A Decision Procedure for Univariate Polynomial Systems Based on Root Counting and Interval Subdivision

ANTHONY NARKAWICZ
CESAR MUNOZ
and
AARON DUTLE
NASA Langley Research Center, Hampton, VA 23681-2199

This paper presents a formally verified decision procedure for determinining the satisfiability of a system of univariate polynomial relations over the real line. The procedure combines a root counting function, based on Sturm's theorem, with an interval subdivision algorithm. Given a system of polynomial relations over the same variable, the decision procedure progressively subdivides the real interval into smaller intervals. The subdivision continues until the satisfiability of the system can be determined on each subinterval using Sturm's theorem on a subset of the system's polynomials. The decision procedure has been formally verified in the Prototype Verification System (PVS). In PVS, the decision procedure is specified as a computable Boolean function on a deep embedding of polynomial relations. This function is used to define a proof producing strategy for automatically proving existential and universal statements on polynomial systems. The soundness of the strategy solely depends on the internal logic of PVS.

1. INTRODUCTION

Proving high-level properties of safety-critical systems that interact with the physical environment often involves reasoning about real-valued functions. At NASA, for example, the need for automated proving techniques for these kinds of properties arises in air traffic management systems [19,20,23], floating point analysis [18], uncertainty and reliability analysis of dynamics and control systems [4], and many others applications. In the last few years, much work has been done in theorem proving to automate the proof of properties involving real-valued functions [1,5,7,16,26], in general, and polynomial functions [6,8–11,13–15,17,21,22,26], in particular.

In previous work [24], the authors formally verified two decision procedures by Basu et al. [2] for the decidability of univariate polynomial relations. These decision procedures are based on Tarski's theorem. In its basic form, Tarski's theorem is called Sturm's theorem and can be used to define a computable function that explicitly computes the cardinality

$$\mathrm{card}(\{x \in [a,b] : \, p(x) = 0\}), \tag{1}$$

where p is a univariate polynomial and a and b are in the set of real numbers extended with infinity values. Sturm's theorem is used to define a decision procedure for determining whether a single polynomial relation $p(x)\,R\,0$, with $R \in \{=, >, <, \neq, \geq, \leq\}$, is satisfiable over the possibly unbounded interval $[a,b]$. This decision procedure counts the number of roots in intervals and subdivides the real line until each subinterval has at most one root, at which point satisfiability of the relation can be determined. In [24], the authors also formally proved a much more complex version of Tarski's theorem that gives a linear matrix relationship between cardinalities of solution sets of systems involving a finite number of polynomials and computable *Tarski queries*. These queries are at the basis of a decision procedure for determining satisfiability of systems of polynomial relations. Both decision procedures are formally specified and verified in the Prototype Verification System (PVS) [25].

This paper presents another formally verified decision procedure for determining the satisfiability of a system of univariate polynomial relations that uses a simpler and easier to verify subdivision procedure. Given a system of univariate polynomials on an interval, Sturm's theorem is used to determine a) whether each of the polynomials has at most one root in the interval and b) whether all the polynomials with one root in that interval have a root at the same location. This latter check is handled by counting the number of roots in the interval of either the product or sum of squares of all polynomials that have exactly one root in the interval. If this method determines that all of the polynomials with one root have the same root, the satisfiability of the system can be completely determined over that interval. The decision method works by continually subdividing the real line until each such subinterval can be decided in this way. Thus, the algorithm in this paper, based on Sturm's theorem, is significantly simpler to verify than other methods based on Tarski's theorem that require reasoning about a large linear matrix equation.

The decision procedure in this paper is defined as a computable function in the PVS specification language and the correctness and completness properties of the decision procedure are specified as theorems about the inputs and output of this function. These theorems are at the basis of a proof producing *strategy* in PVS. This strategy automatically proves satisfiability or validity of univariate systems of polynomials. The core step in the strategy is the invocation of the correctness theorems of the fully specified and verified decision procedure. Hence, no external tools are needed to use the procedure in actual proofs of polynomial relations in PVS.

The rest of this paper is organized as follows. Sturm's theorem is presented in Section 2. The proposed decision procedure for univariate polynomial systems is presented in Section 3. The proof producing strategy `hutch` is defined in Section 4. Section 5 presents a comparison of this strategy to similar proof producing strategies on a set of benchmarks, which are listed in Appendix A. Section 6 discusses related work. Finally, concluding remarks are presented in Section 7.

All theorems presented in this paper are formally verified in PVS. For readability, standard mathematical notation is used throughout the paper. The PVS formal development presented in this paper is electronically available in the `Tarski@hutch` theory of the NASA PVS Library.[1] Appendix B includes a table that maps the theorems presented in this paper to their formalization in the NASA PVS Library.

[1]http://github.com/nasa/pvslib.

2. STURM'S THEOREM

This section briefly presents a formalization of Sturm's theorem in PVS. It necessarily repeats some of the development from the authors' previous work [24, 27]. After stating Sturm's theorem, this section describes the function `roots_in_int`, which is used to explicitly count the number of roots of a polynomial in a closed interval.

Many functions in PVS have a polynomial or a system of polynomials as an input. In PVS, a polynomial is formalized with an infinite array $p\colon \texttt{nat} \to T$, where `nat` is the PVS type of nonnegative integers and T is a subtype of the real numbers, along with a natural number at which the array p is nonzero (the degree). Thus, all PVS functions acting on a polynomial take both the array and the degree as inputs.

In this paper, to save notation and make functions easier to read, a polynomial is defined as a nonempty list p of elements of T whose last element is nonzero. The i-th element of the list represents the i-th coefficient of the polynomial and if n is the length of p, then $n-1$ is the *degree* of p. For instance, the polynomial $1-3x^2$ is represented in PVS by the list $(1, 0, -3)$, whose length is 3. The type T can be instantiated with any subtype of the real numbers. If the elements of T are all either integers or rationals, then p is called an *integer polynomial* or a *rational polynomial*, respectively. The function `eval` converts a list of coefficients to a *polynomial function*:

$$\texttt{eval}(p)(x) \equiv \sum_{i=0}^{n} c_i x^i, \tag{2}$$

where x is a real number, $n+1$ is the *length* of the list p, $c_n \neq 0$, if $n > 0$, and c_i is the i-th element of p, for $0 \le i \le n$. If p and q are any two polynomials, then there exists a polynomial $p \cdot q$ such that $\texttt{eval}(p \cdot q)(x) = \texttt{eval}(p)(x) \cdot \texttt{eval}(q)(x)$ for all $x \in \mathbb{R}$. The list $p \cdot q$ is called the *product* of p and q, and its formal definition is given in the PVS development accompanying this paper. If $\boldsymbol{p} = p_0, \ldots, p_k$ is a sequence of polynomials, then $\prod_i p_i$ denotes the product of all polynomials in $\boldsymbol{p}$. Furthemore, given any two univariate polynomials g and h such that h is nonzero, there are unique polynomials q and r such that r has degree strictly less than h and $\texttt{eval}(g)(x) = \texttt{eval}(q)(x) \cdot \texttt{eval}(h)(x) + \texttt{eval}(r)(x)$ holds for all x. Let $\operatorname{rem}(g, h)$ denote the *remainder* polynomial r in this expression. Henceforth, the notation p' represents the derivative polynomial of p, i.e., if p is represented by the list of coefficients $(c_0, \ldots, c_n)$, p' is represented by the list of coefficients $(c'_0, \ldots, c'_{n-1})$, where $c'_i = c_{i+1} \cdot (i+1)$.

Given a univariate polynomial p, a *Sturm chain* of p is a sequence S of polynomials

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

where

$$\begin{aligned} p_0 &= p, \\ p_1 &= p', \\ \forall\, d > 1\, \exists\, c > 0 \ : p_d &= -c \cdot \operatorname{rem}(p_{d-2}, p_{d-1}), \\ p_m &= 0, \text{and} \\ p_{m-1} &\neq 0. \end{aligned} \tag{4}$$

Evaluating each of the polynomials in a Sturm sequence at some $x \in \mathbb{R}^\infty$ produces a sequence of extended real numbers $S(x)$. A function σ_p is defined on $\mathbb{R}^\infty$ by setting $\sigma_p(x)$ to be equal to the number of sign changes in $S(x)$ after all zeros have been removed from the sequence, where the number of sign changes in a sequence of nonzero real numbers is defined as follows. Let $A = (a_0, a_1, \ldots, a_k)$ be a finite sequence of nonzero extended real numbers. The number of sign changes in A is defined as the number of indices i such that a_i and a_{i+1} have different signs.

Sturm's theorem is stated in Theorem 2.1 below. It has been formally proven in PVS, and that formalization is more thoroughly described in the authors' previous papers [24, 27].

