$\frac{3969}{65536} + \frac{63063}{4096}x^6 + \frac{1792791}{4096}x^{10} + \frac{3002285}{4096}x^{18} + \frac{6600165}{4096}x^{14} - \frac{72765}{65536}x^4 - \frac{3558555}{32768}x^8 - \frac{10207769}{65536}x^{20} - \frac{35043645}{32768}x^{12} - \frac{95851899}{65536}x^{16} > 0,$$

for all $x \in (-1, 1)$*. The strategy* `sturm` *in PVS automatically discharges this theorem in less than a second. In fact, the strategy is able to prove that the polynomial on the left hand side of this inequality is positive over this interval when it is cubed and then has degree 60, a proof that takes the decision procedure less than 10 seconds to run.*

The Appendix includes examples of PVS formulas, including Example 3 and Example 4, that are automatically discharged by the strategy `sturm`.

7 Related Work

The authors are aware of three bodies of work that are closely related to this paper in their use of Sturm's Theorem for nonlinear reasoning in a theorem prover.

The first related work was accomplished by John Harrison in the 90's [13]. He proved Sturm's Theorem in HOL and used it there for root isolation. The intended application of that work was to guarantee error bounds of polynomial approximations to transcendental functions. Root isolation was used on the derivative of the error to find the places where an error may be maximized. In that work, Sturm's Theorem was proved in the case where the polynomial is square free, which simplifies the proof. In that paper, Harrison writes "[...] we would prefer the polynomial to have no multiple real roots [...] Sturm's Theorem is easier to prove for polynomials without multiple real roots - this is actually the only form we have proved in HOL." Harrison was not interested in proving statements like $p(x) \geq 0$, so a decision procedure for such problems was out of the scope of his paper.

The recent work by Eberl [8] is the most similar to this paper. It was apparently being completed concurrently with this work. Eberl completed a formal proof in Isabelle/HOL of Sturm's Theorem and used that to define a proof method `sturm` in Isabelle/HOL for solving simply quantified polynomial inequalities similar to those solved by PVS's strategy `sturm`. In the case of non-strict universally-quantified inequalities, Eberl's proof method relies on unverified ML code to generate the interval splitting as a witness. In practice, the use of ML code improves efficiency

and does not compromises soundness. However, it may compromise completeness. In contrast, the PVS's strategy `sturm` relies on a PVS algorithm that is proven to be correct and complete. Another distinction between the current paper and the work of Eberl is in the use of pseudo division instead of regular division. PVS handles A final point worth noting regarding the differences between the work in this paper and that by Eberl is that the proofs of Sturm's Theorem are different. Eberl proves Sturm's Theorem in the square free case, similar to Harrison's proof in HOL [13]. In the non-square free case, the proof proceeds by dividing each term in the remainder sequence by the greatest common divisor of the original polynomial and its derivative. The resulting sequence is not a Sturm sequence in the standard sense, but it maintains similar properties regarding root counting. The proof in the PVS development presented in this paper is a direct approach that considers the highest power of a linear divisor that divides each polynomial in the sequence and analyzes whether the polynomial swaps signs at the corresponding root. The distinctions between these proof methods primarily amount to user preference.

Sturm sequences and several speed enhancements such as the use of pseudo division are implemented in the SMT solver Z3 [6]. That implementation was the inspiration for the work presented in this paper.[2] Z3 is a highly efficient tool. In some cases, for example when a formula is satisfiable, Z3 can produce models, which can be understood as proof certificates for existential formulas. However, in general, Z3 statements are not supported by formal proofs. Hence, in formal verifications efforts, Z3 is used as a trusted oracle.

Z3 is used as an external algebraic decision method (EADM) in Metitarski, a theorem prover for real numbers [1]. In a recent work, Denman and Muñoz [7] developed the PVS proof rule `metit` that integrates Metitarski as an external oracle into PVS theorem prover. In contrast to the work presented in this paper, `metit` is not implemented as a proof producing strategy. Nevertheless, the integration of Metitarski/Z3 in PVS, while unproven, is quite useful and has helped the authors to check results that were impossible with previous strategies in PVS.

