COMPLEXITY AND APPLICATIONS OF PARAMETRIC ALGORITHMS OF COMPUTATIONAL ALGEBRAIC GEOMETRY

MAREK RYCHLIK*

Abstract. This article has two main goals. The first goal is to give a tutorial introduction to certain common computations in algebraic geometry which arise in numerous contexts. No prior knowledge of algebraic geometry is assumed. The second goal is to introduce a software package, called *CGBlisp* which is capable of performing these computations. This exposition is enhanced with simple examples which illustrate the package's usage. The package was developed as a tool to prove a particular theory in billiard theory, but its scope is very general, as our examples demonstrate. All examples of computations with *CGBLisp* discussed in this paper are included in the distribution of *CGBLisp*.

Key words. Algebraic geometry, geometric theorem proving, billiards, parametric equations, Gröbner basis software.

AMS(MOS) subject classifications. Primary 68Q40, 14Q15, 13P10.

1. Notation. In the current article we will discuss algebraic sets and varieties. Variables of various polynomials and functions will be denoted by $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. We will mostly deal with parametric problems concerning algebraic sets and varieties. Thus conceptually it will be convenient to designate some variables to be parameters and distinguish them from "regular" variables. In the sequel, generic parameters will be denoted by $\boldsymbol{u} = (u_1, u_2, \ldots, u_m)$. By $R = k[\boldsymbol{x}]$ we will denote the ring of non-parametric polynomials with coefficients in the ring k. Furthermore, our main concern is with algorithms which can actually be implemented on a computer. Thus we will assume that k is a *computable ring*, i.e., that all its elements can be represented on a digital computer and all ring operations can be effectively computed using algorithms which terminate in finite time. However, we assume that our computer, although finite, can be arbitrarily large. We note that $k = \mathbb{Z}, \mathbb{Q}$ and $\bar{\mathbb{Q}} \subset \mathbb{C}$ are all computable rings but $\mathbb{R}$ is not. The field $\bar{\mathbb{Q}}$ is the algebraic closure of $\mathbb{Q}$. Let us consider a set of polynomials $F = \{f_1, f_2, \ldots, f_s\} \subseteq R$. The *ideal* spanned by F will be denoted by $I = \mathrm{Id}(F) = \{\sum_{j=1}^{s} a_j f_j : a_j \in R\}$. The *variety* associated with the ideal F is defined as $V(F) = \bigcap_{f \in F} f^{-1}(0)$.

For a given set $W \subseteq k^n$ we may consider the ideal generated by this set: $\mathrm{Id}(W) = \{f \in k[x] : f|W \equiv 0\}$. W does not have to be a variety but this formula always defines an ideal.

The generic ring of polynomials with parameters will be denoted by $S = k[\boldsymbol{u}, \boldsymbol{x}]$. By *specialization* of a set $F \subseteq S$, given a fixed element $\boldsymbol{a} \in k^m$, we mean the set $F_{\boldsymbol{a}} \subseteq R$ which is the result of substituting $\boldsymbol{u} = \boldsymbol{a}$ into F.

* *Department of Mathematics, University of Arizona, Tucson, AZ 85721.*

If $I \subset R$ is an ideal then the *radical ideal* of the ideal I is defined as follows

$$\sqrt{I} = \{f \in R : \exists n \geq 0\ f^n \in I\}. \tag{1.1}$$

An ideal I is called a *radical ideal* iff $I = \sqrt{I}$. We note that for any subset $W \subseteq k^n$ the ideal $\mathrm{Id}(W)$ is radical. The correspondence between ideals and varieties is not 1:1 in general. However, over an algebraically closed field $\mathrm{Id}(V(I)) = \sqrt{I}$ for every ideal I, and thus the correspondence between radical ideals and varieties is 1:1.

2. Parametric vs non-parametric problems. A generic non-parametric problem can be formulated as follows: given $F \subseteq R$, find the dimension, cardinality, degree, etc. of $V(F)$.

A generic parametric problem can be formulated in a similar way: given $F \subseteq S$, find the dimension, cardinality, degree, etc. of $V(F\boldsymbol{a})$ as a function of $\boldsymbol{a}$.

A solution of the parametric problem requires partitioning of the parameter space according to the value of a certain invariant (dimension, cardinality, degree, etc.). We note that this partition is into *constructible sets*; a set is called *constructible* if it can be represented in terms of *equations* ($f(\boldsymbol{u}) = 0$) and *inequations* ($f(\boldsymbol{u}) \neq 0$). This fact follows from *elimination theory*.

3. Monomial ordering. Constructive methods of algebraic geometry have recently developed into a mathematical discipline known as Computational Algebraic Geometry. The fundamental algorithm of this discipline is the *Buchberger algorithm* for calculating *Gröbner bases*. With various modifications, this algorithm is still at the heart of most Gröbner basis calculations.

In order to calculate a Gröbner basis (which will be defined later) we will need to linearly order all monomials with respect to a given set of variables. In this way, every multivariable polynomial will be somewhat similar to a single variable polynomial which is most naturally ordered by putting the monomials with a higher power of the variable before those with a lower power. Not every linear ordering of monomials is suitable for algebraic geometry calculations. Monomial ordering $\succ$ is *admissible* if $\succ$ is:

1. *total*
2. *compatible with multiplication:*

$$\forall \alpha, \beta, \gamma\ \boldsymbol{x}^\alpha \succ \boldsymbol{x}^\beta \Rightarrow \boldsymbol{x}^\alpha \boldsymbol{x}^\gamma \succ \boldsymbol{x}^\beta \boldsymbol{x}^\gamma$$

3. *well-ordering*: every set of monomials has the smallest element in the sense of $\succ$.

Most commonly used monomial orderings assume some specific order of the variables. Examples of admissible monomial orders include:

lexicographic (lex) First we order variables, say $x_1 \succ x_2 \succ \cdots \succ x_n$. A monomial $\boldsymbol{x}^\alpha \succ \boldsymbol{x}^\beta$ if the first slot on which α and β differ is larger in α.

graded lexicographic (grlex) For monomials of equal *total degree*, i.e., the sum of the powers of the variables, possibly taken with weights, this order is exactly the same as the lexicographic order. Otherwise, the monomial with the higher total degree precedes the polynomial with the lower total degree.

graded reverse lexicographic (grevlex) This order is considered the best choice for most situations. Similarly to the **grlex** order, a polynomial with the higher total degree is bigger. However, if the degrees are equal, we consider the monomial which is *smaller* in the lexicographic order to be *bigger* in **grevlex** order. In addition, we reverse the order of the variables in the lexicographic order, i.e., we make comparisons of the powers of the *last* variable first.

Let $f = \sum_m a(m)m$, where m is a monomial and $a(m)$ is a corresponding coefficient. Thus $m = \boldsymbol{x}^\alpha = \prod_{j=1}^n x_j^{\alpha_j}$ where $\alpha \in \mathbb{Z}^n$ is a multi-index. The sum is considered ordered in the order of decreasing monomials. We may define the *leading coefficient* of f, denoted by $LC(f)$, the *leading monomial* of f, denoted by $LM(f)$, and the *leading term* of f, denoted by $LT(f)$. We have $LT(f) = LM(f) \cdot LC(f)$, where $LT(f)$ denotes the first term in the sum.

EXAMPLE 1. *Let $f = 3x^2y + xy^6$. The term $3x^2y$ is bigger in the lexicographic order but smaller in the graded lexicographic order than xy^6. We have $LM(f) = x^2y$, $LT(f) = 3x^2y$ and $LC(f) = 3$ with respect to the lexicographic order (***lex***). For two variables,* **grlex** *and* **grevlex** *are identical. This is probably the main reason to change the order of variables in the lexicographic comparison which is part of the definition of* **grevlex**.

4. The division algorithm. Once we have established what an admissible monomial order should be, we are able to take advantage of it by defining an algorithm for dividing a polynomial by another one. In view of the fact that every polynomial has a distinguished leading monomial, this algorithm looks very similar to *long division* known from algebra. However, the main step in making division useful is to define a division algorithm which will divide a single polynomial by a *family of polynomials*. The pseudo-code description of this algorithm is given in table 1. We note that the result depends on an admissible monomial ordering. The result of the division algorithm is the pair $\left((a_j)_{j=1}^s, r\right)$. The remainder has the property that none of its terms is divisible by any of the leading monomials of the family $F = \{f_1, f_2, \ldots, f_s\}$. It is clear that if $r = 0$ then f is in the ideal generated by F. However, the condition $r = 0$ is only sufficient and not necessary. Only when F is a Gröbner basis does this condition become necessary and sufficient.

TABLE 1
The division algorithm.

