6

Algorithms and their complexities

Juan Sabia

Departamento de Matemática - Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires and CONICET, Argentina, jsabia@dm.uba.ar

Summary. This chapter is intended as a brief survey of the different notions and results that arise when we try to compute the algebraic complexity of algorithms solving polynomial equation systems. Although it is essentially self-contained, many of the definitions, problems and results we deal with also appear in many other chapters of this book. We start by considering algorithms which use the dense representation of multivariate polynomials. Some results about the algebraic complexities of the effective Nullstellensatz, of quantifier elimination processes over algebraically closed fields and of the decomposition of algebraic varieties when considering this model are stated. Then, it is shown that these complexities are essentially optimal in the dense representation model. This is the reason why a change in the encoding of polynomials is needed to get better upper bounds for the complexities of new algorithms solving the already mentioned tasks. The straight-line program representation for multivariate polynomials is defined and briefly discussed. Some complexity results for algorithms in the straight-line program representation model are mentioned (an effective Nullstellensatz and quantifier elimination procedures, for instance). A description of the Newton-Hensel method to approximate roots of a system of parametric polynomial equations is made. Finally, we mention some new trends to avoid large complexities when trying to solve polynomial equation systems.

6.0 Introduction and basic notation

The fundamental problem we are going to deal with, as in most other chapters of this book, is to solve (over the field of complex numbers $\mathbb{C}$) a system of multivariate polynomial equations with coefficients in the field of rational numbers $\mathbb{Q}$ algorithmically, but our particular point of view is related to the question of whether we can predict how long our algorithms will take. Of course, we should define what it means to solve such a system. A first possible answer would be to decide whether there are any solutions to the given system, and, in case there are solutions, to describe them in a 'useful' or at least in an 'easy' way.

Many attempts to do this are based on trying to transform our problem into a linear algebra one. The reason for this is that we know how to solve many linear algebra problems effectively.

The focus of our attention will be the *algorithmic* solutions to these problems; so, we are going to define what an algorithm is for us (perhaps a rather inflexible definition but necessary to meet the requirements of our work). Roughly speaking, the less time an algorithm takes to perform a task, the better. This will lead to the definition of *algebraic complexity*, a kind of measure for the time an algorithm takes to perform what we want it to.

One of the problems we have when we deal with multivariate polynomials is that the known effective ways to factorize them take a lot of time, so we will try not to use this tool within our algorithms.

In the different sections of this chapter, we are going to state the problems that will be taken into account and describe (or just mention, if the description is beyond the scope of this survey) some ways of solving them.

Before we begin considering the problems, we need to fix some notation and give some definitions:

A *system of polynomial equations* is a system

$$\begin{cases} f_1(x_1,\dots,x_n)=0 \\ \qquad\dots \\ f_s(x_1,\dots,x_n)=0 \end{cases}$$

where $f_1,\dots,f_s$ are polynomials in $\mathbb{C}[X_1,\dots,X_n]$ and the solutions considered will be vectors $(x_1,\dots,x_n)\in\mathbb{C}^n$. Whenever we want to speak about a group of variables or a vector, we often use just a capital or lower case letter with no index; for example in this case, we could have written $\mathbb{C}[X]$ or $x\in\mathbb{C}^n$. The set $V\subset\mathbb{C}^n$ of all the solutions of such a system will be called an *algebraic variety* (or simply a *variety* if the context is clear). Its *dimension* is the minimum number of *generic* hyperplanes such that their common intersection with V is empty. For example, a point has dimension zero (a generic hyperplane does not cut it); a line has dimension one (a generic hyperplane cuts it, but two generic hyperplanes do not), etc. For a more precise definition of dimension see, for example, [Sha77] or [CLO97].

From the algorithmic point of view, we deal exclusively with polynomials with coefficients in $\mathbb{Q}$ but we still consider all the solutions to our systems in $\mathbb{C}^n$.

Sometimes it will be useful to take into account fields other than $\mathbb{Q}$ and $\mathbb{C}$. If k is a field, $\overline{k}$ will denote an algebraic closure of k.