There is a much larger collection of works on the general problem of reasoning about nonlinear arithmetic. Sophisticated implementations of the *Cylindrical Algebraic Decomposition* (CAD) [2] procedure are available in the Redlog system[3] and in the QEPCAD library.[4] The systems RealPaver [11] and dReal [9] integrate powerful methods based on interval constraint propagation [11]. MetiTarski [1] and RAHD (Real Algebra in High Dimensions) [25] 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.

Proof tactics that implement Hörmander's quantifier elimination method are available in Coq and HOL Light [18, 19]. These tactics are theoretically interesting.

[2] The authors would like to give credit to Leonardo de Moura for directing their work in this direction and advising them in this effort.

[3] `http://redlog.dolzmann.de`.

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

Hörmander's method is known to be more inefficient than CAD. An implementation of CAD that will eventually yield a proof producing tactic is available in Coq [17]. Another approach to solve multivariate polynomial inequalities in theorem provers is based on polynomial sum of square (SOS) decompositions through semidefinite programming. Such an approach has been implemented in HOL Light [14] and seems to be more promising than quantifier elimination for polynomials with many variables. Semidefinite programming is a somewhat complicated numerical procedure that is usually implemented with floating point numbers. Because of numerical approximation errors, it is difficult to integrate this method into theorem provers. Recent developments in SOS address this issue by producing rational polynomial decompositions [15, 21]. Proof producing strategies for proving real-number properties based on interval arithmetic and branch and bound methods are available in PVS [4], Coq [20], HOL Light [26]. As stated in the introduction, the authors have developed semi-decision procedures for multivariate polynomials, based on Bernstein polynomials [22], and Boolean expressions involving real-valued functions, based on interval arithmetic [23]. Those algorithms are quite powerful and can prove tight bounds on complex polynomials with up to 16 variables and degree 4. However, since Bernstein polynomials and interval arithmetic yield numerical approximation methods, they cannot always prove exact bounds on a polynomial. That is, in general, they cannot prove that $p(x) \geq 0$ for x in a fixed, bounded interval unless it is also true that $p(x) > c$ for some positive c. This limitation is key to the authors' development of Sturm's Theorem in PVS.

8 Conclusion

This paper presented a formalization Sturm's Theorem in PVS along with a decision procedure for deciding the sign of univariate rational polynomials where the polynomial variable ranges over an interval, which can be any connected set of real numbers. The decision procedure, which is is shown to be complete and correct in PVS, is used to implement a proof strategy for automatically discharging sequents involving univariate polynomial inequalities. The correctness of this strategy only depends on the correctness of PVS deduction engine. The strategy is based on computational reflection, which is a theorem proving technique for building efficient strategies. Although the strategy uses concrete data structures for representing polynomials and infinite numbers, these data structures are invisible to the user. The strategy can be used to discharge sequents involving *native* PVS real number expressions.

In its standard form, Sturm's Theorem is only applied when the end points of the interval are not multiple roots and the polynomial inequality is strict. The authors address these limitations by using a branching algorithm that progressively subdivides the interval, noting that a polynomial with at most one root in a given interval is nonnegative on that interval if and only if it is nonnegative at both endpoints. Furthermore, the PVS development presented in this paper uses the remainder of polynomials after pseudo division rather than standard division. This technique ensures that integer polynomials beget more integer polynomials. Moreover, be-

fore working with a polynomial, the PVS decision procedure first divides it by the greatest common divisor of its coefficients. These features substantially improve the efficiency of the decision procedure.

The authors's main motivation for developing Sturm's Theorem in PVS comes from NASA's verification effort on aircraft separation assurance systems.[5] The PVS proofs of correctness of these system are nontrivial and require considerable algebraic manipulations. In one particular case of an actual verification effort, a 176-line sequent involving a 16-variable polynomial was generated. That sequent was automatically checked using a PVS proof rule that integrates Metitarski and Z3 as trusted external oracles into the PVS theorem prover [7]. This example shows the usefulness of techniques based on Sturm sequences, which are implemented in Z3. The ultimate goal of the work presented here is to build a similar algorithm to that used in Z3, directly in PVS with a formal proof of its correctness. The work with univariate polynomials presented in this paper is a first step in that direction. The next step is to define an algorithm that not only handles multivariate polynomials, but also handles arbitrary Boolean expressions involving those polynomials. This work will be guided by the algorithm developed by de Moura in Z3, but the authors expect subtleties to arise, since it will have to be designed with its formal verification in mind. That is, having to prove its correctness may change the way that the algorithm is defined in PVS.