Input: $f_1, f_2, \ldots, f_s, f \in k[\boldsymbol{x}]$
Output: $a_1, a_2, \ldots, a_s, r \in k[\boldsymbol{x}]$ such that
$f = a_1 f_1 + a_2 f_2 + \ldots + a_s f_s + r$
for $i := 1$ **to** s **do**
  $a_i := 0$
$p := f$
while $p \neq 0$ **do**
  $i := 1$
  $flag :=$ **false**
  while $i \leq s$ **and** $flag =$ **false do**
    if $LT(f_i) \mid LT(p)$ **then**
      $a_i := a_i + LT(p)/LT(f_i)$
      $p := p - (LT(p)/LT(f_i))f_i$
      $flag :=$ **true**
    else
      $i := i + 1$
  if $flag =$ **false** then
    $r := r + LT(p)$
    $p := p - LT(p)$

5. The ideal membership problem and the division algorithm. Let us illustrate the division algorithm using the following example:

PROBLEM 1. *Let* $f = z^3 - y^4$. *Is* f *in the ideal* $\mathrm{Id}(\{f_1, f_2\})$ *where* $f_1 = y^2 - x^3$ *and* $f_2 = z - x^2$ *? (Note:* $x \prec y \prec z$, *the order is* **lex**.*)*

The sequence of calculations presented in table 2 corresponds to the steps of the division algorithm. Thus

$$f = (-y^2 - x^3)(y^2 - x^3) + (z^2 + zx^2 + x^4)(z - x^2)$$

and $f \in I$. We note that the division algorithm always produces quotients a_i with the property $LT(a_i f_i) \preceq LT(f)$.

6. Gröbner bases. The division algorithm may yield $r \neq 0$ even if $f \in I$. In fact, this is a common phenomenon. If F is a *Gröbner basis*, the condition $r = 0$ is sufficient and necessary for $f \in I$. A Gröbner basis is a set of polynomials G such that $LM(G) = LM(\mathrm{Id}(G))$, where $LM(F) = \{LM(F) : f \in F\}$ for every family F; equivalently, if $f \in \mathrm{Id}(G)$ then $LM(f)$ is divisible by $LM(g)$ for some $g \in G$. The *Hilbert Basis Theorem* implies that every polynomial ideal has a *finite* Gröbner basis. Gröbner bases are algorithmically constructed using *Buchberger algorithm* or its variations. Our next goal is to describe this algorithm.

The *S-polynomial* (or *syzygy polynomial*) of a pair of polynomials (f, g) is defined as follows: let $\boldsymbol{x}^{\boldsymbol{\gamma}} = LCM(LM(f), LM(g))$ be the least common

TABLE 2
An example of the division algorithm.

$$
\begin{array}{lll}
 & a_1 = -y^2 - x^3 & \\
 & a_2 = z^2 + x^2 z + x^2 & \\
f_1 = y^2 - x^3 & f = z^3 - y^4 & \\
f_2 = z - x^2 & \quad z^3 - x^2 z^2 & \\
 & \overline{\qquad\qquad} & \\
 & \quad -y^4 + x^2 z^2 & \\
 & \quad -y^4 + x^3 y^2 & \\
 & \quad \overline{\qquad\qquad} & \\
 & \qquad x^2 z^2 - x^3 y^2 & \\
 & \qquad x^2 z^2 - x^4 z & \\
 & \qquad \overline{\qquad\qquad} & \\
 & \qquad\quad x^4 z - x^3 y^2 & \\
 & \qquad\quad x^4 z - x^6 & \\
 & \qquad\quad \overline{\qquad\qquad} & \\
 & \qquad\qquad -x^3 y^2 + x^6 & \\
 & \qquad\qquad -x^3 y^2 + x^6 & \\
 & \qquad\qquad \overline{\qquad\qquad} & \\
 & \qquad\qquad\quad 0 &
\end{array}
$$

multiple of the leading monomials. Then

$$S(f,g) = \frac{\boldsymbol{x}^{\boldsymbol{\gamma}}}{LT(f)} f - \frac{\boldsymbol{x}^{\boldsymbol{\gamma}}}{LT(g)} g. \tag{6.1}$$

We note that this is indeed a polynomial. The idea behind the S-polynomial is that we multiply f and g by two minimal terms such that the leading terms of the resulting polynomials will cancel out. The following theorem explains the significance of the S-polynomial:

THEOREM 6.1. *(Buchberger Criterion) G is a Gröbner basis (of the ideal it generates) iff for every $f, g \in G$ we have $S(f,g) \xrightarrow[G]{} 0$, i.e., the remainder of division of $S(f,g)$ by G is 0.*

7. An example of Buchberger Criterion. Let us exemplify the Buchberger Criterion by performing an easy but illustrative calculation in algebraic geometry. Let V be the image of the map $k \ni t \mapsto (t^2, t^3, t^4)$. This is a parametric curve in k^3. Let $W = V(\{y^2 - x^3, z - x^2\})$. It is clear that $V \subseteq W$. However, the equality $V = W$ is not immediately obvious. It is easy to see that $V = W$ follows from the solution of the following exercise:

PROBLEM 2. *Show that $I = I(V) = \mathrm{Id}(\{y^2 - x^3, z - x^2\})$.*

The solution amounts to showing that $G = \{g_1, g_2\}$ where $g_1 = y^2 - x^3$ and $g_2 = z - x^2$, is a Gröbner basis of this ideal with variable ordering

TABLE 3
An example of the Buchberger criterion.

	$a_1 : x^2$
	$a_2 : -x^3$
$y^2 - x^3$	$-x^3z + x^2y^2$
$z - x^2$	$-x^3z + x^5$
	———————-
	$x^2y^2 - x^5$
	$x^2y^2 - x^5$
	———————-
	0

$x \prec y \prec z$ and lex ordering. This can be verified using the Buchberger criterion.

$$S(g_1, g_2) = z(y^2 - x^3) - y^2(z - x^2) = -x^3z + x^2y^2$$

The division algorithm yields the result in table 3. Thus, $S(g_1, g_2) \xrightarrow[G]{} 0$, so G is a Gröbner basis. We complete the argument by considering $f \in I(V)$. We write it as $f = a_1f_1 + a_2f_2 + r$, where $r \in I(V)$ as well. But $r = a(x) + yb(x)$ because no term of r is divisible by z or y^2. The substitution $x = t^2, y = t^3$ yields $a(t^2) + t^3b(t^2)$. Thus $a(t^2) \equiv 0$ and $b(t^2) \equiv 0$ (by comparing even and odd powers of t). Hence $a = b = 0$. Also $\sqrt{I} = I$.

8. Buchberger algorithm. In the algorithm presented in table 4 the notation

$$NormalForm(f, F)$$

is used to denote the remainder part of the output of the division algorithm of f by F. As we can see, in the course of the Buchberger algorithm we select a pair of polynomials (f, g), called the *critical pair*, from the current pool of polynomials G and form the S-polynomial of the two. Subsequently, we try to verify that the resulting S-polynomial is in the ideal spanned by G by means of the division algorithm. If the remainder of the division is non-zero then we add it to G, otherwise we either find another pair for which the remainder is non-zero or, if no such pair exists, declare that G is a Gröbner basis.

9. Quantifier elimination and geometric theorem proving. One of the traditional application domains for the methods just introduced is automatic geometric theorem proving. Let us discuss a simple problem of this sort and see how it is solved using Gröbner bases.

The following theorem is well-known in elementary geometry and it can be proved easily by traditional techniques.

TABLE 4
The Buchberger algorithm for computing Gröbner bases.

Input: $F = (f_1, f_2, \ldots, f_s) \subseteq k[\boldsymbol{x}]$
Output: a Gröbner basis $G = (g_1, g_2, \ldots, g_t)$ of $\mathrm{Id}(F)$
$G := F$
repeat
    $G' = G$
    for each pair $\{p, q\}$, $p \neq q$, **in** G **do**
        $S := NormalForm(S(p, q), G)$
        if $S \neq 0$ **then** $G' := G' \cup \{S\}$
    if $G' = G$
        return G
    else $G := G'$

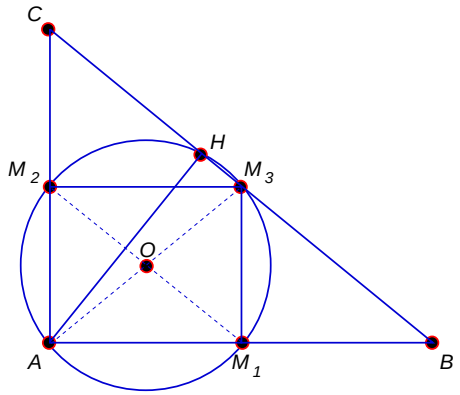


FIG. 1. *An illustration of the Apollonius Circle Theorem.*

THEOREM 9.1. *(Apollonius Circle Theorem) If ABC is a triangle, M_1, M_2, M_3 are the centers of the sides and H is the foot of the altitude drawn from A then, M_1, M_2, M_3 and H lie on one circle.*

However, we will translate this theorem into a statement about polynomials and look at the problem of finding an "automatic" proof for the resulting algebraic statement. Figure 1 serves as a visualization of our notation.