Theorem 2.1. *Let p be a univariate polynomial. For $a, b \in \mathbb{R}^\infty$, with $a < b$, if neither a nor b is a root of both p and p', then $\sigma_p(a) - \sigma_p(b)$ is equal to*

$$\mathrm{card}(\{x \in (a, b] : \texttt{eval}(p)(x) = 0\}).$$

Sturm's theorem is used to define a function `roots_in_int` that explicitly computes the number of roots of a polynomial with integer coefficients inside any closed, bounded interval. This function is a key component of the decision procedure presented in later sections for satisfiability of polynomial systems. The function `roots_in_int` has as inputs real numbers a and b as inputs (with $a < b$), an integer polynomial p, and a list ℓ of polynomials. If ℓ is a Sturm chain of p, then the function returns the number of roots in the closed interval $[a, b]$, not counting multiplicities. This means, for instance, that for the polynomial p with `eval`$(p)(x) = x^3$, it will count exactly one root on the interval $[-1, 1]$. In practice ℓ is set to a Sturm chain of p denoted `sturm_chain`(p). It should be noted that the function `roots_in_int` is not a direct application of Sturm's theorem, because the theorem itself requires that neither a nor b is a root of both p and p'. In the case that either a or b is a root of both polynomials, the function `roots_in_int` applies Sturm's theorem on a slightly larger interval that is guaranteed to have the same number of roots as $[a, b]$. The following theorem, which also appears in earlier work [24], states the correctness of the function `roots_in_int`.

Theorem 2.2. *Let $a, b \in \mathbb{R}^\infty$, with $a < b$, p be an integer polynomial, and $S = \{r \in \mathbb{R} \mid a \leq r \leq b$ and $\texttt{eval}(p)(r) = 0\}$. It holds that*

$$\mathrm{card}(S) = \texttt{roots\_in\_int}(p, a, b, \texttt{sturm\_chain}(p)).$$

One important point about Theorem 2.2 is that the sequence `sturm_chain`(p) does not depend on the interval $[a, b]$ and therefore can be used on different intervals. Indeed, the decision procedure for satisfiability of polynomial systems calls `roots_in_int` many times with the same input polynomial but on different intervals. When the Sturm sequence `sturm_chain`(p) is precomputed before any of these calls and input into each call, computation time of the decision procedure is significantly reduced.

To avoid costly computations with rational coefficients in the remainder sequence, the formal definition of the Sturm sequence `sturm_chain`(p) uses an algorithm called *pseudo division* instead of the standard polynomial division algorithm. This division method does not involve division by coefficients of the polynomials, which means that if the coefficients of the original polynomials are integers, then the coefficients of their remainder after pseudo division will also be integers. The motivation for this is that the coefficients of the polynomials in the standard remainder sequence become quite complex as successive remainders are calculated. Computations involving these rationals with many digits in both their numerators and denominators are less efficient than computations on large integers, making the pseudo-remainder more efficient than the standard remainder. The formalization of the pseudo division algorithm is given in previous work [24].

3. DECISION PROCEDURE BASED ON INTERVAL SUBDIVISION

This section presents a decision procedure for determining the satisfiability of a system of polynomials with rational coefficients over the real line. The development of this decision procedure depends on other decision procedures of incremental generality. First, the case of systems of polynomials with integer coefficients is discussed. For these kinds of polynomial systems, the algorithm `decide_interval` is presented in Section 3.1 for deciding satisfiability on small bounded intervals. The definition of small intervals is presented in Section 3.2. In Section 3.3, the algorithm `decide_interval` is used to define `hutch_int_basic`, which decides satisfiability on any bounded interval using an interval subdivision technique and the root counting function presented in Section 2. The algorithm `hutch_int_basic` is then extended by the algorithm `hutch_int` in Section 3.4 to decide satisfiability of systems of integer polynomials over the real line. Finally, the case of systems of polynomials with rational coefficients is considered. The algorithm `hutch` presented in Section 3.5 uses `hutch_int` to decide satisfiability of systems of rational polynomials over the real line.

A system of polynomials is represented by a tuple $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$, where

—$\boldsymbol{p}$ is a sequence of $k+1$ polynomials $p_0, \ldots, p_k$ with rational coefficients,

—$\boldsymbol{r}$ is a sequence of $k+1$ relations $r_0, \ldots, r_k$ from the set $\{=, >, <, \neq, \geq, \leq\}$, and

—$\boldsymbol{Q}$ is a function that takes a $k+1$-tuple of Booleans as input and returns a Boolean.

Just as the PVS development models a polynomial as an infinite array along with a natural number (the degree), it models the sequence $\boldsymbol{p}$ of polynomials as an infinite array $\texttt{nat} \to [\texttt{nat} \to \mathbb{Q}]$ of coefficients along with an infinite array $\texttt{nat} \to \texttt{nat}$ of degrees. Each of these infinite arrays is an input to each PVS function acting on a system of polynomials, along with the natural number k specifying the number of polynomials in the system. As in the case of a single polynomial described in Section 2, this difference between this paper and the PVS development is to save notation and make the functions easier to read.

The function $\boldsymbol{Q}$ represents the Boolean expression of polynomials in $\boldsymbol{p}$ to be checked for satisfiability, where the i-th parameter of $\boldsymbol{Q}$ is associated to the i-th polynomial in $\boldsymbol{p}$ and the i-th relation in $\boldsymbol{r}$.

The predicate `system_sat?` defines satisfiability for the polynomial system represented by $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$ over a subset A of the real numbers.

$$\begin{aligned}&\texttt{system\_sat?}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, A) \equiv \\ &\quad \exists x \in A : \boldsymbol{Q}(r_0(\texttt{eval}(p_0)(x), 0), \ldots, r_k(\texttt{eval}(p_k)(x), 0)).\end{aligned} \tag{5}$$

Example 3.1. *The satisfiability of the system*

$$\begin{aligned} (x-2)^2 \cdot (-x+4) > 0 \quad &\wedge \\ x^2 \cdot (x-3)^2 \geq 0 \quad &\wedge \\ x \geq 1 \quad &\wedge \\ -(x-3)^2 + 1 > 0 \quad &\wedge \\ -(x-(11/12))^3 \cdot (x-(41/10))^3 < 1/10 & \end{aligned}$$

is given by the truth value of $\textbf{\textit{system\_sat?}}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \mathbb{R})$*, where*

—$k = 4$,

—$\boldsymbol{p} = p_0, \ldots, p_k$ *such that* $\textbf{\textit{eval}}(p_0)(x) = (x-2)^2 \cdot (-x+4)$, $\textbf{\textit{eval}}(p_1)(x) = x^2 \cdot (x-3)^2$, $\textbf{\textit{eval}}(p_2)(x) = x-1$, $\textbf{\textit{eval}}(p_3)(x) = -(x-3)^2+1$, $\textbf{\textit{eval}}(p_4)(x) = -(x-(11/12))^3 \cdot (x-(41/10))^3 - (1/10)$,

—$\boldsymbol{r} = (>, \geq, \geq, >, <)$, *and*

—$\boldsymbol{Q}(b_0, b_1, b_2, b_3, b_4) = b_0 \wedge b_1 \wedge b_2 \wedge b_3 \wedge b_4$.

3.1 Deciding Satisfiability on Small Intervals

This section presents a function that, given a sufficiently small interval, can determine if a univariate polynomial system with integer coefficients is satisfiable on that interval. The following theorem, whose proof depends trivially on the intermediate value theorem, shows that it is possible to define such a function for certain intervals.

Theorem 3.1. *Let a, b, and c be real numbers such that $a \leq c \leq b$, and let c be the only point in $[a, b]$ at which any polynomial in the list $\boldsymbol{p}$ can have a root, i.e., $a \leq x \leq b$ and $\textbf{\textit{eval}}(p_i)(x) = 0$ implies $x = c$, for any x. A system $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$ is satisfiable on $[a, b]$ if and only if $\textbf{\textit{system\_sat?}}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \{a, b, c\})$.*

Recall from the introduction that the main decision procedure presented in this paper continually subdivides an interval until each subinterval is small enough to determine satisfiability. When the algorithm checks a particular interval and then moves on to smaller subintervals, it is often the case that information already computed on the large interval does not have to be recomputed on the smaller subintervals. For instance, if it is determined that a particular polynomial in the list $\boldsymbol{p}$ is always positive on an interval, then once the algorithm subdivides into smaller subintervals, no more computations of the sign of that polynomial have to be done. For this reason, there is a parameter to the decision procedure called $\boldsymbol{\sigma}$ that is a sequence $\sigma_0, \ldots, \sigma_k$ of signs, i.e., elements in the set $\{-1, 0, 1\}$. There is also a predicate `sound_signs?` that expresses whether this sequence correctly stores sign information for the polynomial list $\boldsymbol{p}$ on an interval $[a, b]$.

$$\begin{aligned} &\texttt{sound\_signs?}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}) \equiv \forall\, 0 \leq i \leq k\colon \\ &\qquad (\sigma_i = 0 \iff \exists\, x\colon\ a \leq x \leq b \wedge p_i(x) = 0) \wedge \\ &\qquad (\sigma_i = -1 \iff \forall\, x\colon\ a \leq x \leq b \implies p_i(x) < 0) \wedge \\ &\qquad (\sigma_i = 1 \iff \forall\, x :\ a \leq x \leq b \implies p_i(x) > 0). \end{aligned} \tag{6}$$