6.1 Statement of the problems

In this section we are going to state some of the questions we usually want to answer when dealing with systems of polynomial equations. Some of these problems are also mentioned or studied in other chapters of this book, but we present them here for the sake of this chapter being self-contained.

6.1.1 Effective Hilbert's Nullstellensatz

Let $X = \{X_1, \dots, X_n\}$ be indeterminates over $\mathbb{Q}$. Given s polynomials $f_1, \dots, f_s \in \mathbb{Q}[X]$, if we want to solve the system of polynomial equations

$$\begin{cases} f_1(x_1, \dots, x_n) = 0 \\ \quad\quad\quad \dots \\ f_s(x_1, \dots, x_n) = 0 \end{cases}$$

the very first question we would like to answer is whether there exists any point $(x_1, \dots, x_n) \in \mathbb{C}^n$ satisfying this system (that is to say, if the equations $f_1 = 0, \dots, f_s = 0$ share a common solution in $\mathbb{C}^n$).

When all the polynomials $f_1, \dots, f_s$ have degrees equal to 1, the system we are dealing with is a linear system and there is a simple computation of ranks of matrices involving the coefficients of the polynomials which answers our question:

Suppose our linear system is given by $A.x^t = B$ (with $A \in \mathbb{Q}^{s\times n}$ and $B \in \mathbb{Q}^{s\times 1}$). Then

$$\exists x \in \mathbb{C}^n \;/\; A.x^t = B \iff \operatorname{rank}(A) = \operatorname{rank}(A|B)$$

(where $(A|B)$ denotes the matrix we obtain by adding the column B to the matrix A).

The first step towards a generalization of this result when we deal with polynomials of any degree (generalization in the sense that it relates the existence of solutions to some computations involving the coefficients of the polynomials considered) is the following well-known theorem:

Theorem 6.1.1. *(Hilbert's Nullstellensatz) Let $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$. Then the following statements are equivalent:*
i) $\{x \in \mathbb{C}^n \;/\; f_1(x) = \cdots = f_s(x) = 0\} = \emptyset$.
ii) There exist polynomials $g_1, \dots, g_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that $1 = \sum_{1\le i\le s} g_i.f_i$.

(See Chapter 4 for other versions of this theorem.)

A proof of this theorem can be found in almost any basic textbook on algebraic geometry (see for example [Har83], [Kun85] or [CLO97]). This result was already known by Kronecker and it essentially shows how a geometric problem (Is the variety defined as the common zeroes of a fixed set of polynomials empty?) is equivalent to an algebraic one (Is 1 an element of the ideal $(f_1, \dots, f_s)$?).

We will call an algorithm an *effective Hilbert's Nullstellensatz*, if given as input the polynomials $f_1, \dots, f_s$, the algorithm computes polynomials $g_1, \dots, g_s$ (in case they exist) such that $\sum_{1\le i\le s} g_i.f_i = 1$.

Later on, we will mention some effective Hilbert's Nullstellensätze.

6.1.2 Effective equidimensional decomposition

Supposing we already know that a particular system of polynomial equations has solutions, we may need to answer some questions about the geometry of the algebraic variety they define in $\mathbb{C}^n$: Does it consist only of finitely many points? Is there a whole curve of solutions? Are there isolated solutions?, etc.

All these questions can be answered by means of geometric decompositions of the algebraic variety defined by the original system of polynomials. These decompositions we are going to define are intimately bound up with the primary decomposition of ideals considered in Chapter 2 and Chapter 5 but they do not coincide because our approach is exclusively geometric while these others are purely algebraic.

Definition 6.1.2. *An algebraic variety $C \subset \mathbb{C}^n$ is called* **irreducible** *if it satisfies*

$$C = C_1 \cup C_2 \text{ where } C_1 \text{ and } C_2 \text{ are algebraic varieties} \Rightarrow C = C_1 \text{ or } C = C_2.$$

The following is a classical result from algebraic geometry. It states that the affine space $\mathbb{C}^n$ is a Noetherian topological space when considering the Zariski topology (that is, the topology in which the algebraic varieties are the closed sets) and its proof can be found, for example, in [Sha77] or [CLO97].