There are several ways of finding an algebraic encoding of this theorem. We have chosen one which produces relatively simple equations. The advantage of our encoding is a compact presentation of the resulting algebraic problem. The disadvantage is that we performed some algebraic preprocessing of the geometric problem, which is natural for a human being but may not be entirely obvious to implement algorithmically. Ideally the preprocessing would happen automatically when our algorithm is presented with the geometric formulation of the problem. In section 15 we will describe such a preprocessor.

Let $A = (0,0)$, $B = (u_1, 0)$ and $C = (0, u_2)$. Thus,

$$M_1 = (\frac{u_1}{2}, 0),$$
$$M_2 = (0, \frac{u_2}{2}),$$
$$M_3 = (\frac{u_1}{2}, \frac{u_2}{2})$$

are the midpoints of the sides. Let $H = (x_1, x_2)$.

We can see that $AM_1M_3M_2$ is a rectangle, so the circle containing A, M_1, M_2, M_3 is given by the equation:

$$(x_1 - u_1/4)^2 + (x_2 - u_2/4)^2 = (u_1/4)^2 + (u_2/4)^2. \tag{9.1}$$

The conditions defining H are

1. $AH \perp BC$;
2. B, C, H are collinear.

The first condition translates into the equation

$$f_1 = (x_1, x_2).(u_1, -u_2) = 0. \tag{9.2}$$

The second condition translates into vanishing of the determinant

$$f_2 = \begin{vmatrix} u_1 & 0 & 1 \\ 0 & u_2 & 1 \\ x_1 & x_2 & 1 \end{vmatrix} = 0. \tag{9.3}$$

The expanded polynomials f_1, f_2 and f are:

$$\begin{aligned} f_1 &= x_1u_1 - x_2u_2, \\ f_2 &= -x_1u_2 - u_1x_2 + u_1u_2 \\ f &= (x_1 - u_1/4)^2 + (x_2 - u_2/4)^2 - (u_1/4)^2 - (u_2/4)^2 \\ &= x_2^2 - u_2x_2/2 + x_1^2 - u_1x_1/2. \end{aligned}$$

Thus the Apollonius Circle Theorem admits the following reformulation:

$$\forall u_1 > 0, u_2 > 0 \; (f_1 = 0 \wedge f_2 = 0) \Rightarrow f = 0. \tag{9.4}$$

The following definition is helpful to relating the above statement to the Ideal Membership Problem:

DEFINITION 9.1. *A polynomial* $f \in S$ follows strictly *from* $f_1, \ldots, f_s \in k[\boldsymbol{u}, \boldsymbol{x}]$ *if* $f \in I(V(f_1, \ldots, f_s))$.

If k is algebraically closed then this is equivalent to the condition that $f \in \sqrt{\mathrm{Id}(\{f_1, f_2, \ldots, f_s\})}$. If $k = \mathbb{R}$ then we have no simple algebraic criterion of this sort. However, if f follows strictly over $\mathbb{C}$ then it follows strictly over $\mathbb{R}$. Thus the class of geometric problems which can be reduced to a problem in ideal theory are the ones for which the complexified version

holds. For the Apollonius Circle Theorem we will replace the original problem with the problem of deciding whether the following statement holds:

$$\forall u_1, u_2 \in \mathbb{C}\ (f_1 = 0 \wedge f_2 = 0) \Rightarrow f = 0. \tag{9.5}$$

Problems which rely upon the order structure of real numbers in an essential way require different methods from the ones presented in this article.

There is another complication: hardly any geometry theorem translated into an algebraic statement similar to 9.5 leads to a true algebraic statement. This is due to the existence of exceptional parameters for which the algebraic statement is false. When we formulate a geometric theorem, we add assumptions which are not reflected by equations but by *inequations*, i.e., conditions of the form $f(\boldsymbol{x}, \boldsymbol{u}) \neq 0$. For instance, in the Apollonius Circle Theorem we assume that we are dealing with a genuine triangle, i.e., that the points A, B and C are not collinear. At first, it may seem that adding such conditions is not trivial. Upon further analysis, we notice that such conditions can be moved to the *right-hand side* of the implication 9.5.

For instance, in our example f does not follow strictly from f_1 and f_2. To see this, we observe that if $(\boldsymbol{u}, \boldsymbol{x}) \in V(\{u_1, u_2\})$ then $(\boldsymbol{u}, \boldsymbol{x}) \in V(\{f_1, f_2\})$. Thus, $V(\{u_1, u_2\})$ is a subset of $V(\{f_1, f_2\})$. But this means that (x_1, x_2) are arbitrary, thus f does not have to vanish. Equivalent correct reformulations of the problem are:

$$\begin{aligned}
&\forall u_1, u_2\ \big(f_1 = 0 \wedge f_2 = 0 \wedge (u_1 \neq 0 \vee u_2 \neq 0)\big) \Rightarrow f = 0,\\
&\forall u_1, u_2\ \big(f_1 = 0 \wedge f_2 = 0 \wedge (u_1 u_2 \neq 0)\big) \Rightarrow f = 0,\\
&\forall u_1, u_2\ \big(f_1 = 0 \wedge f_2 = 0\big) \Rightarrow \big(f = 0 \vee u_1 u_2 = 0\big),\\
&\forall u_1, u_2\ \big(f_1 = 0 \wedge f_2 = 0\big) \Rightarrow u_1 u_2 f = 0.
\end{aligned}$$

We skipped the general quantifiers for $\boldsymbol{x}$. Let us conclude by giving two *quantifier-free* versions of the complex reformulation of the Apollonius Circle Theorem:

$$\begin{aligned}
&u_1 u_2 f \in I(V(\{f_1, f_2\}))\\
&u_1 u_2 f \in \sqrt{\mathrm{Id}(\{f_1, f_2\})} \quad (k \text{ algebraically closed}).
\end{aligned}$$

10. Testing radical ideal membership. A *saturation ideal* of an ideal I in another ideal J is defined as follows:

$$I : J^\infty = \{f \in k[\boldsymbol{x}] :\ \exists g \in J\ \exists n \geq 0\ g^n f \in I\}. \tag{10.1}$$

This ideal is fundamental in many calculations. This is due to the fact that over algebraically closed fields the condition that $V(I) \subseteq \bigcup_{k=1}^{t} V(J_k)$ is

equivalent to the statement that the iterated saturation ideal (or *polysaturated ideal*) $I : J_1^\infty : J_2^\infty : \ldots : J_t^\infty$ is trivial, i.e., it contains 1. The operator ":" groups from left to right.

A Gröbner basis of the saturation ideal can be computed using any method for calculating Gröbner bases, based on a number of observations. Let $I = \mathrm{Id}(\{f_1, f_2, \ldots, f_s\})$ and $G = \mathrm{Id}(\{g_1, g_2, \ldots, g_r\})$. Then the algorithm for finding a Gröbner basis of $I : J^\infty$ is as follows:

1. We form the set
$$F' = F \cup \{1 - t_1 g_1 - \ldots - t_r g_r\}$$
where $t_1, t_2, \ldots, t_r$ are new variables.
2. We compute a Gröbner basis H' of $\mathrm{Id}(F')$ with respect to a monomial order in which all monomials containing t's precede all monomials that do not depend on t's. Such orders are called *elimination orders*.
3. The subset $H \subseteq H'$ of those polynomials which do not depend on t's forms a Gröbner basis of $I : J^\infty$.

Thus we have the following criterion: $f \in \sqrt{I}$ iff $1 \in I : f^\infty$. The notation $I : f^\infty$ is an abbreviation for $I : \mathrm{Id}(\{f\})^\infty$.

11. A general scheme for proving geometric theorems. Geometric theorems which can be proved by algebraic methods reduce to the statements of the following form:

$$\begin{aligned}\forall \boldsymbol{x} \in \mathbb{C}^n\ \forall \boldsymbol{u} \in \mathbb{C}^m\, (f_1(\boldsymbol{x},\boldsymbol{u}) = f_2(\boldsymbol{x},\boldsymbol{u}) = \ldots = f_s(\boldsymbol{x},\boldsymbol{u}) = 0) \Rightarrow \\ \left(g_1^{(1)}(\boldsymbol{x},\boldsymbol{u}) = g_2^{(1)}(\boldsymbol{x},\boldsymbol{u}) = \ldots = g_{t_1}^{(1)}(\boldsymbol{x},\boldsymbol{u}) = 0\right) \vee \\ \left(g_1^{(2)}(\boldsymbol{x},\boldsymbol{u}) = g_2^{(2)}(\boldsymbol{x},\boldsymbol{u}) = \ldots = g_{t_2}^{(2)}(\boldsymbol{x},\boldsymbol{u}) = 0\right) \vee \\ \ldots \vee \\ \left(g_1^{(l)}(\boldsymbol{x},\boldsymbol{u}) = g_2^{(l)}(\boldsymbol{x},\boldsymbol{u}) = \ldots = g_{t_l}^{(l)}(\boldsymbol{x},\boldsymbol{u}) = 0\right).\end{aligned}$$

We form the ideals in $k[\boldsymbol{x},\boldsymbol{u}]$:

$$I = \{f_1, f_2, \ldots, f_s\} \tag{11.1}$$

$$J_k = \mathrm{Id}(\{g_1^{(k)}, g_2^{(k)}, \ldots, g_{t_k}^{(k)}\}) \quad \text{for } k = 1, 2, \ldots, l. \tag{11.2}$$

The geometric problem reduces to verifying whether 1 is in the iterated saturation ideal $I : J_1^\infty : J_2^\infty : \ldots : J_l^\infty$.

