A Complete and Practical Algorithm for Geometric Theorem Proving

(Extended Abstract)

Ashutosh Rege *

Computer Science Division
University of California
Berkeley, CA 94720

Abstract

This paper describes a complete and practical algorithm for the problem of geometric theorem proving. The algorithm works over algebraically closed fields as well as over the reals and takes care of degenerate cases. Our work is motivated by several recent improvements in algorithms for sign determination and symbolic-numeric computation. Based on these, we provide an algorithm for solving triangular systems efficiently using straight-line program arithmetic. The report concludes with a description of an implementation and provides preliminary benchmarks from the same.

1 Introduction

In this paper, we describe a practical algorithm (and its implementation) for automatic geometric theorem proving and related problems. Earlier methods for geometric theorem proving were based on a synthetic geometric reasoning approach using natural deduction, forward and backward chaining and the like [7]. Following Wu's seminal work ([17], [18]), a significant amount of the recent work has focussed on an algebraic approach involving determining the feasibility of systems of polynomial equations [3], [10], [9]. Our work is based on this latter methodology and is motivated by recent improvements in algorithms for symbolic-numeric computation with polynomials [13]. The papers by Wu and others showed how a subclass of geometric theorems can be proved by expressing the hypotheses and conclusions as polynomial equations. A geometric theorem can then be proved by essentially determining the algebraic set corresponding to the hypotheses and checking whether the algebraic sets of the conclusions contain it. Different methods have been proposed for doing this - Wu's method involves reduction of the hypotheses system of polynomials to a triangular form (i.e. one in which each successive polynomial has at most one extra variable) and then using successive pseudo-division to determine the feasibility of the conclusion. Other work has focussed on using Gröbner bases to determine whether a system of polynomials (obtained from the original polynomials) has a common zero [10], [9].

Wu's original method is incomplete in the sense that the feasibility of the conclusions is determined over an algebraically closed field whereas geometric theorems are often assertions about real numbers. It is not possible to refute a conjecture over the reals using this approach and in general, confirmation of a theorem can not be obtained automatically from a proof over an algebraically closed field.

Another important consideration in geometric theorem proving is the notion of *genericity*. It is necessary to rule out *degenerate* cases such as non-distinct points or zero-radius circles. Wu's method and others generate a set of subsidiary polynomials representing such cases so that the conclusion polynomial vanishes at the common zeros of the hypotheses which are not zeros of the subsidiary polynomials.

From the above discussion, it seems important that a geometric theorem prover should satisfy the following criteria

1. It should work over the reals as well as the complex field.
2. It should handle degenerate cases.
3. It should be practicable.

In this paper we discuss a method for geometric theorem proving which satisfies the first two criteria and based on an initial implementation, seems to be rather efficient in practice.

*E-mail: rege@cs.Berkeley.edu. Fax: +1-510-642-5775.

Supported by a David and Lucile Packard Foundation Fellowship and by NSF P.Y.I. Grant FD93-20588

The basic problem we consider is the following : *given a system of hypotheses polynomial equations in triangular form, determine whether the set of their common zeros over the reals is contained in the set of real zeros of the conclusion polynomial.* (One can use any standard algorithm for triangulating if the hypotheses are not triangular to begin with, see e.g. [4]). The observation here is that, in order to prove a theorem or refute it, one is only interested in the *sign* (+, - or 0) of the conclusion polynomial at the real zeros of the hypotheses polynomials. We provide an efficient method to encode the roots of a triangular system using the Sylvester resultant. To determine the signs, we make use of Sturm sequences which enable us to determine the number of real roots of the hypotheses where the conclusion is non-zero. Thus our algorithm can determine the truth or falsehood of an assertion over the reals.

The problem of genericity is handled by instantiating the independent variables (or parameters) of the given theorem with random values. Probabilistically, over $\mathbb{C}$ this takes care of degenerate cases without affecting the validity of the conclusion. Over $\mathbb{R}$ it is harder to get a necessary condition for the theorem to be true - the algorithm for checking the necessary condition is given in section 7.

Computation in the prover is done over the field of reals (or rationals) extended by variables. The numbers in this extension field are then rational functions in these variables. It is, however, very expensive to compute with explicit rational functions. We circumvent this difficulty by using *straight-line programs* or SLP's to represent all intermediate calculations. SLP's are extremely useful for various types of computations with polynomials over field extensions. In the algorithms we need for theorem-proving, one is primarily interested in the signs of the coefficients of polynomials over the extension field. These can be determined efficiently when the computation is done over the field of SLP's.

In order to make our prover more efficient, we use "mixed" arithmetic for algebraic computations : the idea is to represent a number as a structure with two values, the first is the integer value of the number modulo a prime p and the second is a floating-point approximation to that number. We use the finite field part for equality tests and the floating point part for relative ordering of two numbers and sign-determination.

Geometric theorem proving falls in the more general class of problems defined by the first-order theory of the reals. The work described in the subsequent sections is part of ongoing research at Berkeley aimed at developing an efficient practicable toolkit for solving more general problems. We plan on applying the tools and techniques described in this paper to a broader scope of problems including solving systems of polynomial equations, point-location in varieties, existential theory of the reals etc. (See [13] for an overview.) Further, one can use these techniques to extend our prover to work with polynomial inequalities and also to solve more general problems in constraint-based reasoning.

The paper is organized as follows : Section 2 gives the formulation of the problem of theorem proving we shall use. In section 3 we give an outline of the prover implemented by Chou [4], since we shall use that as a benchmark to compare our prover with. Section 4 gives an overview of our theorem prover. Section 5 sketches the algorithm for solving triangular systems. Sections 6 and 7 discuss theorem proving and refutation over $\mathbb{R}$ and $\mathbb{C}$. We conclude with Section 8 which describes the implementation and give benchmarks on some problems.