Proposition 6.1.3. *(Irreducible decomposition) Let $V \subset \mathbb{C}^n$ be an algebraic variety. Then, there exist unique irreducible varieties $C_1, \dots, C_r$ such that $C_i \not\subset C_j$ if $i \neq j$ and*

$$V = \bigcup_{1\le i\le r} C_i.$$

From our point of view and our definitions, the irreducible decomposition is not algorithmically achievable. If this were so, just by considering the case $n = 1$, we would be able to find all the roots of any univariate rational polynomial (note that the irreducible decomposition of $\{x \in \mathbb{C} \ / \ \prod_{1\le i\le d}(x - \alpha_i) = 0\}$ is exactly $\bigcup_{1\le i\le d}\{\alpha_i\}$). This is the reason why we are going to consider a less refined decomposition of a variety.

Let $V = \bigcup_{1\le i\le r} C_i$ be the irreducible decomposition of the variety V and, for every $0 \le j \le n$, consider the union of all the irreducible components of V of dimension j

$$V_j := \bigcup_{\substack{\{i \ / \ 1\le i\le r \\ \text{and} \dim C_i = j\}}} C_i.$$

It is obvious that $V = \bigcup_{0\le j\le n} V_j$ where, for every $0 \le j \le n$, either $V_j = \emptyset$ or $\dim V_j = j$. This unique decomposition is called the *irredundant equidimensional decomposition* (or *equidimensional decomposition* for short) of V.

Note that the information given by this decomposition still allows us to answer all the questions we asked above. For example, a non-empty variety

V consists only of finitely many points if and only if $V_0 \neq \emptyset$ and $V_j = \emptyset$ for every $1 \leq j \leq n$.

The equidimensional decomposition has the following property, nice from the algorithmic point of view:

Proposition 6.1.4. *Let $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$ be polynomials and let $V \subset \mathbb{C}^n$ be the algebraic variety of their common zeroes. If V_j is one of the components appearing in the irredundant equidimensional decomposition of V, then there exist polynomials in $\mathbb{Q}[X_1, \dots, X_n]$ defining V_j.*

The core of this result is that there are **rational** polynomials defining V_j, so we have a chance to compute the irredundant equidimensional decomposition algorithmically using only rational coefficients.

We will call an algorithm an *effective equidimensional decomposition algorithm* if given an algebraic variety $V \subset \mathbb{C}^n$ defined by rational polynomials, the algorithm describes the varieties involved in its equidimensional decomposition (i.e. its *equidimensional components*) as separate varieties.

A final comment has to be made about the irreducible decomposition of a variety defined by **rational** polynomials: we could take into account only varieties defined by rational polynomials as closed sets to define the *rational Zariski topology* in $\mathbb{C}^n$. If this is the case, the irreducible components of a variety will be still definable by rational polynomials. For example, in the case of the variety defined by a squarefree polynomial, its rational decomposition will essentially coincide with the factorization of the considered polynomial, but as we have stated before, we do not want to deal with polynomial factorization, and this is why we are not going to consider this problem. (For an algorithm yielding this irreducible decomposition numerically, see Chapter 8.)

6.1.3 Effective quantifier elimination

Many interesting geometric and algebraic problems can be formulated as first order statements over algebraically closed fields and a well-known result from logic states that any first order formula in the language of algebraically closed fields is equivalent to another formula without quantifiers (see [CK90] for details). This is the reason why, in the last decades, special efforts have been made to find efficient algorithms to eliminate quantifiers.

For the sake of simplicity, we will state precisely what elimination of quantifiers means only in a very particular case:

Theorem 6.1.5. *Let $X_1, \dots, X_n, Y_1, \dots, Y_m$ be indeterminates over $\mathbb{Q}$ and let $f_1, \dots, f_s$, $g_1, \dots, g_t \in \mathbb{Q}[X_1, \dots, X_n, Y_1, \dots, Y_m]$ be polynomials. Let*

$$V := \{x \in \mathbb{C}^n \;/\; \exists y \in \mathbb{C}^m \;:\; f_1(x,y) = 0 \;\wedge\; \dots \;\wedge\; f_s(x,y) = 0 \;\wedge$$

$$\wedge\; g_1(x,y) \neq 0 \;\wedge\; \dots \;\wedge\; g_t(x,y) \neq 0\}.$$

Then, there exists a **quantifier free** *formula φ involving only polynomials in $\mathbb{Q}[X_1, \dots, X_n]$, equalities, inequalities and the symbols $\wedge$ and $\vee$ such that*

$$V = \{x \in \mathbb{C}^n \ / \ \varphi(x)\}.$$

Let us give some simple examples to make this statement clearer.