References

1. Behzad Akbarpour and Lawrance C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.

2. George Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Kaiserslautern, 1975. Springer-Verlag.

3. Luis G. Crespo, César A. Muñoz, Anthony J. Narkawicz, Sean P. Kenny, and Daniel P. Giesy. Uncertainty analysis via failure domain characterization: Polynomial requirement functions. In *Proceedings of European Safety and Reliability Conference*, Troyes, France, September 2011.

4. Marc Daumas, David Lester, and César Muñoz. Verified real number calculations: A library for interval arithmetic. *IEEE Transactions on Computers*, 58(2):1–12, February 2009.

5. Florent de Dinechin, Christoph Lauter, and Guillaume Melquiond. Certifying the floating-point implementation of an elementary function using Gappa. *IEEE Transactions on Computers*, 60(2):242–253, February 2011.

[5] http://shemesh.larc.nasa.gov/fm/fm-atm-cdr.html

6. Leonardo de Moura and Grant Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In *Automated Deduction - CADE-24, 24th International Conference on Automated Deduction, Lake Placid, New York, June 9-14, 2013, Proceedings*, 2013.

7. William Denman and César Muñoz. Automated real proving in PVS via metitarski, 2014. Accepted for publication at 19th International Symposium on Formal Methods (FM 2014).

8. Manuel Eberl. A formal proof of Sturm's theorem in Isabelle/HOL. Manuscript. Available from `http://afp.sf.net/entries/Sturm_Sequences.shtml`, 2014.

9. Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer, 2013.

10. J. Garloff. Application of Bernstein expansion to the solution of control problems. *Reliable Computing*, 6:303–320, 2000.

11. Laurent Granvilliers and Frédéric Benhamou. RealPaver: An interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software*, 32(1):138–156, March 2006.

12. John Harrison. Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge, Millers Yard, Cambridge, UK, 1995.

13. John Harrison. Verifying the accuracy of polynomial approximations in HOL. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: 10th International Conference, TPHOLs'97*, volume 1275 of *Lecture Notes in Computer Science*, pages 137–152, Murray Hill, NJ, 1997. Springer-Verlag.

14. John Harrison. Verifying nonlinear real formulas via sums of squares. In *Theorem Proving in Higher Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages 102–118. Springer, 2007.

15. Erich L. Kaltofen, Bin Li, Zhengfeng Yang, and Lihong Zhi. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. In Lorenzo Robbiano and John Abbott, editors, *Approximate Commutative Algebra*, Texts and Monographs in Symbolic Computation. Springer Vienna, 2010.

16. James Kuchar and Lee Yang. A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):179–189, December 2000.

17. Assia Mahboubi. Implementing the cylindrical algebraic decomposition within the Coq system. *Mathematical Structures in Computer Science*, 17(1):99–127, February 2007.

18. Assia Mahboubi and Loïc Pottier. Elimination des quantificateurs sur les réels en Coq. In *Journées Francophone des Langages Applicatifs (JFLA)*, 2002.

19. Sean McLaughlin and John Harrison. A proof-producing decision procedure for real arithmetic. In Robert Nieuwenhuis, editor, *Proceedings of the 20th International Conference on Automated Deduction, proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 295–314, 2005.

20. Guillaume Melquiond. Proving bounds on real-valued functions with computations. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2008.

21. David Monniaux and Pierre Corbineau. On the generation of Positivstellensatz witnesses in degenerate cases. In *Proceedings of Interactive Theorem Proving (ITP)*. Lecture Notes in Computer Science, 2011.

22. César Muñoz and Anthony Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2):151–196, August 2013.

23. Anthony Narkawicz and César Muñoz. A formally verified generic branching algorithm for global optimization. In Ernie Cohen and Andrey Rybalchenko, editors, *Fifth Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE 2013)*, volume 8164 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

24. Sam Owre, John Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceeding of the 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752. Springer, June 1992.