2 Geometric Theorem Proving - Formulation of the Problem

This section provides the formulation we shall use of the questions of what constitutes a geometric theorem and what it means to prove it. Several formulations of these two questions have been proposed [7], [8],[10]. See the book by Chou [4] for a discussion of the various formulations and the limitations of each. We will use the formulation used by Chou [4] which seems to be the one that is the most satisfactory :

By a *geometric statement* (S) we shall mean a collection of polynomial equations

$$h_1(y_1, \ldots, y_m) = \ldots = h_n(y_1, \ldots, y_m) = 0$$

called the *hypotheses* and a *conclusion* $g(y_1, \ldots, y_m) = 0$ where the h_i and g are in $K[y_1, \ldots, y_m]$. Here K is some field under consideration, usually $\mathbb{Q}$.

Let $V = V(h_1, \ldots, h_n)$ denote the set of common zeroes in E^m, where E is an extension of K Thus an instance of the theorem, i.e. a configuration which satisfies the given hypotheses, corresponds to an element of V and vice versa. Note, however, that V contains the so-called degenerate cases which happen to satisfy the hypotheses. In order to distinguish the degenerate cases from the nondegenerate ones, we use the variables $u_1, \ldots, u_d \equiv \mathbf{u}$, called the *parameters*, among the y_i to denote the nonzero coordinates of those points which can be chosen arbitrarily and we use the variables $x_1, \ldots, x_n \equiv \mathbf{x}$, to denote the rest of the y_i. These latter are called *dependent variables*.

If the extension field E is *algebraically closed* a basic fact of algebraic geometry is that every algebraic set has a unique decomposition as a union of irreducible algebraic sets such that no one contains another. Thus the set V can be expressed as a union,

$$V = V_1 \cup \ldots \cup V_s \cup V_1{}^* \cup \ldots \cup V_t{}^*$$

where the V_i are all those (*nondegenerate*) components on which the u_j are algebraically independent and the ${V_k}^*$ are those (*degenerate*) components on which they are dependent. We call the V_i components which are *general for u*.

If E is *not algebraically closed*, the decomposition is, in general, not an irreducible one. However, the u_j are algebraically dependent on the algebraic sets ${V_k}^*$ and the latter still correspond to the degenerate cases. We will still call the V_i components *general for u.* (Note, however, that in this case any V_i can be reducible or empty.)

We say that a geometric statement (S) is *generally true* if g vanishes on all nondegenerate components of V. Else it is *not generally true.* A geometric statement is *generally false* if g vanishes on none of the nondegenerate components of V.

3 The Ritt-Wu-Chou Prover

In this section, we provide a brief description of the prover implemented by Chou [4] based on the work of Ritt and Wu. Among all the approaches that have been used and implemented, Chou's prover seems to be most complete and efficient and we will use this prover as the basis of comparison for our prover. This section will therefore try to give a basic knowledge of the algorithms and techniques used by Chou's prover so we can better understand its strengths and limitations.

In 1978, Wu introduced an elegant method for automated theorem proving for algebraic geometry [17]. The basis of his work was a triangulation procedure which was implicit in the work of Ritt [15], [16] and which was given in detail by Wu. Wu's approach was implemented with various modifications and improvements by Chou in his prover [3], [4].

The basic algorithm for the prover implemented by Chou is the following (assuming the notation introduced in the previous section) :

- **Triangulation** The hypotheses are converted into a triangular form

  $$f_1(\mathbf{u}, x_1), f_2(\mathbf{u}, x_1, x_2) \ \cdots, \ f_n(\mathbf{u}, x_1, \ldots, x_n)$$

- **Successive pseudo-division** Letting $g = R_n$, the remainder R_0 is computed by taking successive pseudo-remainders : at the i^{th} step ($i = n \ldots 0$) the remainder computed is $prem(R_{i+1}, f_{i+1}, x_{i+1})$ $= R_i$. Depending on whether $R_0 = 0$ or not, and depending on other conditions (discussed below) the prover can conclude whether the theorem is generally true or not.

We will now discuss the various issues that come up in the process of triangulation and successive pseudo-division and how Chou's prover deals with them.

3.1 Triangulation

A triangular system can be obtained rather easily from the original system of hypotheses polynomials by using pseudo-remainders to eliminate variables. A simple algorithm for triangulation is the following (see e.g. [4], [5]) :

Assume we are given a system of hypotheses polynomials as before,

$$h_1(\mathbf{u}, \mathbf{x}) = \cdots = h_n(\mathbf{u}, \mathbf{x}) = 0$$

In order to transform it into a triangular form, we first eliminate x_n from $n-1$ polynomials to obtain

$$p_1(\mathbf{u}, x_1, \ldots, x_{n-1}) = \cdots = p_{n-1}(\mathbf{u}, x_1, \ldots, x_{n-1})$$
$$= p_n(\mathbf{u}, x_1, \ldots, x_n) = 0$$

We now apply the same procedure to eliminate x_{n-1} from $n-2$ polynomials and so on. The algorithm eliminates x_n by taking pseudo-remainders as required (see [5]).

At the end of this procedure we obtain a system of the form

$$f_1(\mathbf{u}, x_1) = f_2(\mathbf{u}, x_1, x_2) = \cdots = f_n(\mathbf{u}, x_1, \ldots, x_n) = 0$$

3.2 Successive pseudo-division

Once a triangular form $f_1 \ldots f_n$ is obtained, the next step is to do successive pseudo-division as explained earlier. This results in a final pseudo-remainder R_0 which we denote by *prem*$(g, f_1, \ldots, f_n)$. It is easy to show the following remainder formula :

$$I_1{}^{s_1} \cdots I_n{}^{s_n} g = Q_1 f_1 + \cdots + Q_n f_n + R_0$$

where the I_j are the leading coefficients or *initials* of the f_jr; $s_1, \ldots, s_n$ are non-negative integers and $Q_1, \ldots, Q_n$ are polynomials.

If the final remainder $R_0 = 0$ then we can say that $g = 0$ if $I_1 \neq 0, \ldots, I_n \neq 0$. Thus we get *subsidiary conditions*, namely $I_1 \neq 0, \ldots, I_n \neq 0$ which are included in the hypotheses of the original geometric statement (S). These subsidiary conditions are usually connected with nondegeneracy.