Example 6.1.6. Suppose we want to describe the set of all the polynomials of degree bounded by d in one variable that have at least a root in $\mathbb{C}$. This set is

$$V := \{(x_0, x_1, \dots, x_d) \in \mathbb{C}^{d+1} \ / \ \exists \ y \in \mathbb{C} \ : \ x_d y^d + x_{d-1} y^{d-1} + \cdots + x_0 = 0\}.$$

Evidently, the Fundamental Theorem of Algebra states that a quantifier-free way of defining V is

$$V = \{(x_0, x_1, \dots, x_d) \in \mathbb{C}^{d+1} \ / \ x_0 = 0 \vee x_1 \neq 0 \vee x_2 \neq 0 \vee \cdots \vee x_d \neq 0\}.$$

Example 6.1.7. A very well-known example of a quantifier elimination procedure from linear algebra is the use of the determinant. The set

$$V := \{(x_{ij}) \in \mathbb{C}^{n\times n} \ / \ \exists (y_1, \dots, y_n, y_1', \dots, y_n') \in \mathbb{C}^{2n} :$$

$$(y_1, \dots, y_n) \neq (y_1', \dots, y_n') \wedge \begin{cases} x_{11}y_1 + \cdots + x_{1n}y_n = x_{11}y_1' + \cdots + x_{1n}y_n' \\ \dots \\ x_{n1}y_1 + \cdots + x_{nn}y_n = x_{n1}y_1' + \cdots + x_{nn}y_n' \end{cases} \}$$

is exactly the subset of $\mathbb{C}^{n\times n}$ defined by the determinant:

$$V = \{(x_{ij}) \in \mathbb{C}^{n\times n} \ / \ \det(x_{ij}) = 0\}.$$

Example 6.1.8. The classical resultant with respect to a single variable Y between two polynomials $f_1, f_2 \in \mathbb{Q}[X_1, \dots, X_n][Y]$ monic in Y and of degree r and s respectively is another example of eliminating quantifiers (for the definition and basic properties of the classic resultant between two polynomials, some of which will be used later, see, for example, [CLO97], [Mig92], [vdW49] or [Wal62]):

$$\{x \in \mathbb{C}^n \ / \ \exists y \in \mathbb{C} \ : \ f_1(x, y) = 0 \ \wedge \ f_2(x, y) = 0\} =$$
$$= \{x \in \mathbb{C}^n \ / \ \mathrm{Res}_Y(f_1(x, Y), f_2(x, Y)) = 0\}.$$

For a more general definition of resultants as eliminating polynomials see Chapter 2 and Chapter 1.

As before, we will say that we have an *efficient quantifier elimination procedure* if we have an algorithm that, from a formula of the type

$$\exists y \in \mathbb{C}^m \ : \ f_1(x_1, \dots, x_n, y) = 0 \ \wedge \ \dots \ \wedge \ f_s(x_1, \dots, x_n, y) = 0 \ \wedge$$
$$\wedge \ g_1(x_1, \dots, x_n, y) \neq 0 \ \wedge \ \dots \ \wedge \ g_t(x_1, \dots, x_n, y) \neq 0,$$

produces a quantifier-free formula φ defining the same subset of $\mathbb{C}^n$.

6.2 Algorithms and complexity

When we speak about efficiency related to polynomial equation solving, we mean the existence of algorithms performing different tasks. But what do we call an algorithm?

The idea of algorithm we deal with is the following: given some data in a certain way (numbers, formulae, etc), an algorithm will be a sequential list of fixed operations or comparisons that ends in some logical or mathematical 'object' we would like to compute. For example, suppose you want an algorithm to solve the equation $ax = b$ with coefficients in $\mathbb{Q}$ and that you can deal with rational numbers algorithmically (that is to say, comparisons and operations between rational numbers can be performed somehow). A possible algorithm to do this would be the one shown in Figure 6.1.