25. Grant Olney Passmore and Paul B. Jackson. Combined decision techniques for the existential theory of the reals. In L. Dixon, editor, *Proceedings of Calculemus/Mathematical Knowledge Managment*, number 5625 in LNAI, pages 122–137. Springer-Verlag, 2009.

26. Alexey Solovyev and Thomas C. Hales. Formal verification of nonlinear inequalities with Taylor interval approximations. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *Proceedings of the 5th International Symposium NASA Formal Methods*, volume 7871 of *Lecture Notes in Computer Science*, pages 383–397, 2013.

27. Frank Sottile. Chapter 2: Real solutions to univariate polynomials. Course Notes.

28. C. Sturm. Mémoire sur la résolution des équations numériques. In Jean-Claude Pont, editor, *Collected Works of Charles François Sturm*, pages 345–390. Birkhuser Basel, 2009.

Appendix A

PVS Examples of Strategy `sturm`

```
examples : THEORY
BEGIN
  IMPORTING Sturm@strategies
  x : VAR real

  Example_0 : LEMMA x*(1-x) <= 1/4
%|- Example_0 : PROOF (sturm) QED

  Example_1 : LEMMA
    0 < x AND x < 3 IMPLIES x^120 - 2*x^60 + 1 >= 0
%|- Example_1 : PROOF (sturm) QED

  Example_2_3 : LEMMA
    x < 3 IMPLIES x^120 - (2/3)*x^60 + 1/9 >= 0
%|- Example_2_3 : PROOF (sturm) QED

  Example_4: LEMMA
    abs(x) < 1 IMPLIES
       3969/65536 + 63063/4096 * x^6 + 1792791/4096 * x^10 +
       3002285/4096 * x^18 + 6600165/4096 * x^14
       - 72765/65536 * x^4 - 3558555/32768 * x^8
       - 10207769/65536 * x^20 - 35043645/32768 * x^12
       - 95851899/65536 * x^16 > 0
%|- Example_4 : PROOF (sturm) QED

  Example_5: LEMMA
    abs(x) < 1 IMPLIES
      (3969/65536 + 63063/4096 * x^6 + 1792791/4096 * x^10 +
       3002285/4096 * x^18 + 6600165/4096 * x^14
       - 72765/65536 * x^4 - 3558555/32768 * x^8
       - 10207769/65536 * x^20 - 35043645/32768 * x^12
       - 95851899/65536 * x^16)^3 > 0
%|- Example_5 : PROOF (sturm) QED

  Example_6: LEMMA
    EXISTS (x) : -1 <= x AND x <= 0 AND
      32*x^5 - 160*x^3 + 120*x > 16*x^4 - 48*x^2 +12
%|- Example_6 : PROOF (sturm) QED

END examples
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01- 11 - 2014	Technical Memorandum	

4. TITLE AND SUBTITLE

A Formally-Verified Decision Procedure for Univariate Polynomial Computation Based on Sturm's Theorem

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Narkawicz, Anthony J.; Munoz, Cesar A.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

534723.02.15.07

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L-20489

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA/TM-2014-218548

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 64
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Sturm's Theorem is a well-known result in real algebraic geometry that provides a function that computes the number of roots of a univariate polynomial in a semiopen interval. This paper presents a formalization of this theorem in the PVS theorem prover, as well as a decision procedure that checks whether a polynomial is always positive, nonnegative, nonzero, negative, or nonpositive on any input interval. The soundness and completeness of the decision procedure is proven in PVS. The procedure and its correctness properties enable the implementation of a PVS strategy for automatically proving existential and universal univariate polynomial inequalities. Since the decision procedure is formally verified in PVS, the soundness of the strategy depends solely on the internal logic of PVS rather than on an external oracle. The procedure itself uses a combination of Sturm's Theorem, an interval bisection procedure, and the fact that a polynomial with exactly one root in a bounded interval is always nonnegative on that interval if and only if it is nonnegative at both endpoints.

15. SUBJECT TERMS

Polynomial inequalities; Prototype verification system; Sturm's theorem; non-linear arithmetic decision procedure

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	32	(443) 757-5802