3.3 Problems with the above algorithms

The main problem with the simple triangulation procedure given above is that the variety of the triangular system obtained at the conclusion of the procedure could be strictly larger than the one we started out with due to the process of taking pseudo-remainders. If it turns out that the conclusion polynomial g vanishes on this new larger variety, then the validity of the theorem holds on the smaller variety *a fortiori*. It could easily be the case, however, that g does not vanish on the larger variety. This problem is resolved in the Ritt-Wu-Chou algorithm by testing if the triangular system obtained above is *irreducible* that is, whether each f_i is irreducible in the polynomial ring $K(u_1, \ldots, u_n)[x_1, \ldots, x_i]/(f_1, \ldots, f_{i-1})$. If the system is irreducible and $prem(g, f_1, \ldots, f_n) = 0$, the following can be shown [4]

1. The geometric statement (S) is generally true in all fields
2. For all fields, $(h_1 = 0, \ldots, h_n = 0, I_1 \neq 0, \ldots, I_n \neq 0) \Rightarrow g = 0$ where the I_k are the initials of the f_k.

So given an irreducible system, $prem(g, f_1, \ldots, f_n) = 0$ gives us a sufficiency condition for the validity of the theorem in question. For the converse, we first define

$$P = \{\ g \mid g \in K[u, x] \text{ and } prem(g, f_1, \ldots, f_n) = 0\}$$

Given an irreducible system, $prem(g, f_1, \ldots, f_n) = 0$ is a necessary condition for (S) to be generally true if one of the following is satisfied

1. $V(P)$ is of degree d.
2. E is an algebraically closed field.
3. The system $f_1, \ldots, f_n$ has a generic point in E.

The first question that arises is testing for irreducibility. If it turns out that $deg(f_i, x_i) = 1$ for all i, then the triangular system is trivially irreducible and real generic points trivially exist. In this case, $prem(g, f_1, \ldots, f_n) = 0$ gives a necessary and sufficient condition to determine whether the theorem is generally true or not. Several of the simpler theorems in the collection solved by Chou's prover fall in this category ([4], p. 56). If the degree of a polynomial is not 1 in its primary variable, Chou's prover uses factoring over extension fields to check for irreducibility. This approach, however, has been implemented only for *quadratic* extension fields. If the degree of some f_i in x_i is greater than 2, Chou's prover can not handle this case (unless $i = 1$ in which case only factoring over $\mathbb{Q}$ is required). Further, even in the case of quadratic polynomials, the prover works in general only over *algebraically closed fields*. For the prover to work over the *reals*, the existence of a real generic point needs to be shown as required by the sufficiency condition given above. As far as we know, a mechanical approach to this has not been implemented in Chou's prover ([4], p. 56).

Next, we come to the question of what happens when the triangular system is reducible. If this happens, in general Chou's prover uses Ritt's decomposition algorithm to obtain all irreducible components of the original variety. This also involves factoring over field extensions and is computationally expensive for high degrees. Again, this has been implemented only for quadratic cases.

Finally, we address the question of successive pseudo-division. As shown in the earlier section, we have the following remainder formula

$$I_1{}^{s_1} \cdots I_n{}^{s_n} g = Q_1 f_1 + \cdots + Q_n f_n + R_0$$

If the remainder $R_0 \neq 0$ then we can conclude that the theorem is not generally true since the right side would be non-zero at any instance of the theorem implying $g \neq 0$ at that instance. If, however, the remainder is zero, g could still be non-zero unless $I_j \neq 0$ for all j. These then become the additional subsidiary hypotheses under which the theorem is true. It could be true however that $prem(I_j, f_1, \ldots, f_{j-1}) = 0$ in which the hypotheses would become inconsistent. To avoid this situation, in general, one has to use the complete method in Chou's prover which involves triangulation using the Ritt decomposition algorithm.

To summarize, Chou's prover, though efficient and applicable in the case of a large class of geometric theorems, has the following limitations :

- Applicable in general to systems of degree quadratic or less.
- Does not work in all cases over $\mathbb{Q}$.
- Uses significant amount of computational resources for some problems of degree 2 ([4], p. 69).

4 Overview of the Theorem Prover

We will give a schematic overview of our theorem prover in this section. The notation used in this and subsequent sections is the same as introduced earlier. The prover takes as input a set of hypotheses polynomials $h_1, \ldots, h_n$ and a conjecture or conclusion polynomial g in $\mathbb{Q}[u_1, \ldots, u_m, x_1, \ldots, x_n]$. Here, as before, the x_i's represent the dependent variables and the u_j's, the independent variables. In what follows, we will abuse notation and write the hypotheses as polynomials over the x_i's only. It returns "true" or "false" according to whether the theorem is generally true or not over the reals. The prover can also be used to work over $\mathbb{C}$. The modules of the prover can be summarized as follows :

- **Random instantiation of parameters** The parameters u_i are set to random values to get the hypotheses as polynomials in $x_1, \ldots, x_n$ only. (See the implications of this step for the reals in Section 7).
- **Triangulation** The hypotheses are first triangulated using the triangulation algorithm described earlier to obtain the polynomials

$$f_1(x_1), f_2(x_1, x_2), \ldots, f_n(x_1, \ldots, x_n)$$

.

- **Root representation for triangular systems** This module takes as input the triangular system

$$f_1(x_1) = f_2(x_1, x_2) = \ldots = f_n(x_1, \ldots, x_n) = 0$$

and returns a *symbolic* representation of the roots of the system : the output is a univariate polynomial $p(s)$ and rational functions $r_1(s), \ldots, r_n(s)$. If the roots of $p(s) = 0$ are $\alpha^{(i)} \in \mathbb{C}$, for $i = 1, \ldots, N$, then

$$\xi^{(i)} = r(\alpha^{(i)})$$

One can think of $r(s)$ as a parametric curve in $\mathbb{C}^n$ which passes through all the roots $\xi^{(i)}$, $i = 1, \ldots, n$. The values of s at which it passes through a solution of $f_1 = f2 = \ldots = f_n$ are precisely the roots of $p(s) = 0$. The fact that we have a symbolic representation of the roots allows us to determine correctly the feasibility of the conjecture polynomial given the hypotheses polynomials.