An important property of the predicate `sound_signs?` is that if the sequence $\boldsymbol{\sigma}$ satisfies this predicate on the interval $[a, b]$, then for any subinterval of $[a, b]$, the entries of $\boldsymbol{\sigma}$ that are equal to 0 can be updated so that the new sequence satisfies `sound_signs?` on the subinterval. This is accomplished through the function `signs_upd`, which has as inputs a list $\boldsymbol{p}$ of k polynomials, the interval bounds a and b, and a sequence σ of signs. The output of $\texttt{signs\_upd}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ is a sequence of updated signs $\sigma'_0, \ldots, \sigma'_k$. The i-th sign σ'_i is defined as follows.

```
σ'_i ≡ if $\sigma_i \neq 0$ then $\sigma_i$
       elsif roots_in_int($p_i, a, b, \mathbf{\Lambda}_i$) $\neq 0$ then $0$
       elsif $p_i(a) < 0$ then $-1$                                    (7)
       else $1$
       endif,
```

where $\mathbf{\Lambda}_i = \texttt{sturm\_chain}(p_i)$, for $0 \leq i \leq k$. In PVS, the list of Sturm sequences $\mathbf{\Lambda}_i$ is precomputed and passed as parameter to `signs_upd`. For readability, parameters representing precomputed values are left implicit in this paper.

The correctness statement of the function `signs_upd` is given by the following theorem, which has been proved in PVS.

THEOREM 3.2. *Let $a \leq a' < b' \leq b$, if $\texttt{sound\_signs?}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$, then*

$$\texttt{sound\_signs?}(k, \boldsymbol{p}, a', b', \texttt{signs\_upd}(k, \boldsymbol{p}, a', b', \boldsymbol{\sigma})).$$

In the decision procedure, the parameter $\boldsymbol{\sigma}$ will be initially computed for the polynomials so that it satisfies `sound_signs?`, and it will be updated at each subdivision step so that it it satisfies this predicate on each recursive call as well. It is important to note that if the interval $[a, b]$ is small enough that all polynomials in $\boldsymbol{p}$ with a root in this interval have exactly one root, all of which are at the same location, then the polynomial system is satisfiable at this location in the sense that $\texttt{system\_sat?}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, [a, b])$ holds, if and only if $\boldsymbol{Q}(r_0(\sigma_0, 0), \ldots, r_k(\sigma_k, 0))$ holds. This motivates the definition of the function `decide_interval`, which given an interval $[a, b]$ that contains at most one point that is a root of all polynomials in $\boldsymbol{p}$, determines whether the system is satisfiable on $[a, b]$, i.e., whether $\texttt{system\_sat?}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, [a, b])$ holds. It has a parameter `unsatl?`, which, if set to `true`, implies that the system is not satisfied at the left endpoint of the interval, in which case the computation of satisfiability at that point is not needed.

$$\begin{aligned}
&\texttt{decide\_interval}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, b, \boldsymbol{\sigma}, \texttt{unsatl?}) \equiv \\
&\qquad (\neg\texttt{unsatl?} \wedge \boldsymbol{Q}(r_0(p_0(a), 0), \ldots, r_k(p_k(a), 0))) \vee \\
&\qquad \boldsymbol{Q}(r_0(p_0(b), 0), \ldots, r_k(p_k(b), 0)) \vee \\
&\qquad \boldsymbol{Q}(r_0(\sigma_0, 0), \ldots, r_k(\sigma_k, 0)).
\end{aligned} \tag{8}$$

The correctness theorem for the function `decide_interval` is defined below in Section 3.2. For the output of this function to correctly determine satisfiability on $[a, b]$, it is required that $\texttt{sound\_signs?}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ holds.

3.2 Determining if an Interval is Sufficiently Small

The PVS functions `decidable_interval` and `decidable_interval_sq` determine if an interval is sufficiently small so that the function `decide_interval`, which is defined in Section 3.1, can be used to decide satisfiability of a system of polynomials.

Using different techniques, these two functions check whether every polynomial in $\boldsymbol{p}$ has at most one root in $[a, b]$ and, if so, whether all the polynomials share the same root.

The function `decidable_interval` considers the product $\prod_i p_i$ of all of the polynomials in $\boldsymbol{p}$ and checks whether this product, which is a polynomial, has exactly one root in the interval $[a, b]$. It is formally defined as follows.

$$\begin{aligned}&\texttt{decidable\_interval}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}) \equiv \\ &\quad (\forall\, 0 \leq i \leq k \;:\; \sigma_i = 0 \implies \texttt{roots\_in\_int}(p_i, a, b, \boldsymbol{\Lambda}_i) = 1) \wedge \\ &\quad ((\exists\, 0 \leq i \leq k \;:\; \sigma_i = 0) \implies \texttt{roots\_in\_int}(\pi, a, b, \boldsymbol{\Pi})) = 1),\end{aligned} \tag{9}$$

where $\pi = \prod_i p_i$, $\boldsymbol{\Pi} = \texttt{sturm\_chain}(\pi)$, and $\boldsymbol{\Lambda}_i = \texttt{sturm\_chain}(p_i)$, for $0 \leq i \leq k$, are all precomputed values passed as parameters to `decidable_interval` in the actual PVS definition. The soundness of the function `decidable_interval` requires that the predicate `sound_signs?`$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ holds. Formula (9) could be simplified by removing the first condition since it is implied by the second condition. However, in large systems, the first condition, which checks that each individual polynomial has at most one root in the interval, saves computation time by avoiding the costly evaluation of very large polynomials involved in the second condition. The second condition is only computed when the interval gets small enough to contain at most one root of each individual polynomial.

The function `decidable_interval_sq` considers the sum of the squares of all of the polynomials in $\boldsymbol{p}$ that have a root in $[a, b]$ and checks whether this sum, which is a polynomial, has exactly one root in the interval $[a, b]$. It is formally definited as follows.

$$\begin{aligned}&\texttt{decidable\_interval\_sq}(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}) \equiv \\ &\quad (\forall\, 0 \leq i \leq k \;:\; \sigma_i = 0 \implies \texttt{roots\_in\_int}(p_i, a, b, \boldsymbol{\Lambda}_i) = 1) \wedge \\ &\quad ((\exists\, 0 \leq i \leq k \;:\; \sigma_i = 0) \implies \texttt{roots\_in\_int}(\xi, a, b, \boldsymbol{\Xi}) = 1),\end{aligned} \tag{10}$$

where $\xi = \sum_{i:\, \sigma_i = 0} p_i^2$, $\boldsymbol{\Xi} = \texttt{sturm\_chain}(\xi)$, and $\boldsymbol{\Lambda}_i = \texttt{sturm\_chain}(p_i)$, for $0 \leq i \leq k$. In the actual PVS definition, the value of $\boldsymbol{\Lambda}_i$, for $0 \leq i \leq k$, is precomputed and passed as parameter to `decidable_interval_sq`. The values of ξ and $\boldsymbol{\Xi}$ cannot be precomputed as they depend on σ_i.

Theorem 3.3, which has been proved in PVS, states that if either of the Boolean valued functions `decidable_interval` or `decidable_interval_sq` is satisfied on a given interval for a polynomial system $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$, then satisfiability of the system can be determined by the function `decide_interval`.

THEOREM 3.3. *If* ***sound_signs?***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ *holds and either*

—***decidable_interval***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ *or*

—***decidable_interval_sq***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ *hold,*

then

$$\textit{\textbf{system\_sat?}}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, [a, b]) \iff \textit{\textbf{decide\_interval}}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, b, \boldsymbol{\sigma}).$$

Theorem 3.3 can be viewed as the statement that the decision procedure presented in this paper is correct over small intervals. The following theorems, which are proved in PVS, state that any interval of a sufficiently small size satisfies the predicates `decidable_interval` and `decidable_interval_sq`.

THEOREM 3.4. *There is a positive $\epsilon > 0$ such that for all reals a and b with $0 < b - a < \epsilon$ and* ***sound_signs?***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$*,* ***decidable_interval***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ *holds.*

THEOREM 3.5. *There is a positive $\epsilon > 0$ such that for all reals a and b with $0 < b - a < \epsilon$ and* ***sound_signs?***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$*,* ***decidable_interval_sq***$(k, \boldsymbol{p}, a, b, \boldsymbol{\sigma})$ *holds.*

Only one of the functions `decidable_interval` or `decidable_interval_sq` needs to be used at every recursive call, because these two functions compute the same information and are computationally expensive. Therefore, the actual PVS definition of the decision procedure for satisfiability has an additional Boolean parameter that enables the selection of either one of these functions.