12. A crash introduction to Common Lisp. We developed a software package which performs calculations involving Gröbner bases of ideals and their saturations. This package is called *CGBLisp* and is implemented in Common Lisp, a modern version of a language as old as FORTRAN but entirely different in spirit. The language is known for its ease of implementing symbolic manipulations systems. Recently the standarization

project for Common Lisp has been completed which is important for its future and guarantees stability. Many symbolic computation systems are implemented in Common Lisp, for example, MACSYMA and its public version called MAXIMA. Even modern systems like *Mathematica* bear strong resemblance to Lisp in their approach to symbolic computing.

Common Lisp is not commonly taught by computer science departments except for a short introduction, and it is not well-known to the mathematical community. Therefore, we will give a minimal introduction to the language which will make it possible to understand the syntax of the examples that follow.

First of all, Common Lisp uses prefix notation. A mathematician denotes a function call by $f(x_1, x_2, \ldots, x_n)$. Lisp denotes the same call by

```
(f x1 x2 ... xn).
```

This syntax is prevalent throughout the Lisp language. Every expression written as `(f x1 x2 ... xn)` is called a *form*. When started, Lisp reads forms from the standard input and *evaluates* them, and then it prints the result. This process is called the `read-eval-print` loop. Many forms are function calls and they are evaluated with the obvious semantics. However, function calls cannot be used to implement control flow. The problem is with the fact that arguments to functions are evaluated *before* the functions themselves. In addition, every argument is evaluated. In contrast, the form `(if (zerop x) 0 (/ 5 x))` which implements the mathematical function

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \frac{5}{x} & \text{otherwise} \end{cases} \tag{12.1}$$

would perform division by zero regardless of the consequences. This is the reason why some forms are *special forms*, which means that they evaluate their arguments according to special rules. Lisp has only a few special forms (approximately seven, depending on the implementation). In addition, a user can define *macros* which resemble built-in special forms. In this way, the syntax of Lisp can be easily extended. However, the bulk of work is performed by recursive function calls.

The prototypical special form which does not evaluate its arguments is called `quote`. For example, the expression `(quote x)` returns the symbol `x` and will not try to evaluate x, i.e., find the value of `x` somewhere in Lisp's internal tables and print this value. This special form also can be entered using the *quote* syntax: `'x`. Any input expression `'form` is immediately translated into `(quote form)` when the Lisp interpreter first sees it.

Assignment in Lisp can be accomplished in several ways. Of course, all of them amount to building an apropriate form and evaluating it. The following forms result in associating the value 1 with the symbol `x`:

1. `(set 'x 1);`
2. `(setq x 1);`
3. `(setf x 1).`

Here `set` is a function, `setq` is a special form and `setf` is a macro. The semantics are the same. The reader may ask: why not quote 1? This is because 1 is an atom, i.e., a form which evaluates to itself. Numbers and strings (denoted `"abc..."`) and several other data types are atoms.

The name "Lisp" stands for *List Processing*. A list is denoted by `(a b c ... )` where `a, b, c, ...` could be any objects. One may ask: what is the difference between lists and forms? A form is a list which is suitable as input to the Lisp interpreter, i.e., a list whose first element is either a function, a special form or a macro. Only then does Lisp know what to do with the form.

The user can define functions of his own by evaluating a `defun` form. For instance, `(defun f (x) (if (zerop x) 0 (/ 5 x)))` defines a function. After this is done, entering the form `(f 0)` results in 0 being printed, `(f 3)` results in `5/3` being printed, etc. (Note: Common Lisp has practically infinite precision integers and rational numbers.)

A line beginning with a ";" is a comment in Lisp. `T` stands for "true" and `NIL` for "false". In particular, `T` cannot be normally used as a variable name. Both `T` and `NIL` are atoms.

13. About the syntax of *CGBLisp*. Packages like MACSYMA hide the syntax of Lisp under a layer of Pascal-like syntax. This is accomplished using Lisp's capability of extending its own syntax. *CGBLisp* does not use this approach. The syntax is that of Lisp. However, typing in expressions in Lisp syntax is a bit awkward. For instance, $x^3 + 2xy$ is translated into `'(+ (expt x 3) (* 2 x y))`. We chose to have explicit functions which will translate a polynomial or a list of polynomials to the internal notation. For instance, the following command translates the above polynomial to internal representation used by *CGBLisp*:

```
USER(7): (string-read-poly "x^3+2*x*y" '(x y))
Args:X^3 + 2 * X * Y
(((3 0) . 1) ((1 1) . 2))
```

Polynomial expressions are first translated to infix notation `'(+ (expt x 3) (* 2 x y))` and then to *distributed representation*. In this representation, a polynomial is represented as a list of pairs corresponding to the terms of the polynomial. The first element of each pair is a list of powers of each variable. The second element of the pair is the corresponding coefficient. We note that the functions `string-read-poly` received two arguments. The first argument is the string containing the polynomial and the second argument, `'(x y)`, is a quoted list of variables. The default order of the resulting polynomial is lexicographic. Another example is

```
USER(14): (string-read-poly "x^2*y+x*y^6" "[x,y]" :order #'grevlex>)
Args:X * Y^6 + X^2 * Y
```

```
(((1 6) . 1) ((2 1) . 1))
```

In this example, we read the polynomial $x^2y + xy^6$ using a list of variables, represented as a string `"[x,y]"`, using **grevlex** as the monomial order. In the above example, we use a *keyword parameter* to a Lisp function. Keyword parameters are passed by preceding them with a *keyword*, in this case, `:order`. Keywords are symbols starting with ":". The argument `#'grevlex>` is the actual function object which determines the order of two monomials. This is the function associated with the symbol `grevlex>` made by evaluating the form `(defun grevlex> ... )` somewhere in the source code of *CGBLisp*.

14. Calculations using *CGBLisp*. The actual calculations which perform an automatic proof of the Apollonius Circle Theorem (see section 9) are performed by a simple Common Lisp program:

```
;; An automatic proof of the Apollonius Circle Theorem
;; Encoding by hand
(setf vars '(x1 x2))
(setf params '(u1 u2))
(setf allvars (append vars params))
(setf hypotheses "[x1*u1-x2*u2, -x1*u2-u1*x2+u1*u2]")
(setf conclusions  "[2*x2^2-u2*x2+2*x1^2-u1*x1,u1,u2]")
;; Saturation test
(string-ideal-polysaturation-1 hypotheses conclusions allvars)
```

The output of this program is presented below. The output was produced by invoking the Lisp interpreter from the operating system shell and by loading a file included in the distribution of *CGBLisp* in the course of an interactive session:

```
CGB-LISP(74): (load "../examples/apollonius0")
; Loading ../examples/apollonius0.lisp
Args1:[ X1 * U1 - X2 * U2,  - X1 * U2 - X2 * U1 + U1 * U2 ]
Args2:[ 2 * X1^2 - X1 * U1 + 2 * X2^2 - X2 * U2, U1, U2 ]
[ 1 ]
T
```

The output produced by the function

`string-ideal-polysaturation-1`

is on the line next to the last one: `[ 1 ]`. It is a *reduced* Gröbner basis of the saturation ideal $\mathrm{Id}(H) : Id(C)^{\infty}$, where $H = hypotheses = \{f_1, f_2\}$ and $C = conclusions = \{f\}$ in the notation of section 9. The last line is produced by `load`. A Gröbner basis is reduced if no element contains a monomial divisible by the leading monomial of another element. A reduced Gröbner basis is constructed by repeatedly dividing each element

by the remaining elements. In general, `string-ideal-polysaturation-1` produces a reduced Gröbner basis of the iterated saturated ideal $\mathrm{Id}(F) : g_1^\infty : g_2^\infty : \ldots : g_r^\infty$ where F is the first argument (`hypotheses` in our example) and $G = \{g_1, g_2, \ldots, g_r\}$ is the second argument. The abbreviation $I : f^\infty$ stands for $I : \mathrm{Id}(\{f\})$. There is also a function

`string-ideal-polysaturation`

which takes a *list of lists* of polynomials as the second argument, and it computes $I : J_1^\infty : J_2^\infty : \ldots : J_r^\infty$, where J_k is generated by the k-th sublist extracted from the second argument.