- **Substitution** We put $x_i = r_i(s)$, $1 \leq i \leq n$, in the conjecture polynomial g. This reduces g to a univariate polynomial $g(s)$.

- **Sign Determination** This module determines the sign, i.e. +, - or 0, of the instantiated conjecture $g(s)$ at the real roots of $p(s) = 0$ using Sturm sequences.

- **Theorem proving or refutation :** The conjecture should vanish at all of the common real roots of the hypotheses polynomials. Thus the sign determination module should return 0 for the sign of $g(s)$ at every root. If not, there exists a root where the conjecture does not hold and is therefore false OR the triangulation process introduced extra roots. Section 6 discusses what to do in that case.

5 An Algorithm for Solving Triangular Systems

This section describes the basic algorithm for solving triangular systems used in our geometric theorem prover. Section 5.1 sketches the algorithm from a theoretical viewpoint, section 5.2 goes into more details of the algorithm especially the use of straight-line programs. We conclude with section 5.3 which gives a complexity analysis of the algorithm. Complete details on the algorithm including implementation issues can be found in the full version of the paper [11]. It should be noted that our algorithm for triangular systems is probabilistic - this arises from the fact that we use straight-line programs for representing the rational functions which make up the coefficients of the polynomials generated in the algorithm. However, most algorithms for polynomials require us to verify that the leading coefficient is not identically zero. This is done by giving random values to the variables in the corresponding SLP and evaluating the SLP at those values.

5.1 Theoretical Algorithm

The input to the algorithm is a system of triangular polynomial equations in $\mathbb{Q}(x_1, \ldots, x_n)$,

$$f_1(x_1) = f_2(x_1, x_2) = \cdots = f_n(x_1, \ldots, x_n) = 0$$

Let $\xi^{(i)} \in \mathbb{C}^n$ denote the solutions to the given system.

As described earlier, the algorithm returns a *symbolic* representation of the roots of the system i.e. the output is a univariate polynomial $p(s)$ and rational functions $r_1(s), \ldots, r_n(s)$. If the roots of $p(s) = 0$ are $\alpha^{(i)} \in \mathbb{C}$, for $i = 1, \ldots, n$, then

$$\xi^{(i)} = r(\alpha^{(i)})$$

Our algorithm is based on an approach for computing (p, r) for more general systems due to Renegar [14]. We use his basic method but take advantage of certain properties of triangular systems.

The algorithm proceeds iteratively; at the i^{th} step it adds the polynomial f_i and eliminates the extra variable x_i introduced by the polynomial to get a rational univariate representation $r_1(s_i), \ldots, r_n(s_i)$ of the roots of the first i polynomials. More precisely, at the start of the i^{th} step, the algorithm has computed

$$r_1(s_{i-1}), \ldots, r_{i-1}(s_{i-1}) \text{ and } p_{i-1}(s_{i-1})$$

The algorithm now introduces $f_i(x_1, \ldots, x_i)$. It sets $x_1 = r_1, \ldots, x_{i-1} = r_{i-1}$ in f_i. For the new variable, x_i, the algorithm sets,

$$x_i = \frac{s_i - l(\mathbf{x})}{l_i} = \frac{s_i - l_0 - l_1 x_1 - \cdots - l_{i-1} x_{i-1}}{l_i}$$

where $x_1, \ldots, x_{i-1}$ are instantiated to $r_1, \ldots, r_{i-1}$ respectively. Thus we obtain from f_i a rational function g_i in s_{i-1} and s_i. We denote by g_{i_n} the numerator polynomial of g_i and by g_{i_d} the denominator.

The algorithm computes the (univariate) Sylvester resultants $R(g_{i_n}, p_{i-1})$ and $R(g_{i_d}, p_{i-1})$ with respect to the variable s_{i-1}. We have :

Lemma 5.1 *$R(g_{i_d}, p_{i-1})$ divides $R(g_{i_n}, p_{i-1})$ exactly.*

Proof Given below.

Thus we can set

$$R = \frac{R(g_{i_n}, p_{i-1})}{R(g_{i_d}, p_{i-1})}$$

If the polynomial $l(\mathbf{x}) = l_0 + l_1 x_1 + \cdots l_{i-1} x_{i-1}$ is specialized to a random linear polynomial L, then with probability one, the values $l(\xi^{(j)})$ will be distinct for distinct $\xi^{(j)}$. The roots of the resultant R will also be distinct. The algorithm sets $p_i(s_i) = R$.

We obtain the functions r_j by differentiating R. We define r_j as

$$r_j(s_i) = \left. \frac{\left(\frac{dR}{dl_j}\right)}{\left(\frac{dR}{dl_0}\right)} \right|_{l=L} \qquad j = 1, \ldots, i$$

It can be verified that the r_j's have the correct values at $s_i = \alpha^{(k)}$.

At termination the algorithm eliminates all the variables $x_1, \ldots, x_n$ and returns rational functions $r_j(s)$ in the variable $s = s_n$ and the polynomial $p(s) = p_n(s_n)$. (Note : for the first step, the algorithm only needs to set $x_1 = (s_1 - l_0)/l_1$ in f_1 to get $R = p_1(s_1)$, i.e. $p_0 \equiv 0$.)