The functions `decidable_interval` or `decidable_interval_sq` are by no means the only possible functions that could be used to determine if all polynomials with a single root have their roots at the same location. In fact, many other methods are possible to solve this problem that could potentially improve the computational complexity of the decision procedure. Another method would be to create a list of all polynomials with a single root in the interval and then to count the number of roots of each product of two successive polynomials in this list. However, this method would compute far more Sturm sequences than both `decidable_interval` and `decidable_interval_sq`, although in the best case, some of those Sturm sequences would be for polynomials of lesser degree than those considered in computing `decidable_interval_sq`. In general, the function `decidable_interval` has a distinct advantage in that the corresponding Sturm sequence only has to be computed once at the top level of the algorithm and then never again. Nevertheless, it is a Sturm sequence of a polynomial with high degree. Similarly, the function `decidable_interval_sq` has a distinct advantage in that it only has to compute one Sturm sequence at each subdivision of the decision procedure, and the degree of the corresponding polynomial is no more than twice the maximum degree of the polynomials in the system. It is possible that other methods exist that could combine the advantages of `decidable_interval` and `decidable_interval_sq` into a quick method that computes a minimum number of polynomial remainders.

3.3 Decision Procedure for Integer Polynomials on Bounded Intervals

This section presents the decision procedure for satisfiability of systems of polynomials with integer coefficients over a closed bounded interval $[a, b]$. The function `hutch_int_basic`, defined below, checks whether `system_sat?`$(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, [a, b])$

holds.

```
hutch_int_basic($k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, b, \boldsymbol{\sigma}$, unsatl?, sos?) $\equiv$
  if ($\neg$sos? $\wedge$ decidable_interval($k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}$)) $\vee$
     (sos? $\wedge$ decidable_interval_sq($k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}$))
  then decide_interval($k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, b, \boldsymbol{\sigma}$, unsatl?)
  else let $c = (a+b)/2$ in                                                          (11)
    hutch_int_basic($k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, c$, signs_upd($k, \boldsymbol{p}, a, c, \boldsymbol{\sigma}$), unsatl?, sos?) $\vee$
    hutch_int_basic($k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, c, b$, signs_upd($k, \boldsymbol{p}, c, b, \boldsymbol{\sigma}$), true, sos?)
  endif.
```

The function hutch_int_basic has an optional boolean flag sos? as an input that specifies the whether decidable_interval_sq or decidable_interval should be used to determine if an interval is sufficiently small to determine satisfiability using decide_interval. The input sos? can be set to either true or false in the PVS development without affecting the the correctness of any of the functions or theorems.

The function hutch_int_basic is recursive. In PVS, defining a recursive function requires proving that the function terminates for every input. This is accomplished by supplying a *measure* function on the inputs of the function that returns an element of a set with a well-founded relation. Proving termination entails proving that the measure function strictly decreases at every recursive call. In the case of hutch_int_basic, the measure function returns the least natural number d such that for all a', b' with $0 < b' - a' \leq (b-a)/2^d$ and every $\boldsymbol{\sigma}$ such that sound_signs?($k, \boldsymbol{p}, a', b', \boldsymbol{\sigma}$) holds, either decidable_interval($k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}$) or decidable_interval_sq($k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}$) holds. Theorem 3.4 and Theorem 3.5 are used to show that the measure function is well-defined, i.e., such a natural number d always exists. Proving that it strictly decreases at every recursive call is straightforward.

The correctness property hutch_int_basic is stated in Theorem 3.6. In the PVS development, this correctness property is stated in the return type of the function hutch_int_basic rather than as an explicit theorem.

THEOREM 3.6. *Let $\boldsymbol{p} = p_0, \ldots, p_k$ be a sequence of polynomials with integer coefficients such that sound_signs?($k, \boldsymbol{p}, a, b, \boldsymbol{\sigma}$) holds. Suppose that unsatl? implies $\neg\boldsymbol{Q}(r_0(\mathit{eval}(p_0)(a), 0), \ldots, r_k(\mathit{eval}(p_k)(a), 0))$. Then*

$$\textit{hutch\_int\_basic}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, a, b, \boldsymbol{\sigma}, \textit{unsatl?}, \textit{sos?}) = \textit{true} \tag{12}$$

if and only if the system represented by $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$ is satisfiable over the interval $[a, b]$, i.e., system_sat?($k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, [a, b]$) holds.

It is important to note the following about the above implementation of the function hutch_int_basic. Other parameters added to the PVS definition of hutch_int_basic are the precomputed values $\pi = \prod_i p_i$, $\boldsymbol{\Pi} = \texttt{sturm\_chain}(\pi)$, and the list of Sturm sequences $\boldsymbol{\Lambda}_i = \texttt{sturm\_chain}(p_i)$, for $0 \leq i \leq k$. Since the function hutch_int_basic is defined recursively, computing these values at every recursive call would significantly impact computation time. Instead, these values are computed *before* the function `hutch_int_basic` is called and passed as parameters to the functions that need them, and therefore they are computed only once each.

3.4 Decision Procedure for Integer Polynomials on the Real Line

The function `hutch_int`, defined in this section, computes satisfiability of a system of polynomials with integer coefficients over $\mathbb{R}$. It uses `hutch_int_basic` on a closed bounded interval that is guaranteed to strictly contain all points that are roots of some polynomial in the list $\boldsymbol{p}$ of integer polynomials. While there are many possible definitions of a bound on these roots, the formalization presented in this paper uses Knuth's bound [6, 24]. The reader is referred to the development `reals` in the NASA PVS library, which defines Knuth's bound in a function called `Knuth_poly_root_strict_bound`.

THEOREM 3.7. *For any nonzero real polynomial p, any root of p lies in the open interval* $(-M, M)$, *where* $M = \textit{Knuth\_poly\_root\_strict\_bound}(p)$.

The function `hutch_int` is formally defined as follows.

$$\begin{aligned}
&\texttt{hutch\_int}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \texttt{sos?}) \equiv\\
&\quad \texttt{let } M = \max_{0\le i\le k}\{\texttt{Knuth\_poly\_root\_strict\_bound}(p_i)\},\\
&\qquad\quad \boldsymbol{\sigma} = \texttt{signs\_upd}(k, \boldsymbol{p}, -r, r, \mathbf{0}) \texttt{ in}\\
&\quad \texttt{hutch\_int\_basic}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, -r, r, \boldsymbol{\sigma}, \texttt{false}, \texttt{sos?}),
\end{aligned} \tag{13}$$

where $\mathbf{0}$ is the sequence $0, \ldots, 0$ of length k whose entries are all 0. In PVS, `hutch_int` also computes $\pi = \prod_i p_i$, $\mathbf{\Pi} = \texttt{sturm\_chain}(\pi)$, and the list of Sturm sequences $\mathbf{\Lambda}_i = \texttt{sturm\_chain}(p_i)$, for $0 \le i \le k$, and passes these precomputed values to `hutch_int_basic`.

The correctness statement of the algorithm `hutch_int` is the following theorem, which has been proved in PVS.

THEOREM 3.8. *Let* $\boldsymbol{p} = p_0, \ldots, p_k$ *be a sequence of polynomials with integer coefficients and positive degree, then*

$$\textit{hutch\_int}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \textit{sos?}) = \textit{true} \tag{14}$$

if and only if the system represented by $(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q})$ *is satisfiable over the real line, i.e.,* $\textit{system\_sat?}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \mathbb{R})$ *holds.*

3.5 Decision Procedure for Rational Polynomials on the Real Line

This section presents a decision procedure called `hutch` that determines the satisfiability of a system of polynomials over the real line. The algorithm `hutch` computes whether $\texttt{system\_sat?}(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}, \mathbb{R})$ holds, where $\boldsymbol{q} = q_0, \ldots, q_k$ is a list of univariate polynomials with rational coefficients. The algorithm works by first converting the list $\boldsymbol{q}$ to a list of polynomials with integer coefficients and then calling the function `hutch_int` on the resulting list of polynomials. Each polynomial in the list $\boldsymbol{q}$ is multiplied by the product of the denominators of its coefficients, which results in a polynomial with integer coefficients that satisfies the same sign conditions as the original. This process follows the process in Section 4.2 of [24], where a similar method was used to extend a decision procedure for integer coefficient polynomials to those with rational coefficients. In PVS, rational numbers are represented by a primitive type. For example, numerical constants $\frac{1}{2}$, $\frac{2}{4}$, and 0.5 are indistinguishable. Hence, the definition of a PVS function that computes the numerator and denominator of a rational number is not straightforward. The solution to this problem, while interesting in the context of an interactive theorem prover such as PVS, is not directly necessary for the explanations in this section. In this section, it is assumed that there is a function, namely `compute_pos_rat`, that takes a positive rational number r as input and returns a pair of natural numbers (a, b), relative primes, such that $r = a/b$. For more details, the reader is referred to [24].

The function `rat2poly`, which is defined in PVS, takes a rational polynomial and coverts it to an integer polynomial that is a positive constant multiple of the original. 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. It is formally verified in PVS that `rat2poly`(p) is an integer polynomial that is a positive multiple of p.