15. Automation of geometric theorem proving. One can go a step further in automatic geometric theorem proving by defining an interface which allows one to enter theorems in a way which corresponds very closely to a formulation found in a geometry textbook. *CGBLisp* has such an interface. The idea is to define a number of standard relations which are used to formulate geometric theorems. For instance, the Apollonius Circle Theorem (see section 9) can be formulated and proven by presenting the input in table 5 to *CGBLisp*. The structure of the input can be easily described. The arguments of the macro `prove-theorem` are two lists. The first list `((perpendicular A B A C) ... (perpendicular A H B C))` represents the assumptions of the Apollonius Circle Theorem. The second list `((equidistant M O H O) (identical-points B C))` represents the conclusions. The assumption `(perpendicular A B A C)` is stated using the relation `perpendicular` which represents the mathematical statement $AB \perp AC$, i.e., that the line passing through the points A and B is perpendicular to the line passing through the points A and C. The statement `(midpoint B C M)` means that M is the midpoint of the segment BC, etc. We note that the conclusion list states that either the distance between M and O is equal to the distance between H and O or the points B and C are identical. The conclusion that $B = C$ is naturally formulated as an inequation $B \neq C$ and placed in the assumption list. The condition $B \neq C$ is necessary, as we have noticed, because otherwise the triangle ABC degenerates to a point and the Apollonius Circle Theorem is false. We have already discovered that inequations can be avoided by negating them and placing them in the conclusion list.

The output produced by the input in table 5 is `[ 1 ]`, which is the Gröbner basis of the corresponding saturation ideal, and is described in the previous section.

How does the automatic proof work internally? First, the geometric relations are translated into equations. The macro `translate-theorem` used in place of `prove-theorem` produces the ideals I, J_1, ..., J_l given by formulas 11.1 and 11.2; more precisely, the output is the following nested list:

$$\left((f_j)_{j=1}^{s} , \left(\left(g_r^{(k)} \right)_{r=1}^{t_r} \right)_{k=1}^{l} \right). \tag{15.1}$$

TABLE 5
An automated proof of Apollonius Circle Theorem.

```
;;
;; Prove Apollonius circle theorem
;;
(prove-theorem
 ((perpendicular A B A C)
  (midpoint B C M)
  (midpoint A M O)
  (collinear B H C)
  (perpendicular A H B C))
 ((equidistant M O H O)
  (identical-points B C)
  ))
```

Of course, the resulting expressions are given in infix notation, used by Lisp. Each of the points A, B, ... enters these expressions as a pair of coordinates, for instance, $A = (A1, A2)$. The list `(M1 M2 ... C2)` which strictly speaking does not appear in formula 15.1, is simply a list of all variables which appear in the assumptions list, and it amends the assumptions list. Similarly, the variable list `(M1 M2 H1 ... C2)` amends the conclusion list. Both are needed for bookkeeping purposes.

The structure of the equations produced by the macro `translate-theorem` can be easily understood. For instance, the condition `(perpendicular A B A C)` translates into the equation `(+ (* (- A1 B1) (- A1 C1)) (* (- A2 B2) (- A2 C2)))` which written in infix notation is $(A1 - B1)(A1 - C1) + (A2 - B2)(A2 - C2)$, and is simply the dot product $(A - B) \cdot (A - C)$.

We note that the automatically translated equations have more variables and are generally more complicated than the equations produced by hand in section 9. The running time of the Gröbner basis calculation is not affected in our example. However, more complicated theorems may result in substantially more complicated equations, and the running time may be vastly increased as compared to a clever by-hand encoding of the problem.

16. The Comprehensive Gröbner Basis algorithm. The Comprehensive Gröbner Basis Algorithm (for brevity, we will refer to it as *CGB algorithm* as well) is a version of the ordinary Buchberger algorithm for calculating Gröbner bases, adapted to parametric problems. Although we have seen that parametric problems can be handled using the usual Gröbner basis algorithm, sometimes it is beneficial to separate the computations which take place on a subset of variables designated as parameters from other calculations. A simple example is quite good at explaining the

TABLE 6
Automatic theorem translation.

```
CGB-LISP(13): (translate-theorem
 ((perpendicular A B A C)
  (midpoint B C M)
  (midpoint A M O)
  (collinear B H C)
  (perpendicular A H B C))
 ((equidistant M O H O)
  (identical-points B C)
  ))
(((+ (* (- A1 B1) (- A1 C1)) (* (- A2 B2) (- A2 C2)))
  (- (* 2 M1) B1 C1) (- (* 2 M2) B2 C2)
  (- (* 2 O1) A1 M1) (- (* 2 O2) A2 M2)
  (+ (- (* H1 C2) (* H2 C1)) (- (* B2 C1) (* B1 C2))
     (- (* B1 H2) (* B2 H1)))
  (+ (* (- A1 H1) (- B1 C1)) (* (- A2 H2) (- B2 C2))))
 (M1 M2 O1 O2 A1 A2 H1 H2 B1 B2 C1 C2))
((((- (+ (EXPT (- M1 O1) 2) (EXPT (- M2 O2) 2))
      (+ (EXPT (- H1 O1) 2) (EXPT (- H2 O2) 2))))
  ((- B1 C1) (- B2 C2)))
 (M1 M2 H1 H2 O1 O2 B1 B2 C1 C2))
```

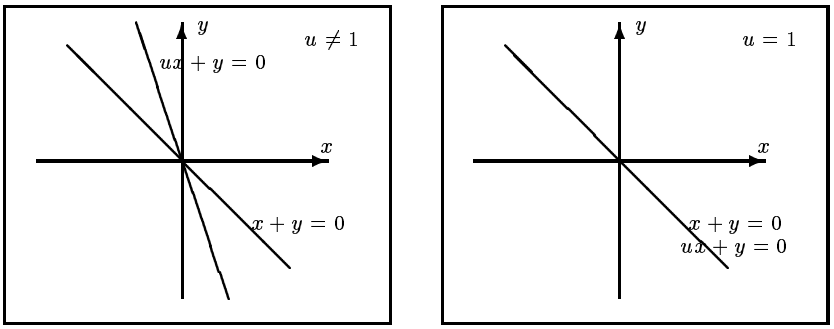


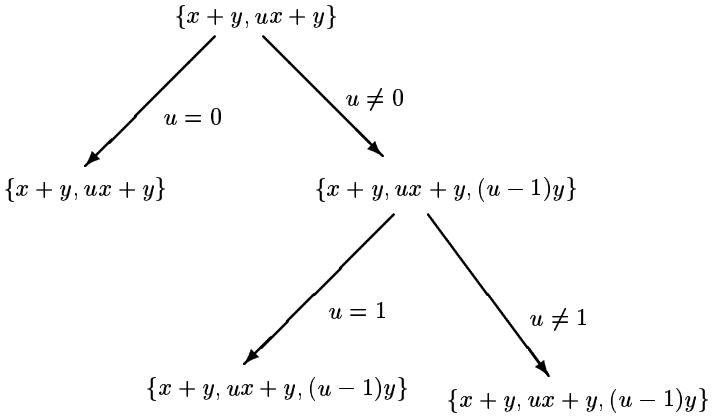
FIG. 2. *A simple parametric problem.*

complications when dealing with parameters:

PROBLEM 3. *What is the dimension of the intersection of two lines* $x+y=0$ *and* $ux+y=0$ *as a function of the parameter* u*? (see figure 2.)*

We will solve this problem by defining the ideal $F = \{x+y, ux+y\} \subset k[u,x,y]$. Let us first find the Gröbner basis as a function of the parameter. We run into a problem: we are not able to determine $LM(ux+y)$. The idea is to branch the computation into two cases: $u=0$ and $u \neq 0$. In fact, this is the main idea behind the Comprehensive Gröbner Basis Algorithm.

Similar branching will be performed at various stages of the computation, so that the leading monomial of the polynomials in question is determined. The result of performing the CGB algorithm is the following tree:



A *Gröbner system* is the set of leaves of the above tree. More precisely, a *Gröbner system* consists of pairs. The second element of each pair is a Gröbner basis, which is a leaf of the tree. The first element is a *condition*, i.e., a set of equations, also called the *green list* and inequations called the *red list*. Both green and red lists are obtained by walking down from the root of the tree to the leaf and adjoining all partial conditions which are labels of the edges of the above tree. The Gröbner system associated with the above tree is:

$$\begin{aligned}&\{(\{u=0\}, \{x+y, ux+y\}),\\ &\ (\{u\neq 0, u=1\}, \{x+y, ux+y, (u-1)y\}),\\ &\ (\{u\neq 0, u\neq 1\}, \{x+y, ux+y, (u-1)y\})\ \}.\end{aligned}$$

This Gröbner system can be *reduced* by continuing down until a reduced Gröbner basis is obtained.

The set of leaves of the above tree (without the conditions) is called a *Comprehensive Gröbner Basis*. The original focus of *CGBLisp* was an implementation of the Comprehensive Gröbner Basis algorithm. Thus the *CGB* part of the name of the package refers to this algorithm. A Comprehensive Gröbner Basis has the property that its specialization, i.e., substitution of a particular value of $\boldsymbol{u}$, is a Gröbner basis for the specialized ideal of the original ideal. For instance, for the above example, the Comprehensive Gröbner Basis is:

$$\{x+y, ux+y, (u-1)y\}.$$

For any substitution of the parameters, this set is a Gröbner basis.