Sketch of proof (Lemma 5.1) The proof is based on the following well-known property of resultants - the resultant of $n+1$ polynomials f_i, $(i = 1, \ldots, n+1)$, with the first n polynomials having common roots $\xi^{(j)}$, $j = 1, \ldots, N$, is given by $\prod_{j=1}^{N} f_{n+1}(\xi^{(j)})$. In particular if $f_{n+1} = s - l_0 - l_1 x_1 - \cdots - l_n x_n = s - l(\mathbf{x})$, then the resultant of the f_i's is $R(s,l) = \prod_{j=1}^{N}(s - l(\xi^{(j)}))$. Further, let $r_{in}(s,l) = (dR/dl_i)$ and $r_d(s,l) = (dR/dl_0)$ and let $\xi^{(j,k)}$ denote the k^{th} coordinate of the j^{th} root. Then we have

$$r_{in}(s,l) = \frac{dR}{dl_i} = \sum_{j=1}^{N}(s - l(\xi^{(1)})) \cdots$$

$$(s - l(\xi^{(j-1)}))(-\xi^{(j,i)})(s - l(\xi^{(j+1)})) \cdots (s - l(\xi^{(N)}))$$

It follows from the above property that the resultant of R and $r_{in}(s,l)$ w.r.t. s is (by evaluating $r_{in}(s,l)$ at the roots of R)

$$\prod_{j=1}^{N}(l(\xi^{(j)}) - l(\xi^{(1)})) \cdots (l(\xi^{(j)}) - l(\xi^{(j-1)}))(-\xi^{(j,i)})$$

$$(l(\xi^{(j)}) - l(\xi^{(j+1)})) \cdots (l(\xi^{(j)}) - l(\xi^{(N)}))$$

Similarly the resultant of R and $r_d(s,l)$ can be shown to be

$$\prod_{j=1}^{N}(l(\xi^{(j)}) - l(\xi^{(1)})) \cdots (l(\xi^{(j)}) - l(\xi^{(j-1)}))(-1)$$

$$(l(\xi^{(j)}) - l(\xi^{(j+1)})) \cdots (l(\xi^{(j)}) - l(\xi^{(N)}))$$

Thus we have

$$\mathit{Resultant}(R, r_d) \mid \mathit{Resultant}(R, r_{in}) \qquad (1)$$

To use these results in our algorithm, we note that the two step process, at the i^{th} step, of instantiation of f_i with the rational functions and computing the resultants $R(g_{in}, p_{i-1})$ and $R(g_{id}, p_{i-1})$ corresponds exactly to computing the resultant of $f_1, \ldots, f_i$ and $s - l_0 - l1x1 - \cdots - l_i x_i$, since the input system of equations was triangular. Finally, to complete the proof, we apply the same approach as we did above for showing (1) to $R(g_{in}, p_{i-1})$ and $R(g_{id}, p_{i-1})$. □

5.2 Algorithm Details

We now give a more complete description of the algorithm based on the theoretical version given in section 5.1. The algorithm performs all its computations representing the polynomials involved as *univariate* polynomials in the *primary variable* with coefficients which are rational functions in the remaining variables. These coefficients are represented by *straight-line programs* (see 8.1 for a summary of SLP properties; [12] discusses issues with SLP computations in greater detail).

As mentioned in 5.1, the first step of the algorithm corresponds to simply computing the resultant of f_1 and $s - l_0 - l_1 x_1$ by setting $x_1 = (s - l_0)/l_1$ in f_1 and then taking derivatives of this resultant w.r.t l_0 and l_1 to get the rational function representation for x_1. The i^{th} step of the algorithm proceeds as follows :

1. Instantiation At the start of the i^{th} step the algorithm the algorithm has computed

$$r_1(s_{i-1}), \ldots, r_{i-1}(s_{i-1}) \text{ and } p_{i-1}(s_{i-1})$$

These are represented as quotients of polynomials in s_{i-1}, the primary variable for the i^{th} iteration (the coefficients of these polynomials are simply numbers as will be seen at the last step of the loop). The algorithm sets

$$x_i = \frac{s_i - l_0 - l_1 x_1 - \cdots - l_{i-1} x_{i-1}}{l_i}$$

with $x_1, \ldots, x_{i-1}$ instantiated to the rational functions r_j. Thus, all of these rational functions including the one for x_i are represented as quotients of univariate polynomials in s_{i-1} whose coefficients are rational functions in s_i and the l variables and represented by SLP's.
2. Substitution The rational functions are substituted in the input polynomial f_i. We therefore get f_i as a rational function which is a quotient of polynomials in s_{i-1}.
3. Resultant computation. The resultants of the numerator and the denominator of the instantiated f_i with the previous resultant p_{i-1} are now computed w.r.t s_{i-1}. These two computations are identical to univariate computations since the cost of doing SLP arithmetic is constant time per operation. The output of this step will be the two resultants. However, since the resultant

is a polynomial in the coefficients of the two input polynomials, the resultants so obtained will be both SLP's in s_i and the l_k's.
4. Division We divide the two resultants in SLP form by simply creating a divide node with the two SLP's as children. Call this resultant R.
5. Differentiation To compute the rational functions r_j required for the next step, we need to compute the derivatives of R w.r.t. the l_k's. We do this by differentiating the SLP corresponding to R with respect to each variable as described in section 8.1. We now have a collection of SLP's in s_i and the l_k's.
6. Random instantiation We set each l_k to a random value, to get the random hyperplane corresponding to the L. Our collection of SLP's is now defined only over s_i.
7. Interpolation For the start of the next iteration, we need the resultant $R = p_i$ and the numerators and denominator of the rational functions r_j as *polynomials* in s_i. To do this we interpolate the SLP's representing them into polynomials over s_i. This can be done with straight-forward univariate polynomial interpolation. The degree of the polynomial an SLP represents is obtained by a single depth-first traversal of the SLP. The value of the polynomial at $s_i = q \in \mathbb{Q}$ is obtained by simply evaluating the SLP with s_i set to q, an operation also requiring a depth-first traversal of the SLP. At the end of this process we have the p_i and the numerators and denominator of the r_j's as polynomials in s_i as required by the first step.

5.3 Complexity Analysis

We now analyze the time and space requirements of the algorithm in 5.2. We will look at the final loop of the algorithm (when x_n is eliminated) since that step dominates the time and space complexity. Note that the space complexity corresponds to the number of operations we will perform over coefficients of univariate polynomials, since instead of actually performing these operations we store them in SLP's.