The function `hutch` is defined as follows, where the list $\boldsymbol{q}$ of polynomials with rational coefficients is given by $\boldsymbol{q} = q_0, \ldots, q_k$.

$$\begin{aligned}&\texttt{hutch}(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}, \texttt{sos?}) \equiv \\ &\qquad \texttt{let } \boldsymbol{p} = \texttt{rat2poly}(q_0), \ldots, \texttt{rat2poly}(q_k) \texttt{ in} \\ &\qquad \texttt{hutch\_int}(k, \boldsymbol{p}, \boldsymbol{r}, \boldsymbol{Q}, \texttt{sos?}).\end{aligned} \tag{15}$$

The correctness statement for the function `hutch` is given below, and it has been proved in PVS.

THEOREM 3.9. *Let $\boldsymbol{q} = q_0, \ldots, q_k$ be a sequence of polynomials with rational coefficients and positive degree, then*

$$\texttt{hutch}(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}, \texttt{sos?}) = \texttt{true} \tag{16}$$

if and only if the system represented by $(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q})$ is satisfiable over the real line, i.e., `system_sat?`$(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}, \mathbb{R})$ holds.

4. AUTOMATED PROOF PRODUCING STRATEGY

The function `hutch` and Theorem 3.9 can be used to *automatically* prove existentially or universally quantified formulas involving polynomial relations. For instance to prove the universally quantified formula

$$\forall x \in \mathbb{R}\colon (x-1)^3 < 0 \,\wedge\, (x+1)^3 > 0 \implies x^2 < 1, \tag{17}$$

the following general method can be used. First, it is noted that proving Formula (17) is equivalent to proving unsatisfiability of the system $(x-1)^3 < 0 \wedge (x+1)^3 > 0 \wedge x^2 - 1 \geq 0$.

(1) Find k, $\boldsymbol{q}$, $\boldsymbol{r}$, and $\boldsymbol{Q}$ so that the unsatisfiability statement to be proved is equivalent to $\neg$ `system_sat?`$(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}, \mathbb{R})$. In this particular case,

—$k = 2$,
—$\boldsymbol{q} = q_0, q_1, q_2$, with $q_0 = (-1, 3, -3, 1)$, $q_1 = (1, 3, 3, 1)$, $q_2 = (-1, 0, 1)$,
—$r = (<, >, \geq)$, and
—$\boldsymbol{Q}(b_0, b_1, b_2) = b_0 \wedge b_1 \wedge b_2$.

(2) Prove that the propositions $\texttt{eval}(q_0)(x) = (x-1)^3$, $\texttt{eval}(q_1)(x) = (x+1)^3$, and $\texttt{eval}(q_2)(x) = x^2 - 1$ hold identically.

(3) Evaluate the ground expression

$$\texttt{hutch}(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q}). \tag{18}$$

(4) If the evaluation in Step 3 results in `false`, by Theorem 3.9, Formula 17 holds. Otherwise, by Theorem 3.9, the original formula does not hold for all x.

In most interactive theorem provers, the general method described above can be mechanized as a proof producing strategy. In PVS, this method is implemented in a strategy called `hutch`. This strategy automatically discharges Formula (17) in PVS. Particular implementation details of this strategy are specific to each system. However, some technical issues may be common to theorem provers based on higher-order logic. For instance, since a polynomial expression such as $(x-1)^3$ is just a real number in the specification language, the implementation of Step 1 requires reflective capabilities in the strategy language, i.e., the ability to consider expressions in the specification language as data in the strategy language. Using these capabilities, a representation such as $q_0 = (-1, 3, -3, 1)$ can constructed from a polynomial expression like $(x-1)^3$. However, it is still necessary to formally prove that the interpretation of this representation coincides with the polynomial expression, e.g., $\texttt{eval}(q_0)(x) = (x-1)^3$. This can be done in Step 2 by reducing both sides of the equation to the normal form $-1 + 3x + -3x^2 + x^3$. Since the expression in Formula (18) is ground, Step 3 can be efficiently computed using a ground evaluator in a theorem prover that supports this feature. Otherwise, this step can be accomplished by β-reducing the function call $\texttt{hutch}(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q})$.

The PVS strategy `hutch` automatically discharges sequents having one of the following forms.

(1) $\vdash \forall x \colon \mathbb{R}.\ B(x)$
(2) $\vdash \exists x \colon \mathbb{R}.\ B(x)$
(3) $B_1(x), \ldots, B_n(x) \vdash B_{n+1}(x), \ldots, B_m(x)$

where

—$B(x)$ and $B_1(x), \ldots B_m(x)$, with $0 \leq m$, denote arbitrary Boolean expressions involving relations of polynomial expressions on the variable x. It is assumed that all polynomial expressions have numerical coefficients.
—x is either a quantified variable, as in the case of sequents of the form 1 and 2, or an uninterpreted real constant, as in the case of sequents of the form 3.

Sequents of the form 3 are reduced to the form 1 by setting $B = B_1(x) \wedge \ldots \wedge B_n(x) \implies B_{n+1}(x) \vee \ldots \vee B_m(x)$. It is noted that `hutch` does not use external solvers and since it is a proof producing strategy, its correctness only depends on the internal logic of the theorem prover.

Example 4.1. *In PVS, Formula* (17) *can be discharged in the following way.*

```
  |-------
{1}   FORALL (x: real):
        (x - 1)^3 < 0 AND (x + 1)^3 > 0 IMPLIES x^2 < 1

Rule? (hutch)
Q.E.D.
```

In actual proofs, it is often the case that a formula such as Formula (17) *appears implicit in the sequent, e.g.,*

```
{-1}  (x - 1)^3 < 0
{-2}  (x + 1)^3 > 0
  |-------
{1}   x^2 < 1

Rule? (hutch)
Q.E.D.
```

The strategy `hutch` has an optional flag `sos?` that specifies the input `sos?` used in the PVS function `hutch`, which, as in Section 3, determines whether to use the function `decidable_interval_sq` or `decidable_interval` to determine if an interval is small enough to determine satisfiability of the system using `decide_interval`.

5. BENCHMARKS

Table I compares the strategy `tarski`, from the authors' previous work [24], to the strategy `hutch` presented in this paper. The strategy `hutch` is used with the flag `sos?` enabled (default behavior) and with the flag disabled (`sos?` set to `nil`). For reference, Table I also compares these strategies to the strategy `metit` [7], which uses the automatic theorem prover MetiTarski [1] in conjunction with the SMT solver Z3 [6] as external solvers. The strategy `metit` is considerably faster than `tarski` and `hutch`. However, in contrast to the later strategies, `metit` is implemented as a trusted oracle that is not supported by a PVS proof. These strategies are compared on the problems listed in Appendix A.

The problems labeled `Ex1` through `Ex7` come directly from Wi, Passmore, and Paulson [12]. The lemmas labeled `quads_2` through `quads_10` are designed to compare the complexity of `hutch` versus `tarski`. Times in Table I are given in seconds, where PVS is run in batch mode in a Mac Pro 6-Core Intel Xeon E5 3.5 GHz with 32GB of RAM. In each case, the number in parentheses is the CPU time of the ground evaluation of the formally verified decision procedure, i.e., the time that takes the evaluation of `hutch`$(k, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{Q})$ in Step 3 (Section 4). Unreported times correspond to cases where the ground evaluation does not terminate in 5 min. In the case of `metit`, this strategy does not support existential quantification. Therefore, the times for `Ex3` and `Ex6` are reported as N/A. In the cases of strategies `tarski` and `hutch`, the difference between the time in parentheses and the total time is mainly spent in Step 2 (Section 4), i.e., in formally verifying for every polynomial in the system that `eval`$(q)(x) = p(x)$, where q is a list of coefficients representing the polynomial expression $p(x)$. As illustrated by the times in the table, this step,

Problem	`tarski`	`hutch`	`hutch :sos? nil`	`metit`
`Ex1`	1.78 (0.12)	2.68 (0.02)	1.73 (0.02)	0.05
`Ex2`	4.80 (2.16)	2.91 (0.07)	3.19 (0.38)	0.05
`Ex3`	5.07 (0.26)	30.19 (27.22)	6.41 (1.74)	N/A
`Ex4`	12.58 (9.69)	4.07 (0.01)	4.09 (0.05)	0.04
`Ex5`	225.45 (237.96)	5.55 (0.01)	4.44 (0.17)	0.05
`Ex6`	–	70.02 (3.02)	–	N/A
`Ex7`	–	76.76 (51.85)	–	0.05
`quad2`	1.82 (0.01)	1.85 (0.00)	1.85 (0.00)	0.05
`quad3`	2.28 (0.05)	1.05 (0.00)	2.27 (0.00)	0.05
`quad4`	1.83 (0.44)	2.74 (0.00)	2.75 (0.01)	0.05
`quad5`	5.57 (2.88)	3.20 (0.01)	3.25 (0.03)	0.05
`quad6`	22.16 (21.61)	3.75 (0.01)	3.82 (0.08)	0.05
`quad7`	154.43 (175.47)	4.26 (0.01)	4.48 (0.24)	0.05
`quad8`	–	8.73 (0.01)	4.11 (0.53)	0.05
`quad9`	–	11.90 (0.01)	4.46 (1.09)	0.05
`quad10`	–	14.19 (0.02)	7.98 (2.07)	0.05

Table I. Comparison between `tarski` vs. `hutch` (with flag `sos?` enabled and disabled)

which mainly involves simple symbolic checks, takes a significant percentage of the time.