The conditions provide valuable information about the problem in question. For our example, they detect that for $u=1$ the two lines coincide. The corresponding specialized ideal is $\mathrm{Id}(\{x+y\})$. For other u the

TABLE 7
A simple CGB computation.

```
CGB-LISP(89): (string-grobner-system "[x+y,u*x+y]" '(x y) '(u))
------------------- CASE 1 -------------------
Condition:
  Green list: [ U ]
  Red list: [  ]
 Basis: [ (1) * X, (1) * Y ]
------------------- CASE 2 -------------------
Condition:
  Green list: [  ]
  Red list: [ U, U - 1 ]
 Basis: [ (U^2 - U) * X, (U - 1) * Y ]
------------------- CASE 3 -------------------
Condition:
  Green list: [ U - 1 ]
  Red list: [ U ]
 Basis: [ (1) * X + (1) * Y ]
```

specialized Comprehensive Gröbner Basis can be reduced to $\{x, y\}$ which means that $(0, 0)$ is the only intersection point of the two lines.

Finally, let us run *CGBLisp* on the above example with the following input:

```
(string-grobner-system "[x+y,u*x+y]" '(x y) '(u))
```

The results are presented in table 7.

As we can see, there is some difference from our computation by hand: the implementation of the algorithm carries out the reduction. Thus, should we substitute any value of the parameter, there will be exactly one condition which is satisfied by that value (i.e., all green polynomials will vanish and all red polynomials will not vanish) and the corresponding set of polynomials will become a reduced Gröbner basis for the resulting ideal. The reduction step can be omitted by using a keyword parameter `:reduce` set to `nil` in `string-grobner-system`. The corresponding input and output are in table 8.

17. The CGB algorithm applied to the Apollonius Circle Theorem. It is interesting to see this algorithm in action to automatically prove the Apollonius Circle Theorem when applied to the polynomials introduced in section 9. This time, we present the following input to *CGB-Lisp*:

```
(string-grobner-system
```

TABLE 8
A simple CGB computation with the reduction step suppressed.

```
CGB-LISP(5): (string-grobner-system "[x+y,u*x+y]" '(x y) '(u)
          :reduce nil)
------------------- CASE 1 -------------------
Condition:
  Green list: [ U ]
  Red list: [  ]
 Basis: [ (1) * X + (1) * Y, (1) * Y ]
------------------- CASE 2 -------------------
Condition:
  Green list: [  ]
  Red list: [ U, U - 1 ]
 Basis: [ (1) * X + (1) * Y, (U) * X + (1) * Y, (U - 1) * Y ]
------------------- CASE 3 -------------------
Condition:
  Green list: [ U - 1 ]
  Red list: [ U ]
 Basis: [ (1) * X + (1) * Y, (1) * X + (1) * Y ]
```

```
      "[x1*u1-x2*u2, -x1*u2-u1*x2+u1*u2,
      1-s*(2*x2^2-u2*x2+2*x1^2-u1*x1)]"
      '(s x1 x2) '(u1 u2))
```

The output of this example is somewhat longer than expected and is presented in table 9. This is a direct application of the method for calculating the Gröbner basis of a saturated ideal. We compute the Gröbner basis of $\{f_1, f_2, 1 - sf\}$ with variables x_1, x_2 and s and parameters u_1 and u_2. The theorem is true for only these conditions for which a non-zero constant polynomial is in the Gröbner basis. By inspection we identify these cases as CASES 2–5. CASE 1 has $\{u_1, u_2\}$ as its green list. In this case the algebraic version of the Apollonius Circle Theorem fails. Thus, the Comprehensive Gröbner Basis algorithm detected the missing assumption! Finally, we can add the assumption $u_1 \neq 0 \vee u_2 \neq 0$ to the specification of the algorithm as the initial set of conditions (called *cover*):

```
(string-grobner-system
      "[x1*u1-x2*u2, -x1*u2-u1*x2+u1*u2,
      1-s*(2*x2^2-u2*x2+2*x1^2-u1*x1)]"
      '(s x1 x2) '(u1 u2) :cover '(("[]" "[u1]") ("[]" "[u2]")))
```

The resulting output is in table 10. We see that now every case produces a non-zero constant in the Gröbner basis. Hence, the automatic proof is complete.

TABLE 9
The output of the CGB algorithm on the Apollonius Circle Theorem problem.

```
CGB-LISP(88): (string-grobner-system
      "[x1*u1-x2*u2, -x1*u2-u1*x2+u1*u2,
      1-s*(2*x2^2-u2*x2+2*x1^2-u1*x1)]"
      '(s x1 x2) '(u1 u2))
------------------- CASE 1 -------------------
Condition:
  Green list: [ U2, U1 ]
  Red list: [  ]
 Basis: [ ( - 2) * S * X1^2 + (0) * S * X1 +
      ( - 2) * S * X2^2 + (0) * S * X2 + (1) ]
------------------- CASE 2 -------------------
Condition:
  Green list: [ U2 ]
  Red list: [ U1 ]
 Basis: [ ( - U1^3) ]
------------------- CASE 3 -------------------
Condition:
  Green list: [ U1 ]
  Red list: [ U2 ]
 Basis: [ (U2^2) ]
------------------- CASE 4 -------------------
Condition:
  Green list: [  ]
  Red list: [ U1, U2, U1^2 + U2^2 ]
 Basis: [ ( - U1^2) ]
------------------- CASE 5 -------------------
Condition:
  Green list: [ U1^2 + U2^2 ]
  Red list: [ U1, U2 ]
 Basis: [ ( - U2^3) ]
```

18. Mathematical robotics. Mathematical models of robots are a rich source of examples of various levels of complexity. Figure 3 depicts a typical model of a robot. We will only consider robots whose joints are contained in a single plane. This particular robot has three *joints* and three *argms*, whose positions are described by three angles θ_j, $j = 1, 2, 3$ and is endowed with a *grasper* at the end of its arm. The lengths of the arms are ℓ_2, ℓ_3 and ℓ_4. (The reason for starting from ℓ_2 is somewhat obscure and there is no significance to this fact.) The internal configuration of the robot is described by the *joint space*: $\mathcal{J} = \{(\theta_1, \theta_2, \ldots, \theta_m)\}$. The *configuration space* describes the position of the grasper: $\mathcal{C} = \{(a, b)\}$. The *joint map* $f : \mathcal{J} \to \mathcal{C}$ assigns the position of the grasper to any internal state of the robot.

TABLE 10
CGB algorithm using an initial cover.

```
CGB-LISP(87): (string-grobner-system
      "[x1*u1-x2*u2, -x1*u2-u1*x2+u1*u2,
      1-s*(2*x2^2-u2*x2+2*x1^2-u1*x1)]"
      '(s x1 x2) '(u1 u2) :cover '(("[]" "[u1]") ("[]" "[u2]")))
------------------- CASE 1 -------------------
Condition:
  Green list: [ U1 ]
  Red list: [ U2 ]
 Basis: [ (U2^2) ]
------------------- CASE 2 -------------------
Condition:
  Green list: [  ]
  Red list: [ U1, U2, U1^2 + U2^2 ]
 Basis: [ ( - U1^2) ]
------------------- CASE 3 -------------------
Condition:
  Green list: [ U1^2 + U2^2 ]
  Red list: [ U1, U2 ]
 Basis: [ ( - U2^3) ]
------------------- CASE 4 -------------------
Condition:
  Green list: [ U2 ]
  Red list: [ U1 ]
 Basis: [ ( - U1^3) ]
------------------- CASE 5 -------------------
Condition:
  Green list: [  ]
  Red list: [ U1, U2, U1^2 + U2^2 ]
 Basis: [ ( - U1^2) ]
------------------- CASE 6 -------------------
Condition:
  Green list: [ U1^2 + U2^2 ]
  Red list: [ U2, U1 ]
 Basis: [ ( - U2^3) ]
```

A *kinematic singularity* is a situation when a joint is moving with infinite velocity while the grasper is moving with finite velocity. These are simply the critical points of the joint map. Determination of kinematic singularities is one of the standard questions of mathematical robotics. We will focus our attention on the following:

PROBLEM 4. *Determine the dimensions of the varieties* $f^{-1}(c)$ *for all points* c *for the two-arm robot with* $l_2 = l_3 = 1$.