Let d_i be the total degree of f_i. Then it is straightforward to show that the **degree of the final resultant** $p_n(s_n)$ will be $D = d_1 \cdots d_n$. As explained in 5.2, Steps 1 and 2 in the algorithm (instantiation and substitution) have the same complexity as substitution with univariate polynomials. The polynomials we substitute with (i.e. the numerators and denominator of the r_j's) have degree $d_1 \cdots d_{n-1}$ in s_{n-1}. The time and space complexity of these steps is therefore $\mathcal{O}((d_1 \cdots d_n)^2)$. The resultant computation now takes as input two polynomials with degree $\leq D$ and the time and space required is therefore $\mathcal{O}((d_1 \cdots d_n)^2)$. The next step, differentiation takes as input the resultant SLP obtained in the previous step which has size $\mathcal{O}((d_1 \cdots d_n)^2)$. The worst case while differentiating arises when we encounter a division node which gives rise to 5 SLP nodes and we have to take derivatives w.r.t $n+1$ variables. Thus the time required is $\mathcal{O}(5(n+1)(d_1 \cdots d_n)^2)$ and space required is $\mathcal{O}(5(d_1 \cdots d_n)^2)$ (there are $n+1$ derivatives we need to take but we can by perform the differentiation and interpolation w.r.t. a given l_k one after the other and then re-use the space for the next l_k).

Finally we interpolate these SLP's. Each SLP has to be evaluated a number of times equal to the degree of the polynomial (in s_n) it represents. This degree being D, it follows that the time spent in evaluating an SLP's is $\mathcal{O}((d_1 \cdots d_n)^3)$ and since there are $n+1$ SLP's, the total time for interpolation is $\mathcal{O}(n(d_1 \cdots d_n)^3)$ The space required for univariate interpolation is linear in the degree of the input using any standard interpolation algorithm ([19]) and we can re-use the space from the previous interpolation as explained before. Hence the space required for interpolation is $\mathcal{O}((d_1 \cdots d_n))$. We have n such iterations of the algorithm, however the space required in one iteration can be re-used in the other since once the SLP's have been interpolated we no longer need the space to represent them. From this analysis it follows that :

Theorem 5.2 *The (probabilistic) time complexity of the above algorithm for determining the root representation of a triangular system is $\mathcal{O}(n^2(d_1 \ldots d_n)^3)$ where d_i is the total degree of f_i. i.e. $\mathcal{O}(n^2 d^{\mathcal{O}(n)})$ where d is an upper bound on each d_i. The space required is $\mathcal{O}((d_1 \ldots d_n)^2)$ or $\mathcal{O}(d^{\mathcal{O}(n)})$*

In contrast the worst-case complexity of computing characteristic sets as required by the original algorithm of Ritt with subsequent modifications by Wu is non-elementary. However, it was shown in [6] that the degree of the polynomials in a characteristic set for a zero-dimensional ideal $I \subseteq \mathbb{Q}[x_1, \ldots, x_n]$ is $\mathcal{O}(d^{\mathcal{O}(n)})$ and the complexity of computing these characteristic sets is $\mathcal{O}(n^{3.376} d^{\mathcal{O}(n^3)})$. (The complexity of the algorithm used by Chou's prover, as far as we know, is not known). Thus, at even relatively small degrees, $d \geq 4$, the computation required by the modified Wu's method can be rather expensive. On the other hand, our algorithm is probabilistic as mentioned in the introduction to section 5.

6 Sign Determination and Theorem Proving

Once we have a symbolic representation (p, r) of the roots of the triangular system of hypotheses, it is relatively straightforward to determine whether the theorem is true or not. The first step is to reduce the conjecture polynomial c to a univariate one by substituting $x_i = r_i(s)$, $1 \leq i \leq n$. This reduces c to a univariate polynomial $g(s)$. We now need to compute the sign of $g(s)$ at the real roots of $p(s)$. We can use Sturm sequences to do that as follows :

If $f(s)$ and $g(s)$ are polynomials, let the Sturm sequence of f and g be denoted $r_0, r_1, \ldots, r_k$, where $r_0(s) = f(s)$, $r_1(s) = g(s)$, and the intermediate remainders are computed via $r_{i+1} = q_i r_i - r_{i-1}$. where q_i is the pseudo-quotient of the polynomial division of r_{i-1} by r_i. For a real value v, let $\mathrm{SA}(f, g, v)$ denote the number of changes in sign in the sequence $r_0(v), r_1(v), \ldots, r_k(v)$ and $\mathrm{SC}(f, g)$ denote $\mathrm{SA}(f, g, +\infty) - \mathrm{SA}(f, g, -\infty)$. The classical Sturm theorem states that $\mathrm{SC}(f, f')$ equals the number of real roots of $f(s)$.

We now compute $r(s) = p(s)/(gcd(p(s), g(s)))$. Assuming we didn't introduce any extra roots in the triangulation procedure, we have the following conditions

- For theorem proving over the $\mathbb{C}$, the theorem is true iff r(s) is identically 1.
- For theorem proving over the reals, if r has no real roots then the theorem is true. However, if r does have real roots we need to check if those roots correspond to degenerate conditions, i.e. where the u_j's are algebraically dependent. and this is discussed in section 7. For the sufficiency condition, we compute $\mathrm{SC}(r, r')$. Assuming r is square-free, if r has no real roots, $\mathrm{SC}(r, r')$ will be 0. Section 7 discusses how to get a necessary condition on r.

We could have the problem discussed earlier of having introduced extra roots in the triangulation procedure. To check for this eventuality, we instantiate the original hypotheses polynomials h_j with the rational functions $r_i(s)$ as we did for g. We can now do sign-determination for the system $h_1(s), \ldots, h_n(s)$ at the roots of $r(s) = 0$ using the algorithm of [2]. This procedure yields for each root of $r(s)$ an ordered sign sequence in $\{+, -, 0\}^{n+1}$ where the i^{th} element in the sign sequence corresponds to the sign of the i^{th} polynomial in $h_1(s)$, $\ldots, h_n(s)$ at that root of $r(s)$. Thus we can find out if the any of the roots of $r(s)$ are actual roots of the original system. If none are then we are done and the theorem is true. Else, over $\mathbb{C}$ the theorem is false. Over $\mathbb{R}$ we need to use the necessary condition as discussed in section 7.