Table I shows that, on these problems, the flag `sos?` does not make a significant impact on problems with small number of polynomials, e.g., 5, and that `hutch` is slightly better than `tarski`. In larger problems, `hutch` with the flag `sos?` enabled outperforms `hutch` with the flag `sos?` disabled as the product polynomial π and its corresponding Sturm chain Π get too big. Compared to `hutch`, `tarski` gets slow quicker on larger problems. An exception to the previous comment is `Ex3`, where both `tarski` and `hutch` with `sos?` disabled performs better than `hutch` with `sos?` enabled. This is possibly due to the high degrees of the polynomials involved in this example.

For reference, Table II shows a comparison, reported in [12], of the Isabelle/HOL tactics `univ_rcf` and `univ_rcf_cert` to the PVS strategy `tarski`.[2] Unreported times correspond to instances where the tactics and strategy do not finish in 2 min. As far as the authors know, these proof producing tactics are the closest in spirit to `hutch`. The Isabelle/HOL tactics call an external tool (Mathematica) in a sound way to do root isolation, while the PVS strategy `hutch` does the root isolation inside the theorem prover. However, Table I and Table II suggest that the ground evaluation of the verified decision procedure `hutch` with the flag `sos?` enabled is approximately the same order of magnitude as the times spent by the Isabelle/HOL tactics `univ_rcf` and `univ_rcf_cert`.

6. RELATED WORK

Work on similar decision procedures has been done by the authors. In their previous work [24], the authors presented two formally verified, computable functions for determining the satisfiability of a system of univariate polynomial relations (equalities

[2]Since the authors do not have access to the Mathematica system, they are unable to replicate these results.

Problem	`univ_rcf` (Isabelle/HOL)	`univ_rcf_cert` (Isabelle/HOL)	`tarski` (PVS)
`Ex1`	0.9	0.3	2.0
`Ex2`	1.4	0.6	6.8
`Ex3`	1.6	0.7	13.0
`Ex4`	1.3	0.5	20.1
`Ex5`	1.6	0.6	315.7
`Ex6`	5.6	3.9	–
`Ex7`	38.4	34.9	–

Table II. Comparison between `univ_rcf` vs. `tarski` as reported in [12]

and inequalities) over the real line. This current paper presents another formally verified algorithm that enjoys the same properties as the algorithms in their previous work. However, the algorithm presented here is simpler and shorter and should be more easily implemented by others wishing to incorporate such functionality into theorem provers other than PVS. A notable difference between the strategy `hutch`, presented here, and the strategy `tarski`, presented in [24], is that `hutch` discharges sequents written in a more general form. In particular, the strategy `hutch` supports arbitrary Boolean expressions involving polynomial relations, while `tarski` requires either existentially quantified formulas of conjunctive relations or universally quantified formulas of disjunctive relations. While it is technically possibly to extend the strategy `tarski` to support arbitrary Boolean expressions, the implementation of this feature may require several calls to the decision procedure upon which that strategy is based. In the strategy `hutch`, the implementation is straightforward because the decision procedure `hutch`, presented in Section 3, already supports arbitrary Boolean expressions.

Aside from the authors' previous work, the algorithm presented in this paper is closest in spirit to that of Wi, Paulson, and Passmore [12], which provides an algorithm to determine the satisfiability of polynomial systems such as those discussed in this paper. The correctness of the algorithm presented in that work depends only on the kernel of the prover (in that case, Isabelle). However, that procedure relies on an external tool (Mathematica) for root isolation. If the root isolation is not precise enough, the external tool is asked for more precise root isolation intervals. The correctness of that method does not depend on the correctness of the external tool. However, completness of that method may be compromised if the external tool is not sound. In contrast to that work, the method presented in the current paper does not depend on an external tool and therefore it is sound and complete assuming that the PVS internal logic is sound. The algorithm presented in this paper stops subdividing when there is at most one point that is a root of any polynomial in each subinterval. This improves the efficiency of the method as it quickly determines whether the subdivision intervals are small enough to contain at most one root of the polynomials. Finally, the algorithm presented in this paper is recursive in nature, and its proof is therefore different than the algorithm of Wi, Paulson, and Passmore [12]. That proof makes an argument at the global level once all roots have been isolated, whereas the algorithm in this paper is verified by induction on the number of intervals, the hardest part being the verification of correctness of the subdivision step.

For a more comprehensive description of related work in this area, see the authors' previous paper [24], which covers work by Harrison [10], Mahboubi [13], Eberl [8], de Moura [6], and others. Previous work by Cohen and Mahboubi [3] is also especially notable because they formally proved a full decision procedure for multivariate polynomial systems. Their method is also based on Tarski queries, and the formalization is in Coq. One difference is that their procedure is not executable, although it will provide a useful resource as the authors of the current paper attempt an executable decision procedure in PVS for multivariate systems.

7. CONCLUSION

This paper presents a formally verified decision procedure that determines satisfiability of univariate polynomial systems using root counting and interval subdivision. This decision procedure, which is proven to be sound and complete, is at basis of a proof producing automated strategy whose correctness solely depends on the internal logic of the theorem prover. The procedure uses a root counting technique based on Sturm's theorem. It progressively subdivides the real line until it is determined that all polynomials in the system with a root in the given interval have their roots at the same single location. Satisfiability is easily determined on intervals satisfying this property. Preliminary results show that the algorithm scales well to systems with a large number of polynomials. An optional parameter allows for additional tuning of the algorithm, specially when dealing when polynomials with high degrees.

Other algorithms already exist to determine the satisfiability of these types of systems, including one by the authors based on Sturm's and Tarski's theorems [24], and another one by Wi, Paulson, and Passmore [12] based on first isolating roots with an external tool (Mathematica) and then checking the answer. The fact that the algorithm presented in this paper is formally verified in the same tool that it will be called is compelling.

It is not straightforward to compare the performance of strategies in different theorem provers since each system is particular in how problems are posed to the theorem prover. In the case of PVS, the strategy presented in this paper supports general polynomial expressions written as real number expressions and that are not expected to be in a particular normal form. This support is convenient for the user. However, the price of this convenience is that these real numbers expressions have to be automatically translated by the strategy into a deep embedding representation of polynomial expressions. The correctness of these translations has to be proved for every polynomial expression. These proofs, while logically simple, are time consuming for large polynomials. The bottleneck appears to be due to the performance limitations of the PVS rewriting capabilities. Implementing the algorithm presented in this paper in an interactive theorem prover with a better support for rewriting and computational reflection, such as Isabelle/HOL or Coq, may provide a simple to verify, efficient approach to determining satisfiability of univariate systems.

References

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

[2] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Cyril Cohen and Assia Mahboubi. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Logical Methods in Computer Science*, 8(1:02):1–40, February 2012.

[4] 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.

[5] 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.

[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. In Cliff Jones, Pekka Pihlajasaari, and Jun Sun, editors, *Proceedings of the 19th International Symposium on Formal Methods (FM 2014)*, volume 8442 of *Lecture Notes in Computer Science*, pages 194–199, Singapore, May 2014. Springer.

[8] Manuel Eberl. Sturm's theorem. *Archive of Formal Proofs*, January 2014. http://afp.sf.net/entries/Sturm_Sequences.shtml, Formal proof development.

[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] 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.

[11] 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.

[12] Wenda Li, Grant Olney Passmore, and Lawrence C. Paulson. Deciding univariate polynomial problems using untrusted certificates in Isabelle/HOL. *Journal of Automated Reasoning*, Aug 2017.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] Mariano Moscato, Laura Titolo, Aaron Dutle, and César A. Muñoz. Automatic estimation of verified floating-point round-off errors via static analysis. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security: 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings*, volume 10488 of *Lecture Notes in Computer Science*, pages 213–229, Cham, 2017. Springer International Publishing.

[19] César Muñoz, Víctor Carreño, and Gilles Dowek. Formal analysis of the operational concept for the Small Aircraft Transportation System. In *Rigorous Engineering of Fault-Tolerant Systems*, volume 4157 of *Lecture Notes in Computer Science*, pages 306–325, 2006.

[20] César Muñoz, Aaron Dutle, Anthony Narkawicz, and Jason Upchurch. Unmanned Aircraft Systems in the National Airspace System: A formal methods perspective. *ACM SIGLOG News*, 3(3):67–76, July 2016.

[21] 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.

[22] 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.

[23] Anthony Narkawicz, César Muñoz, and Gilles Dowek. Provably correct conflict prevention bands algorithms. *Science of Computer Programming*, 77(1–2):1039–1057, September 2012.