The coordinates in the configuration space will be (c_j, s_j) where $c_j =$

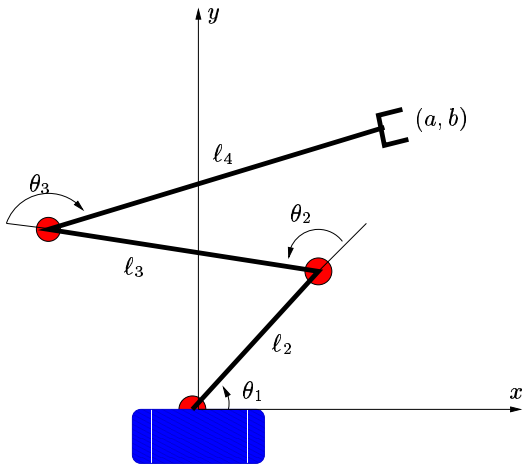


FIG. 3. *A three-arm robot.*

TABLE 11
CGBLisp *input for the robot example.*

```
(string-grobner-system
"[a-l3*c1*c2+l3*s1*s2-l2*c1, b-l3*c1*s2-l3*c2*s1-l2*s1,
        c1^2+s1^2-1, c2^2+s2^2-1]"
'(c2 s2 c1 s1)
'(a b l2 l3)
:cover '(("[l2-1,l3-1]" "[]"))
:main-order #'grevlex>
:parameter-order #'lex>
)
}}
```

$\cos\theta_j$ and $s_j = \sin\theta_j$. In these coordinates the joint map is a polynomial map which makes it possible to answer our question using Gröbner bases.

The *CGBlisp* input which solves this problem is presented in table 11. The settings $l_2 = l_3 = 1$ are included as part of the green list instead of modifying the equations. The reason for setting the lengths of both arms to 1 is that for this choice there is a kinematic singularity and yet the entire output of the CGB algorithm is still relatively compact. In this example, the main variable order is set to **grevlex**.

The dimension of an algebraic variety V over an algebraically closed field can be found using Gröbner bases. The idea is to take the monomial ideal $LM(I)$ where $V = V(I)$. It can be shown that the dimension of the variety V is the same as the dimension of the variety $V(LM(I))$:

TABLE 12
A CGB analysis of a two-arm robot.

```
CGB-LISP(90): (time
(string-grobner-system
"[a-l3*c1*c2+l3*s1*s2-l2*c1, b-l3*c1*s2-l3*c2*s1-l2*s1,
c1^2+s1^2-1, c2^2+s2^2-1]"
'(c2 s2 c1 s1)
'(a b l2 l3)
:cover '(("[l2-1,l3-1]" "[]"))
:main-order #'grevlex>
:parameter-order #'lex>
))
------------------- CASE 1 -------------------
Condition:
  Green list: [ L2 - 1, L3 - 1 ]
  Red list: [ A^2 + B^2, L2, A, L3 ]
 Basis: [ ( - 2 * A) * S2 + ( - 2 * A^2 - 2 * B^2) * S1 + (A^2 * B + B^3),
   ( - 2) * C2 + (A^2 + B^2 - 2), (2 * A) * C1 + (2 * B) * S1 + ( - A^2 - B^2),
   (4 * A^2 + 4 * B^2) * S1^2 + ( - 4 * A^2 * B - 4 * B^3) * S1
   + (A^4 + 2 * A^2 * B^2 - 4 * A^2 + B^4) ]
------------------- CASE 2 -------------------
Condition:
  Green list: [ L2 - 1, L3 - 1, A^2 + B^2 ]
  Red list: [ A^4 + 2 * A^2 * B^2 - 2 * A^2 * L2^2 - 2 * A^2 * L3^2 + B^4
    + 2 * B^2 * L2^2 - 2 * B^2 * L3^2 + L2^4 - 2 * L2^2 * L3^2 + L3^4, L2, A, L3 ]
 Basis: [ (32 * B^4) ]
------------------- CASE 3 -------------------
Condition:
  Green list: [ L2 - 1, L3 - 1, A ]
  Red list: [ L3, B * L2 ]
 Basis: [ (4 * B^2) * C1^2 + (B^4 - 4 * B^2),
   (2 * B) * S2 + ( - 2 * B^2) * C1 + (0), ( - 2) * C2 + (0) * C1 + (B^2 - 2),
   (2 * B) * S1 + ( - B^2) ]
------------------- CASE 4 -------------------
Condition:
  Green list: [ L2 - 1, L3 - 1, A, B ]
  Red list: [ L3 ]
 Basis: [ (1) * C1^2 + (1) * S1^2 + ( - 1), (1) * S2 + (0) * C1 + (0) * S1,
   (1) * C2 + (0) * C1 + (0) * S1 + (1) ]
; cpu time (non-gc) 920 msec user, 10 msec system
; cpu time (gc)     70 msec user, 0 msec system
; cpu time (total)  990 msec user, 10 msec system
; real time  1,434 msec
; space allocation:
;  689,190 cons cells, 0 symbols, 276,848 other bytes
```

$\dim(V(I)) = \dim(V(LM(I)))$. In view of the fact that $LM(I)$ is generated by monomials, the variety $V(LM(I))$ is easy to compute and it is a union of coordinate subspaces. It can also be shown that $LM(I) = LM(G)$ where G is a Gröbner basis, i.e., $LM(I)$ is generated by the leading monomials of the elements of G.

One condition which must be satisfied for the dimension calculation to work is that the monomial order be a *graded order*, i.e., a monomial of a higher total degree preceeds one with a lower total degree.

The *CGBlisp* output with CPU times performed on a 266 MHz Pentium II processor using Franz Allegro Common Lisp (version 4.3 for the Linux operating system) are included in table 12.

The dimension could be calculated automatically, but we can do it by hand easily as well. The following table does the job:

Case #	$LM(I)$	dimension
1	$\mathrm{Id}(\{s_2, c_2, c_1, s_1^2\})$	0
2	$\mathrm{Id}(\{1\})$	-1
3	$\mathrm{Id}(\{c_1^2, s_2, c_2, s_1\})$	0
4	$\mathrm{Id}(\{c_1^2, s_2, c_2\})$	1

Based on this table, we come to the following conclusions:

CASE 1 In this case $a, b \neq 0$ and $a^2 + b^2 \neq 0$ (non-geometric condition). This is the *generic condition* distinguished by the fact that the CGB algorithm made all polynomials it encountered in the tree non-zero. There exist only a finite number of points in $f^{-1}(a, b)$.

CASE 2 In this case $a^2 + b^2 = 0$. This conditions is non-geometric and it holds along the pair of straight lines in the complex space $b = \pm ai$. By plugging in $l_2 = l_3 = 1$ and $b = \pm ai$ into the red list, we can see that the red list is satisfied iff $a \neq 0$. In this case there are no points in $f^{-1}(a, b)$. These points cannot be reached by the robot. None of the points are real.

CASE 3 This case holds when $a = 0$ and $b \neq 0$. In this case $f^{-1}(0, b)$ is finite.

CASE 4 When $a, b = 0$ we have a kinematic singularity. The variety $f^{-1}(0, 0)$ is 1-dimensional. The joint variety $\mathcal{J}$ is 2-dimensional and the configuration variety is 2-dimensional as well. Thus, for non-singular values (a, b) the dimension of $f^{-1}(a, b)$ will be

$$dim(\mathcal{C}) - dim(\mathcal{J}) = 0.$$

The variety $f^{-1}(0, 0)$ in CASE 4 can be determined easily:

$$c_2 = -1,\ s_2 = 0,\ s_1 \text{ is arbitrary.}$$

19. Flip bifurcation in the quadratic family. The following example comes from dynamical systems. *CGBLisp* has a number of support functions for bifurcation calculations in dynamical systems.

The *quadratic family* is the family of mappings $f_c : \mathbb{C} \to \mathbb{C}$ given by the formula

$$f_c(z) = z^2 + c. \tag{19.1}$$

A *flip bifurcation* occurs for a given periodic point z and parameter value c if $f_c'(z) = -1$. Our goal is to write down equations for c for which there exists a periodic point of a prescribed period n, for which a flip bifurcation occurs. Thus, we need to "solve" the following system of equations:

$$\begin{aligned} f_c^n(z) &= z, \\ f_c'(z) &= -1. \end{aligned}$$

The *CGBLisp* input for this example is presented in table 13. The output is presented in table 14. The input for this example is most conveniently put in a file. The output is produced by first loading the file and then calling `print-bifurcation` with an integer argument $n = 1, 2, 3, 4$. For this example we used an efficient version of the Buchberger algorithm with so-called "sugar strategy". A plain Buchberger algorithm would not complete the calculation for $n = 4$. The form `select-grobner-algorithm` is used to choose one of several algorithms. The comments in the code provide some information about what particular lines of code accomplish. However, to fully understand this example one needs to refer to both the Common Lisp documentation as well as *CGBLisp* documentation. The following brief description should be nearly sufficient:

`string-read-poly` Read a polynomial entered in infix notation as a string. The second argument is a list of variables.

`cdr` Also called `rest`; returns a list with its first element dropped. (Common Lisp)

`poly-dynamic-power` Calculates the composition $f \circ f \circ \cdots \circ f$ (n-times), where $f : k^r \to k^r$ is a polynomial map, given as a list of polynomials (the coordinates of f). The list of polynomials is passed as the first argument and n as the second argument.