7 Genericity and Completeness

We now turn our attention to the implications of the fact that we are actually determining the feasibility of conjectures based on hypotheses defined in $K[u_1, \ldots, u_d, x_1, \ldots, x_n]$ rather than just $K[x_1, \ldots, x_n]$. In the latter case, the algorithms given above work in a straightforward manner – one uses simple triangulation followed by the algorithm for triangular systems and then that for sign determination. In the former case, things are not quite as nice. In the algorithm given in section 4, we eliminate the u_i's in step 1 by setting them to random values. This approach works when computing over $\mathbb{C}$ since with probability 1 we will not run into a degenerate situation. However, when the field is not algebraically closed, the situation becomes complicated. Consider, for example, the polynomial equation $x^2 + u^2 - 1 = 0$. For values of u such that $-1 \leq u \leq 1$, the equation has real solutions for x. However for other values of u there are no real solutions. If u were a parameter and the equation was an input hypothesis polynomial, random instantiation for u could lead to a wrong conclusion about the validity of the conjecture over the reals.

In order to disprove conjectures over the reals, we need to ensure the existence of a *real generic point*. This can be done as follows : we run the same algorithms as before (i.e. triangulation and triangular system solution) but instead of random instantiation for the u_j's we do our computation over $K(u_1, \ldots, u_d)$. The coefficients of the various polynomials will be rational functions in $\mathbf{u}$ and are represented by straight-line-programs. At the end of this process we obtain $r(s)$ as in section 6 except that r is in $K[\mathbf{u}, s]$ instead of $K[s]$. Now we need to find those components of the variety of r on which the u_j's are algebraically independent - if there are any, then we know that a real generic point exists and the theorem is generically false.

To get this necessary condition we do the following : First, we compute the resultant of $r(s, \mathbf{u})$ and $dr(s, \mathbf{u})/ds$ with respect to s. Computing the resultant involves determining the Sturm sequence of the two polynomials. The resultant is obtained as a polynomial in $\mathbf{u}$, say $R(\mathbf{u})$. Geometrically, this operation corresponds to projecting the variety $V \subseteq \mathbb{R}^{d+1}$ of $r(s, \mathbf{u})$ along the s direction to get the variety $W \subseteq \mathbb{R}^d$ of $R(\mathbf{u})$. Consider now the connected components of the complement of W in $\mathbb{R}^d$. Note that these components are necessarily open sets in $\mathbb{R}^d$. Any point in one of these connected components corresponds to a selection of the parameters $\mathbf{u}$. For each such selection, we get a certain number of solutions in s for $r(s, \mathbf{u})$. The observation is that the number of solutions remains constant within every connected component of W^c, the complement of W. (Geometrically, this corresponds to looking at the fiber above a point in that connected component). Therefore every connected component can be represented by a "witness point" which describes the behavior of the entire component. Further the fiber for each such witness point tells us what the behavior of the variety V is over that component. If the variety has a real zero in the fiber of that witness point then we know that all values of $\mathbf{u}$ in that connected component give rise to real zeroes. This means that a generic real solution exists for r and the necessary condition would imply that the theorem was false. If no connected component has a real zero in the fiber of its witness point then all the real solutions of r correspond to degenerate conditions on $\mathbf{u}$.

Now all we need to do is find the witness points and check if they correspond to real zeroes : The complement of W can be represented by the polynomial condition : $R' = (R+\epsilon)(R-\epsilon)$ where ϵ represents an infinitesimal variable. We can now compute the witness points in each connected component of W^c as follows : We choose a linear map $\pi : \mathbb{R}^d_{|W^c} \to \mathbb{R}$ and find all the critical points of this map. Let $u'_2, \ldots, u'_d$ be a coordinate basis for the kernel of π which is a $d-1$ dimensional linear space. We now solve the following system in the limit as $\epsilon \to 0$:

$$R' = 0 \qquad \frac{\partial R}{\partial u'_2} = \cdots = \frac{\partial R}{\partial u'_d} = 0 \tag{2}$$

The algorithm for solving this system is described in [1] and it involves computing the u-resultant of the system, arranging it in powers of ϵ and retaining the lowest degree coefficient. The result is (as in our triangular system algorithm) a polynomial H(t) and rational functions $(h_1(t), \ldots, h_d(t))$ describing the witness points in the connected components as required. Given this description, we need to figure out the behavior of the variety V at these witness points. To do that, we use the Sturm sequence which was computed to get the resultant $R(\mathbf{u})$. This is a sequence of polynomials $p_1(s), \ldots, p_k(s)$ with coefficients which are polynomials in $\mathbf{u}$. As per the Sturm theorem the number of real roots of r is given by $\mathrm{SC}(r, r')$, that is, we look at the leading coefficients and determine the sign variations. In our case, we are interested in the number of real roots at a given witness point as obtained above - so we instantiate the leading coefficients $l_1(\mathbf{u}), \ldots, l_k(\mathbf{u})$ of the polynomials $p_1(s), \ldots, p_k(s)$ with $u_i = h_i(t)$ to get polynomials $l_1(t), \ldots, l_k(t)$. The signs of these polynomials at the roots of the primitive element polynomial $H(t)$ are exactly the signs of these polynomials at the witness points. We now apply the sign determination algorithm of [2] to determine the signs of the leading coefficients of the p_i's at the witness points. Each sign sequence gives us the number of roots in the fiber corresponding to that witness point. As mentioned earlier, those connected components which have no real roots in the fiber for their witness points correspond to degenerate values of $\mathbf{u}$ for which the hypotheses are not valid. If however a component does have real roots, then we have an open set of $\mathbf{u}$ values on which the u_j's are independent and and hence the theorem is false. It can be shown that the worst case complexity of the above procedure is $\mathcal{O}(d^{\mathcal{O}(n^2)})$. This algorithm has *not* been implemented in our prover yet - more general algorithms for determining whether a semi-algebraic set is not empty will be implemented in our toolkit in the near future [13].