[24] Anthony Narkawicz, César Muñoz, and Aaron Dutle. Formally-verified decision procedures for univariate polynomial computation based on Sturm's and Tarski's theorems. *Journal of Automated Reasoning*, 54(4):285–326, 2015.

[25] 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.

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

A. BENCHMARKS

$$Ex1: \forall\, x \in \mathbb{R} : x \geq -9 \,\wedge\, x < 10 \,\wedge\, x^4 > 0 \implies x^{12} > 0.$$

$$\begin{aligned} Ex2: &\forall\, x \in \mathbb{R} : (x-2)^2 \cdot (-x+4) > 0 \,\wedge\, x^2 \cdot (x-3)^2 \geq 0 \,\wedge\, x - 1 \geq 0 \,\wedge \\ &-(x-3)^2 + 1 > 0 \implies (-(x - 11/12))^3 \cdot (x - 41/10)^3 \geq 0. \end{aligned}$$

$$\begin{aligned} Ex3: &\exists\, x \in \mathbb{R} : x^5 - x - 1 = 0 \,\wedge\, x^{12} + 425/23 \cdot x^{11} - 228/23 \cdot x^{10} - 2 \cdot x^8 \\ &- 896/23 \cdot x^7 - 394/23 \cdot x^6 + 456/23 \cdot x^5 + x^4 + 471/23 \cdot x^3 \\ &+ 645/23 \cdot x^2 - 31/23 \cdot x - 228/23 = 0 \,\wedge\, x^3 + 22 \cdot x^2 - 31 \geq 0 \,\wedge \\ &x^{22} - 234/567 \cdot x^{20} - 419 \cdot x^{10} + 1948 > 0. \end{aligned}$$

$$\begin{aligned} Ex4: &\forall\, x \in \mathbb{R} : x > 0 \,\vee\, -((61 \cdot x)/9) + (5 \cdot x^2)/9 + (20 \cdot x^3)/9 > -4 \,\vee \\ &1 \leq x \,\vee\, x \leq 0 \,\vee\, -((19 \cdot x)/9) + (10 \cdot x^2)/9 \leq -1 \,\vee\, -((13 \cdot x)/9) \\ &+ (31 \cdot x^2)/45 + x^3/18 \leq -(7/10) \,\vee\, -((61 \cdot x)/9) + (5 \cdot x^2)/9 \\ &+ (20 \cdot x^3)/9 \leq -4. \end{aligned}$$

$$\begin{aligned} Ex5: &\forall\, x \in \mathbb{R} : -((5 \cdot x)/6) - (10 \cdot x^2)/3 - x^3/3 > 0 \,\vee\, (5 \cdot x)/6 \\ &+ (10 \cdot x^2)/3 + x^3/3 > 0 \,\vee\, 1 \leq x \,\vee\, x \leq 0 \,\vee\, -((19 \cdot x)/9) \\ &+ (10 \cdot x^2)/9 \leq -1 \,\vee\, -((13 \cdot x)/9) + (31 \cdot x^2)/45 + x^3/18 \leq -(7/10) \\ &\vee\, -((101 \cdot x)/30) - (64 \cdot x^2)/15 + (14 \cdot x^3)/15 \leq -(11/5) \,\vee \\ &-((61 \cdot x)/9) + (5 \cdot x^2)/9 + (20 \cdot x^3)/9 \leq -4. \end{aligned}$$

$$
\begin{aligned}
Ex6: &\exists\, x \in \mathbb{R} : -((51 \cdot x)/10) - (267 \cdot x^2)/2 - (5409 \cdot x^3)/10 - (4329 \cdot x^4)/5 \\
&- (2052 \cdot x^5)/5 - 70 \cdot x^6 > -(7/10) \;\wedge\; -((10327 \cdot x)/270) \\
&- (71681 \cdot x^2)/270 - (135853 \cdot x^3)/810 - (57328 \cdot x^4)/135 \\
&+ (77743 \cdot x^5)/135 + (115774 \cdot x^6)/405 + (175 \cdot x^7)/18 + (49 \cdot x^8)/3 \\
&+ (49 \cdot x^9)/162 > -(721/90) \;\wedge\; -((2981 \cdot x)/90) - (251 \cdot x^2)/6 \\
&- (24217 \cdot x^3)/270 + (2698 \cdot x^4)/135 + (18964 \cdot x^5)/135 \\
&- (595 \cdot x^6)/54 + (280 \cdot x^7)/27 + (7 \cdot x^8)/27 > -(206/45) \;\wedge \\
&- ((799 \cdot x)/90) + (169 \cdot x^2)/18 - (7933 \cdot x^3)/270 + (2672 \cdot x^4)/135 \\
&+ (329 \cdot x^5)/90 + (112 \cdot x^6)/27 + (7 \cdot x^7)/54 > -(103/90) \;\wedge \\
&- ((781 \cdot x)/90) - (701 \cdot x^2)/6 - (12217 \cdot x^3)/270 + (11323 \cdot x^4)/135 \\
&+ (7264 \cdot x^5)/135 + (935 \cdot x^6)/54 + (280 \cdot x^7)/27 \\
&+ (7 \cdot x^8)/27 > -(77/15) \;\wedge\; -((361 \cdot x)/30) \\
&- (811 \cdot x^2)/30 + (307 \cdot x^3)/45 + (2353 \cdot x^4)/90 - (17 \cdot x^5)/6 \\
&+ (52 \cdot x^6)/9 + (2 \cdot x^7)/9 > -(44/15) \;\wedge\; -((33 \cdot x)/10) - (2 \cdot x^2)/15 \\
&+ (41 \cdot x^3)/90 + (2 \cdot x^4)/15 + 2 \cdot x^5 + x^6/9 > -(11/15) \;\wedge \\
&- ((1339 \cdot x)/405) - (70225 \cdot x^2)/324 - (11549 \cdot x^3)/270 \\
&+ (65378 \cdot x^4)/405 + (23483 \cdot x^5)/810 + (1109 \cdot x^6)/27 \\
&+ (1540 \cdot x^7)/81 + (49 \cdot x^8)/162 > -(721/60) \;\wedge\; -((10741 \cdot x)/540) \\
&- (2263 \cdot x^2)/45 + (5191 \cdot x^3)/180 + (7753 \cdot x^4)/270 - (52 \cdot x^5)/9 \\
&+ (203 \cdot x^6)/18 + (7 \cdot x^7)/27 > -(103/15) \;\wedge\; -((1481 \cdot x)/90) \\
&- (811 \cdot x^2)/180 + (2113 \cdot x^3)/90 - (493 \cdot x^4)/36 + (59 \cdot x^5)/9 \\
&+ (2 \cdot x^6)/9 > -(22/5) \;\wedge\; -((913 \cdot x)/180) + (563 \cdot x^2)/90 \\
&- (257 \cdot x^3)/60 + (17 \cdot x^4)/9 + x^5/9 > -(11/10) \;\wedge \\
&- ((91 \cdot x)/18) + (10 \cdot x^2)/3 - (5 \cdot x^3)/2 + (20 \cdot x^4)/9 > -2 \;\wedge \\
&- ((2 \cdot x)/9) - (25 \cdot x^2)/18 + (10 \cdot x^3)/9 > -(1/2) \;\wedge \\
&- ((61 \cdot x)/9) + (5 \cdot x^2)/9 + (20 \cdot x^3)/9 > -4 \;\wedge\; 1 > x \;\wedge\; x > 0 \;\wedge \\
&- ((19 \cdot x)/9) + (10 \cdot x^2)/9 > -1 \;\wedge\; -((13 \cdot x)/9) + (31 \cdot x^2)/45 \\
&+ x^3/18 > -(7/10) \;\wedge\; -((253 \cdot x)/90) - (53 \cdot x^2)/30 + (34 \cdot x^3)/15 \\
&+ x^4/9 > -(11/5) \;\wedge\; -((97 \cdot x)/90) - (2051 \cdot x^2)/90 + (86 \cdot x^3)/15 \\
&+ (82 \cdot x^4)/9 + (2 \cdot x^5)/9 > -(44/5) \;\wedge\; -((93307 \cdot x)/1620) \\
&- (298609 \cdot x^2)/810 + (30583 \cdot x^3)/270 + (264373 \cdot x^4)/810 \\
&- (289811 \cdot x^5)/1620 + (3113 \cdot x^6)/27 + (931 \cdot x^7)/81 + (8 \cdot x^8)/81 > \\
&- (193/5) \;\wedge -((4741 \cdot x)/540) - (9151 \cdot x^2)/90 + (6397 \cdot x^3)/60 \\
&- (2686 \cdot x^4)/135 + (28 \cdot x^5)/9 + (38 \cdot x^6)/3 + (7 \cdot x^7)/27 > -(77/10).
\end{aligned}
$$

