THE IMA VOLUMES
IN MATHEMATICS
AND ITS APPLICATIONS VOLUME 118

Rafael de la Llave Linda R. Petzold
Jens Lorenz
Editors

Dynamics of Algorithms

Springer

The IMA Volumes in Mathematics and its Applications

Volume 118

Series Editor
Willard Miller, Jr.

Springer Science+Business Media, LLC

Institute for Mathematics and its Applications IMA

The **Institute for Mathematics and its Applications** was established by a grant from the National Science Foundation to the University of Minnesota in 1982. The IMA seeks to encourage the development and study of fresh mathematical concepts and questions of concern to the other sciences by bringing together mathematicians and scientists from diverse fields in an atmosphere that will stimulate discussion and collaboration.

The IMA Volumes are intended to involve the broader scientific community in this process.

Willard Miller, Jr., Professor and Director

* * * * * * * * * *

IMA ANNUAL PROGRAMS

1982–1983	Statistical and Continuum Approaches to Phase Transition
1983–1984	Mathematical Models for the Economics of Decentralized Resource Allocation
1984–1985	Continuum Physics and Partial Differential Equations
1985–1986	Stochastic Differential Equations and Their Applications
1986–1987	Scientific Computation
1987–1988	Applied Combinatorics
1988–1989	Nonlinear Waves
1989–1990	Dynamical Systems and Their Applications
1990–1991	Phase Transitions and Free Boundaries
1991–1992	Applied Linear Algebra
1992–1993	Control Theory and its Applications
1993–1994	Emerging Applications of Probability
1994–1995	Waves and Scattering
1995–1996	Mathematical Methods in Material Science
1996–1997	Mathematics of High Performance Computing
1997–1998	Emerging Applications of Dynamical Systems
1998–1999	Mathematics in Biology
1999–2000	Reactive Flows and Transport Phenomena
2000–2001	Mathematics in Multimedia
2001–2002	Mathematics in the Geosciences

Continued at the back

Rafael de la Llave Linda R. Petzold
Jens Lorenz
Editors

Dynamics of Algorithms

With 30 Illustrations

Springer

Rafael de la Llave
Department of Mathematics
University of Texas
Austin, TX 78712-1802, USA

Linda R. Petzold
Department of Mechanical and
Environmental Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106-5070, USA

Jens Lorenz
Department of Mathematics and Statistics
University of New Mexico
Albuquerque, NM 87131, USA

Series Editor:
Willard Miller, Jr.
Institute for Mathematics and its
Applications
University of Minnesota
Minneapolis, MN 55455, USA

Mathematics Subject Classification (1991): 34C35, 58F05, 58F10, 58F23, 58FXX
65C20, 65L05, 65L20

Library of Congress Cataloging-in-Publication Data
Dynamics of algorithms / editors, Rafael de la Llave, Linda R.
Petzold, Jens Lorenz
p. cm. — (The IMA volumes in mathematics and its
applications ; 118)
Includes bibliographical references.
ISBN 978-1-4612-7073-7 ISBN 978-1-4612-1274-4 (eBook)
DOI 10.1007/978-1-4612-1274-4
1. Algorithms Congresses. 2. Differentiable dynamical systems
Congresses. I. Llave, Rafael de la. II. Petzold, Linda Ruth.
III. Lorenz, Jens, 1949– . IV. Series: IMA volumes in mathematics
and its applications ; v. 118.
QA9.58.D96 1999
511'.8—dc21 99-42675

Printed on acid-free paper.

Production managed by Allan Abrams; manufacturing supervised by Erica Bresler.
Camera-ready copy prepared by the IMA.

9 8 7 6 5 4 3 2 1

ISBN 978-1-4612-7073-7

FOREWORD

This IMA Volume in Mathematics and its Applications

DYNAMICS OF ALGORITHMS

is based on the proceedings of a workshop with the same title. The workshop was an integral part of the 1997–98 IMA program on "EMERGING APPLICATIONS OF DYNAMICAL SYSTEMS."

I would like to thank Rafael de la Llave, University of Texas-Austin (Mathematics), Linda R. Petzold, University of California-Santa Barbara (Mechanical and Environmental Engineering), and Jens Lorenz, University of New Mexico (Mathematics and Statistics) for their excellent work in editing the proceedings.

I also take this opportunity to thank the National Science Foundation (NSF), and the Army Research Office (ARO), whose financial support made the workshop possible.

Willard Miller, Jr., Professor and Director

PREFACE

Algorithms and dynamics reinforce each other since iterative algorithms can be considered as a dynamical system: a set of numbers produces another set of numbers according to a set of rules and this gets repeated. Issues such as convergence, domains of stability etc. can be approached with the methods of dynamics.

On the other hand, the study of dynamics can profit from the availability of good algorithms to compute dynamical objects.

Fundamental concepts such as entropy in dynamical systems and computational complexity seem remarkably related.

This interaction has been apparent in the study of algorithms for numerical integration of ordinary differential equations and differential algebraic equations from the beginning (Newton already worried how to compute numerical solutions of ODE's) and in other areas such as linear algebra, but it is spreading to more areas now, and deeper tools from one field are being brought to bear on the problems of the other.

This collection of papers represents the talks given by the participants in a workshop on "Dynamics of Algorithms" held at the IMA in November 1997. We hope that it can give a feel for the excitement generated during the workshop and that it can help to further the interest in this important and growing area full of fruitful challenges.

Rafael de la Llave
Department of Mathematics
University of Texas-Austin

Linda R. Petzold
Department of Mechanical and Environmental Engineering
University of California-Santa Barbara

Jens Lorenz
Department Mathematics and Statistics
University of New Mexico

CONTENTS

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 $LT(f_i) \big| LT(p)$ **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 $\boldsymbol{x}^\gamma = LCM(LM(f), LM(g))$ be the least common

TABLE 2
An example of the division algorithm.

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

multiple of the leading monomials. Then

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

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 $G := G'$

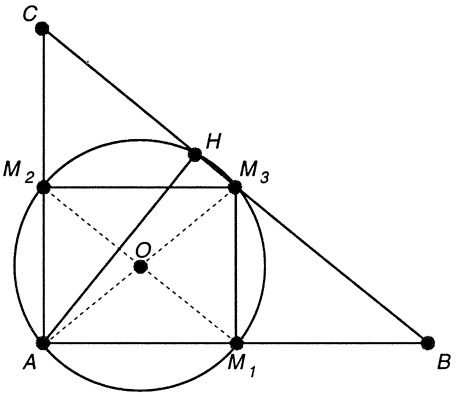

FIG. 1. *An illustration of the Apollonius Circle Theorem.*

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
```