`car` Also called `first`; returns the first element of a list. (Common Lisp)

`partial` Returns the partial derivative of a polynomial map, given as a list of polynomials; `(partial f 0)` returns the partial over the first (i.e., 0-th) variable.

`mapcar` The result of evaluating `(mapcar f l1 l2 ... ln)`, where `f` is a function and lj are lists, is a list obtained by applying f to the first, second, etc., elements of the lists lj; for instance `(f '(a b) '(c d))` returns the same result as `(list (f a c) (f b d))`. (Common Lisp)

`list` Simply constructs a list from its arguments; `(list 1 2)` results in `(1 2)`; `list` is a function, i.e., its arguments are evaluated before `list` is applied. (Common Lisp)

`poly+` Adds two polynomials given in internal form.

`cons` When applied to an item and a list, it prepends the item to the list and it returns the new list. (Common Lisp)

`poly-print` Prints a polynomial given as first argument in internal form. Variables are given as the second argument. They are needed because the internal form represents monomials as multi-indices, and thus it knows of no variable names. A list of polynomials is printed by prepending the symbol `'[` to the list.

`terpri` Prints a new line character. (Common Lisp)

`poly-contract` It drops the first variable from a polynomial which explicitly does not depend on it.

`elimination-ideal` The form `(elimination-ideal f j)` evaluates to a Gröbner basis of the j-th *elimination ideal* of the ideal generated

TABLE 13
The flip bifurcation example.

```
;;
;; A computation of the places at which flip bifurcation occurs in
;; the quadratic family.
;;
;;Define the map f: (z,c)->(z^2+c,c)
;;
(setf f (cdr (string-read-poly "[z^2+c,c]" '(z c))))
;;
;;Define the identity map as a polynomial
;;
(setf id (cdr (string-read-poly "[z,c]" '(z c))))
;;
;;Define a constant polynomial 1 in variables z and c
;;
(setf one (string-read-poly "1" '(z c)))
;;
;;(f-composition n) returns f o f o ... o f (n-times) as polynomial
;;
(defun f-composition (n) (poly-dynamic-power f n))
;;
;; g = f^n-id
;;
(defun g (n) (car (mapcar #'poly- (f-composition n) id)))
;;
;; (f^n)' (derivative over z = 0-th variable)
;;
(defun df (n) (car (partial (f-composition n) 0)))
;;
;; Flip bifurcations occur when derivative is -1 at some fixed point
;; (ideal n) is the ideal spanned by f^n-id and f'+1 and its zeros
;; are clearly the locus of flip bifurcations
;;
(defun ideal (n) (list (g n) (poly+ (df n) one)))
;;
;; Printer of the n-th ideal
;;
(defun print-ideal (n) (poly-print (cons '[ (ideal n)) '(z c)) (terpri))
;;
;; Eliminate z from the equations, because we are just after the values of
;; the parameter c
;;
(defun bifurcation (n)
  (mapcar #'poly-contract (elimination-ideal (ideal n) 1)))
;;
;; Print the polynomial whose zeros are the values of c
;; for which flip bifurcation occurs
;;
(defun print-bifurcation (n)
  (poly-print (cons '[ (bifurcation n)) '(c))
  (terpri))
```

by a list of polynomials `f`. The j-th elimination ideal of an ideal $I \subset k[\boldsymbol{x}]$ is the intersection $I \cap k[x_{j+1}, x_{j+2}, \ldots, x_n]$.

The distribution of *CGBLisp* also contains a modified version to a bifurcation problem in two dimensions with two parameters, the Hénon map: $H_{a,b}(x, y) = (a - x^2 + by, x)$.

TABLE 14
A CGBLisp *session illustrating the flip bifurcation example.*

```
CGB-LISP(26): (load "../examples/bifurcation")
; Loading ../examples/bifurcation.lisp
Args:[ Z^2 + C, C ]
Args:[ Z, C ]
Args:1
T
CGB-LISP(27): (select-grobner-algorithm :sugar)
GROBNER::BUCHBERGER-WITH-SUGAR
CGB-LISP(28): (dolist (i '(1 2 3 4)) (print-bifurcation i))
[ 4 * C + 3 ]
[ 4096 * C^3 + 9984 * C^2 + 7984 * C + 2125 ]
[ 16777216 * C^6 + 83886080 * C^5 + 223084544 * C^4 + 644063232 * C^3
+ 936223744 * C^2 + 968578240 * C + 1490365625 ]
[ 281474976710656 * C^12 + 2885118511284224 * C^11 -
14196285008401924096 * C^10 - 117192578793725755392 * C^9 -
17977235568944125837312O * C^8 - 98730918364257582055424O * C^7 +
18105190024753243391686344704 * C^6 + 76953991785035325551472017408 *
C^5 - 11413537850042676262995533974732O * C^4 -
31411999101116702091577367008051 2 * C^3 -
57687976065049875559641715250215910 40 * C^2 -
72107488880573546992855597783617749 76 * C +
72800277498483204244381516476647262692361 ]
NIL
```

20. Final remarks. We have to admit that most examples of interest are beyond the capacity of current computers. However, in recent years there has been substantial progress in both computing technology and algorithms for calculating Gröbner bases. The main progress in this area should be achieved by the ongoing development of parallel algorithms. Also, modular methods are used for Gröbner bases with integer coefficients.

We remark that *CGBLisp* was originally developed to study the following unsolved problem in analysis:

PROBLEM 5. *Is there a billiard with a positive Lebesgue measure set of periodic orbits of period n?*

The solution for $n = 3$ was given by the author in 1986 [5]. The calculation can be performed by hand although it was originally performed by automatic computation.

The theoretical complexity bounds in Gröbner basis calculations are notoriously very pessimistic which renders them useless. In practice, some classes of problems can be handled quite easily while others cannot be solved in a reasonable amount of time. Still, there is progress to be made to incorporate in the algorithms heuristic methods which allow a human being to solve many of the problems which a straightforward automatic computation cannot handle. Computing Gröbner bases in practice runs both into time and memory constraints even on seemingly simple examples.

We note that modern algebraic geometry literature gradually incorporates the algorithmic approach, for instance [4]. Thus there is hope that this interest will generate more powerful algorithms in the future.

21. An extremely brief review of the literature. An excellent introduction to Computational Algebraic Geometry is contained in [2]. The book [1] gives a more complete account of various Gröbner basis related algorithms. In particular, the algorithm for computing a Gröbner basis of the radical of an ideal is given there. The Comprehensive Gröbner Basis algorithm was introduced in [7]. The version of the CGB algorithm implemented in *CGBLisp* is described in the dissertation of W. Dunn [3] and in *CGBLisp* documentation. A standard Common Lisp reference is [6] and it is currently available as an HTML document to be accessed at the following World Wide Web site:

`http://heureka.elte.hu/cltl/clm/clm.html`

22. A review of available software. Many symbolic computations systems can calculate Gröbner bases, for instance Maple, Mathematica and Macsyma. Version 3.0 of Mathematica has a greatly improved Gröbner basis package. Publicly available implementations are available in Macauley, MAS and others. *CGBLisp* is currently fully available in source form from the Web site:

`http://alamos.math.arizona.edu`

Several implementations of the Comprehensive Gröbner Basis algorithm are available today. The original implementation used the MAS system. Another implementation uses Axiom.

A parallel C++ implementation of the Gröbner basis algorithm is contained in the package GB available from the following Web site:

`http://posso.ibp.fr/Gb.html`

A number of European organizations are involved in the *PoSSo* project, which includes development of a high performance C++ library for Gröbner basis calculation. The information about the project is available at the Web site:

`http://janet.dm.unipi.it`

This account is by no means complete. An internet search using "Grobner" or "Groebner" as keyword reveals vast resources related to Gröbner basis calculations.

REFERENCES

[1] T. Becker and V. Weispfenning, *Gröbner bases—A Computational approach to Commutative Algebra*, Springer-Verlag, New York, Berlin, Heidelberg, 1993.

[2] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties and Algorithms—An Introduction to Algebraic Geometry and Commutative Algebra*, Springer-Verlag, New York, Berlin, Heidelberg, 1992.

[3] W. M. Dunn III, *Algorithms and Applications of Comprehensive Groebner Bases*, Ph. D. dissertation, University of Arizona, Tucson, 1995.

[4] D. Eisenbud, *Commutative Algebra with a View Toward Algebraic Geometry*, Springer-Verlag, New York, Berlin, Heidelberg, 1995.

[5] M. Rychlik, *Periodic points of period three of the billiard ball map in a convex domain have measure 0*, J. of Diff. Geometry, **30**, 1989, pp. 191–205.

[6] G. L. J. STEELE, *Common Lisp: The Language*, Digital Press, Burlington, MA., 1984.
[7] V. WEISPFENNING, *Comprehensive Gröbner bases*, Journal of Symbolic Computation, **14**, 1992, pp. 1–29.