$$\begin{aligned}
Ex7 : &\forall\, x \in \mathbb{R} : x < -1 \,\vee\, 0 > x \,\vee\, (41613 \cdot x)/2 + 26169 \cdot x^2 \\
&+ (64405 \cdot x^3)/4 + 4983 \cdot x^4 + (7083 \cdot x^5)/10 + (1207 \cdot x^6)/35 \\
&+ x^7/8 > -6435 \,\vee\, 11821609800 \cdot x + 22461058620 \cdot x^2 + 35 \cdot x^{12} \leq \\
&4171407240 \cdot x^3 + 45938678170 \cdot x^4 + 54212099480 \cdot x^5 \\
&+ 31842714428 \cdot x^6 + 10317027768 \cdot x^7 + 1758662439 \cdot x^8 \\
&+ 144537452 \cdot x^9 + 5263834 \cdot x^{10} + 46204 \cdot x^{11} \,\vee\, x \leq 0 \,\vee \\
&9609600 \cdot x + 45805760 \cdot x^2 + 92372280 \cdot x^3 + 102560612 \cdot x^4 \\
&+ 68338600 \cdot x^5 + 27930066 \cdot x^6 + 6857016 \cdot x^7 + 938908 \cdot x^8 \\
&+ 58568 \cdot x^9 + 753 \cdot x^{10} \leq 0 \,\vee\, 788107320 \cdot x + 1101329460 \cdot x^2 \\
&+ 10 \cdot x^{11} \leq 782617220 \cdot x^3 + 2625491260 \cdot x^4 + 2362290448 \cdot x^5 \\
&+ 1063536663 \cdot x^6 + 240283734 \cdot x^7 + 24397102 \cdot x^8 + 1061504 \cdot x^9 \\
&+ 9179 \cdot x^{10} \,\vee\, 90935460 \cdot x + 81290790 \cdot x^2 + 5 \cdot x^{10} \leq 125595120 \cdot x^3 \\
&+ 237512625 \cdot x^4 + 161529144 \cdot x^5 + 51834563 \cdot x^6 + 6846880 \cdot x^7 \\
&+ 356071 \cdot x^8 + 2828 \cdot x^9 \,\vee\, 640640 \cdot x + 2735040 \cdot x^2 + 4837448 \cdot x^3 \\
&+ 4581220 \cdot x^4 + 2505504 \cdot x^5 + 794964 \cdot x^6 + 138652 \cdot x^7 + 11237 \cdot x^8 \\
&+ 207 \cdot x^9 \leq 0 \,\vee\, 5 \cdot x^8 \leq 73920 \cdot x + 238560 \cdot x^2 + 303324 \cdot x^3 \\
&+ 192458 \cdot x^4 + 63520 \cdot x^5 + 10261 \cdot x^6 + 608 \cdot x^7 \,\vee\, 73920 \cdot x \\
&+ 278880 \cdot x^2 + 424284 \cdot x^3 + 332962 \cdot x^4 + 142928 \cdot x^5 + 32711 \cdot x^6 \\
&+ 3514 \cdot x^7 + 98 \cdot x^8 \leq 0 \,\vee\, x \leq -1.
\end{aligned}$$

$$\begin{aligned}
quads\_2 : &\forall\, x \in \mathbb{R} : x > 0 \wedge x < 2 \implies ((x-0)\cdot(x-1) \leq 0 \,\vee \\
&(x-1)\cdot(x-2) \leq 0).
\end{aligned}$$

$$\begin{aligned}
quads\_3 : &\forall\, x \in \mathbb{R} : x > 0 \wedge x < 3 \implies ((x-0)\cdot(x-1) \leq 0 \,\vee \\
&(x-1)\cdot(x-2) \leq 0 \,\vee\, (x-2)\cdot(x-3) \leq 0).
\end{aligned}$$

$$\begin{aligned}
quads\_4 : &\forall\, x \in \mathbb{R} : x > 0 \wedge x < 4 \implies ((x-0)\cdot(x-1) \leq 0 \,\vee \\
&(x-1)\cdot(x-2) \leq 0 \,\vee\, (x-2)\cdot(x-3) \leq 0 \,\vee\, (x-3)\cdot(x-4) \leq 0).
\end{aligned}$$

$$\begin{aligned}
quads\_5 : &\forall\, x \in \mathbb{R} : x > 0 \wedge x < 5 \implies ((x-0)\cdot(x-1) \leq 0 \,\vee \\
&(x-1)\cdot(x-2) \leq 0 \,\vee\, (x-2)\cdot(x-3) \leq 0 \,\vee\, (x-3)\cdot(x-4) \leq 0 \,\vee \\
&(x-4)\cdot(x-5) \leq 0).
\end{aligned}$$

$$\begin{aligned}
quads\_6 : &\forall\, x \in \mathbb{R} : x > 0 \wedge x < 6 \implies ((x-0)\cdot(x-1) \leq 0 \,\vee \\
&(x-1)\cdot(x-2) \leq 0 \,\vee\, (x-2)\cdot(x-3) \leq 0 \,\vee\, (x-3)\cdot(x-4) \leq 0 \,\vee \\
&(x-4)\cdot(x-5) \leq 0 \,\vee\, (x-5)\cdot(x-6) \leq 0).
\end{aligned}$$

$$\begin{aligned} quads\_7 :& \forall x \in \mathbb{R} : x > 0 \land x < 7 \implies ((x-0)\cdot(x-1) \leq 0 \lor \\ & (x-1)\cdot(x-2) \leq 0 \lor (x-2)\cdot(x-3) \leq 0 \lor (x-3)\cdot(x-4) \leq 0 \lor \\ & (x-4)\cdot(x-5) \leq 0 \lor (x-5)\cdot(x-6) \leq 0 \lor (x-6)\cdot(x-7) \leq 0). \end{aligned}$$

$$\begin{aligned} quads\_8 :& \forall x \in \mathbb{R} : x > 0 \land x < 8 \implies ((x-0)\cdot(x-1) \leq 0 \lor \\ & (x-1)\cdot(x-2) \leq 0 \lor (x-2)\cdot(x-3) \leq 0 \lor (x-3)\cdot(x-4) \leq 0 \lor \\ & (x-4)\cdot(x-5) \leq 0 \lor (x-5)\cdot(x-6) \leq 0 \lor (x-6)\cdot(x-7) \leq 0 \lor \\ & (x-7)\cdot(x-8) \leq 0). \end{aligned}$$

$$\begin{aligned} quads\_9 :& \forall x \in \mathbb{R} : x > 0 \land x < 9 \implies ((x-0)\cdot(x-1) \leq 0 \lor \\ & (x-1)\cdot(x-2) \leq 0 \lor (x-2)\cdot(x-3) \leq 0 \lor (x-3)\cdot(x-4) \leq 0 \lor \\ & (x-4)\cdot(x-5) \leq 0 \lor (x-5)\cdot(x-6) \leq 0 \lor (x-6)\cdot(x-7) \leq 0 \lor \\ & (x-7)\cdot(x-8) \leq 0 \lor (x-8)\cdot(x-9) \leq 0). \end{aligned}$$

$$\begin{aligned} quads\_10 :& \forall x \in \mathbb{R} : x > 0 \land x < 10 \implies ((x-0)\cdot(x-1) \leq 0 \lor \\ & (x-1)\cdot(x-2) \leq 0 \lor (x-2)\cdot(x-3) \leq 0 \lor (x-3)\cdot(x-4) \leq 0 \lor \\ & (x-4)\cdot(x-5) \leq 0 \lor (x-5)\cdot(x-6) \leq 0 \lor (x-6)\cdot(x-7) \leq 0 \lor \\ & (x-7)\cdot(x-8) \leq 0 \lor (x-8)\cdot(x-9) \leq 0 \lor (x-9)\cdot(x-10) \leq 0). \end{aligned}$$

B. TABLE OF THEOREMS AND THEIR CORRESPONDING PVS NAMES

In following table, the notation `lib@th.thm` refers to the theorem `thm` in the theory `th` of the library `lib`. The notation `lib@th.func(...)` refers to the type of the function `func` in the theory `th` of the library `lib`.

Theorem	**PVS Name**
Theorem 2.1	`Tarski@sturmtarski.sturm_tarski` and `Tarski@sturmtarski.sturm_tarski_unbounded`
Theorem 2.2	`Sturm@compute_sturm.roots_closed_int_def`
Theorem 3.1	`Tarski@hutch.decide_interval_def`
Theorem 3.2	`Tarski@hutch.known_signs_update_sound`
Theorem 3.3	`Tarski@hutch.decide_interval_def`
Theorem 3.4	`Tarski@hutch.decidable_intervals_exist`
Theorem 3.5	`Tarski@hutch.decidable_intervals_sq_exist`
Theorem 3.6	`Tarski@hutch.hutch_int_basic(...)`
Theorem 3.7	`reals@hutch.Knuth_poly_root_strict_bound(...)`
Theorem 3.8	`Tarski@hutch.hutch_int_def`
Theorem 3.9	`Tarski@hutch.hutch_def`