8 Implementation

In this section, we briefly describe a first implementation of the above algorithms. Most of the modules are part of a larger project involving the development of a toolkit for solving problems in non-linear algebra. (This toolkit will soon be available in the public domain). The basic modules perform polynomial arithmetic over different fields such as finite fields, rationals, infinitesimal extensions, "mixed" arithmetic and most importantly straight-line programs or SLP's.

8.1 Straight-line Programs

All our computations are done over the field of arithmetic straight-line programs. An SLP can be represented by a directed acyclic graph, with each node representing an operation such as addition, and a value. *All the polynomials are thus represented as polynomials over the field of SLP's, i.e. the coefficients are straight-line programs and not numbers.* As mentioned in the introduction, this allows us to represent coefficients which are rational functions in the parameters $u_1 \ldots u_m$. In other words the extension field $\mathbb{Q}(u_1, \ldots, u_m)$ has a 1-1 correspondence with SLP's defined over $u_1, \ldots, u_m$.

Arithmetic with SLP's is easy - for example to add two SLP's one creates a new node with the operation "+" and two edges directed from the new node to the two operand nodes. The value of an SLP node (over the base field such as $\mathbb{Q}$), can be obtained by simply adding the values of its children nodes. Thus evaluation involves a depth-first search of the SLP graph and can be done in time linear in the size of the SLP.

Computing derivatives as required by the algorithm in Section 5 is particularly straightforward, as is determination of the signs of the coefficients of various polynomials (required by the Sturm sequence algorithm). To compute the derivative SLP of an SLP node one simply creates a new SLP using the usual rules of differentiation. This has roughly double the size of the original SLP. For sign determination, one uses successive differentiation and evaluation of SLP's. The details of how this works can be found in the complete version of the paper.

8.2 Some benchmarks

The table below gives some benchmarks on some standard problems. A more complete set can be found in [11]. All the benchmarks were run on a Sun Sparc 10 machine. The code is written in C. These benchmarks were obtained from a first implementation and we expect to reduce the running time by using various optimizations but the initial benchmarks seem to be rather promising. (The times given for Chou's prover are from

[4] and were obtained on a Symbolics 3600. The geometric statements for these theorems are also from [4].)

Geometric Theorem	Total deg	No. of vars	Chou's Prover	This Prover
Pappus	1	7	1.52s	0.11
Pappus Dual	1	7	1.45s	0.12
AMS-1	1	13	4.05s	0.33s
Pascal Conic-1	1	15	14.43s	0.56s
Simson	2	11	-	0.47s
Ptolemy	8	6	-	2.9
Steiner	8	17	-	33.0

9 Code Availability and Acknowledgements

The code for the prover and the toolkit described in [13] as well as papers on the same can be obtained by anonymous ftp from *robotics.eecs.berkeley.edu*. See the file pub/rege/README for details.

I would like to thank my advisor John Canny for directing me to this problem and for some very useful and insightful discussions regarding this and other topics. I would also like to thank Phil Liao for providing me with the theorems in the form required by the code and also both him and Richard Fateman for useful discussions.

10 References

[1] J.F. Canny. Computing roadmaps of general semi-algebraic sets. In *AAECC-91*, 1991. New Orleans.

[2] J.F. Canny. An improved sign determination algorithm. In *AAECC-91*, 1991. New Orleans.

[3] S.-C. Chou. Proving elementary geometry theorems using Wu's algorithm. In W.W. Bledsoe and D.W. Loveland, editors, *Theorem Proving : After 25 Years*. American Mathematical Society, 1984.

[4] S.-C. Chou. *Mechanical Geometry Theorem Proving*. Mathematics and its applications. D. Reidel, Holland, 1988.

[5] J. Little D. Cox and D. O'Shea. *Ideals, Varieties, and Algorithms*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 1992.

[6] G. Gallo and B. Mishra. Efficient algorithms and bounds for Wu-Ritt characteristic sets. In T. Mora and C. Traverso, editors, *Effective Methods in Algebraic Geometry*, volume 94 of *Progress in Mathematics*, pages 119–142. Birkhäuser, 1991.

[7] H. Gelernter. Realization of a geometry theorem proving machine. In E.A. Feigenbaum and J.E. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, 1963.

[8] D. Kapur. Geometry theorem proving using Hilbert's Nullstellensatz. In *Symposium on Symbolic and Algebraic Computation*, pages 202–208, 1986. Waterloo, Ontario.

[9] D. Kapur. A refutational approach to geometry theorem proving. In D. Kapur and J.L. Mundy, editors, *Geometric Reasoning*. MIT Press, 1988.

[10] B. Kutzler and S. Stifter. On the application of Buchberger's algorithm to automated geometry theorem proving. *J. Symbolic Comput.*, 2:409–420, 1986.

[11] A. Rege. A complete and practical algorithm for geometric theorem proving. Manuscript for journal submission and Tech Report (UC Berkeley), 1995.

[12] A. Rege and J. Canny. Straight-line programs for real field extensions and perturbations. To be submitted to special issue of *International Journal of Computational Geometry and Applications*, 1995.

[13] A. Rege and J. Canny. A toolkit for algebra and geometry. Submitted to the *International Symposium on Symbolic and Algebraic Computation*, Montreal, 1995.

[14] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I, II and III. Technical Report 852,855,856, Cornell University, Operations Research Dept., 1989.

[15] J.F. Ritt. *Differential Equations from the Algebraic Standpoint*. Number 14 in AMS Colloquium publications. American Mathematical Society, New York, 1932.

[16] J.F. Ritt. *Differential Algebra*. Number 33 in AMS Colloquium publications. American Mathematical Society, New York, 1950.

[17] W. Wu. On the decision problem and mechanization of theorem proving in elementary geometry. *Sci. Sinica*, 21:150–172, 1978.

[18] W. Wu. Some recent advances in mechanical theorem proving of geometries. In W.W. Bledsoe and D.W. Loveland, editors, *Theorem Proving : After 25 Years*. American Mathematical Society, 1984.

[19] R. Zippel. *Effective Polynomial Computation*. Kluwer Academic publishers, Massachusetts, 1993.