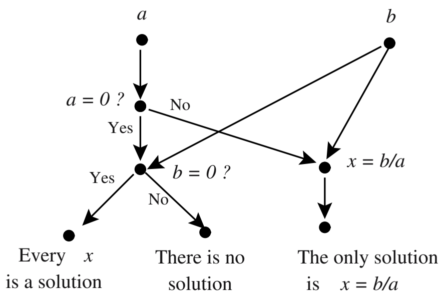

Fig. 6.1. A possible algorithm to solve the equation $ax = b$

Speaking a little more formally, our algorithms are *directed acyclic graphs.* Each node of a graph represents an element of $\mathbb{Q}$, an operation or a comparison between two elements of $\mathbb{Q}$. Each 'incoming' arrow denotes that the previously computed element or condition is needed to perform the following operation. Of course, as any graph, our algorithms have only finitely many nodes. A further comment has to be said about the graphs being 'acyclic'. As we want to predict how long our algorithms will take to compute some object, we will handle a very fixed or restricted family of algorithms: no 'WHILE' instruction is admitted in our algorithms. We can replace each 'WHILE' instruction by a 'FOR', provided we know beforehand how many times we have to repeat the procedure involved. So, an instruction of the kind 'WHILE $x > \epsilon$ DO...' is not acceptable in our algorithms, unless it can be translated into one of the type 'FOR $i = 1$ TO n DO...' and therefore 'disentangled' into a known number of sequential operations to avoid cycles in our graph.

The idea of complexity of an algorithm is related to the time it would take the algorithm to perform the desired task. The more 'complicated' our graph is, the longer the time it will take. So, a first measure of complexity to be taken into account may be the number of nodes in the graph. This will be the notion of *complexity* we are going to use throughout these notes, also known as sequential complexity.

Needless to say, this measure of complexity is not very accurate. For example, it is much simpler for a machine to perform the sum $1 + 1$ than to add two huge numbers but our measure of complexity does not take this into account. Moreover, it is generally quicker to compute a sum than a product. These considerations give place to a number of different kinds of complexities (non-scalar complexity, bit complexity, etc) which we will not take into account. But, of course, if an algorithm has a very high complexity in our terms, then it will be useless to try to run it on any computer.

There are other possible variables to be taken into account when considering the feasibility of an algorithm: for instance, the space in memory needed to perform it or whether it is well-parallelizable (that is to say, roughly speaking, whether it can be run fast enough provided we can use simultaneously a considerable number of processors at a time, or more precisely, that the depth of the algorithm is polynomial in the log of its sequential complexity). However, our approach to the subject is intended to be basic and we are not going to consider these aspects in this chapter either.

To run an algorithm, we need to encode some given data: for the moment, we will refer to the number of nodes we need to encode the input data our algorithm can deal with as the *size of the input*. This size generally depends on some quantities such as the number of variables and the number and degrees of polynomials involved. We will say an algorithm is *polynomial* when its complexity is bounded by a polynomial function in the size of the input.

We will also use the usual $\mathcal{O}$ notation to express orders of complexities: given two functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$, we say that $f = \mathcal{O}(g)$ if and only if there exists $k \in \mathbb{N}$ such that $f(x) \leq kg(x)$ for all $x \in \mathbb{N}$.

6.3 Dense encoding and algorithms

As we are trying to solve algorithmic problems involving polynomials, we need to encode them somehow. The first (and most naive) way of encoding a polynomial is to copy the usual way a polynomial is given: as a sum of monomials. To do this in a way a computer can understand it, we need to know a bound for the degree of the polynomial and the number of variables involved in advance. Then, we should order somehow all the monomials of degree less than or equal to the known bound for the degree in the number of variables involved. Once this is done, we can encode the polynomial as the vector of its coefficients in the preset order.

For example, let $f(X, Y) = X^2 - 2XY + Y^2 + 3$ be a polynomial we want to encode. As we know $\deg(f) = 2$, we only have to store the coefficients of the monomials up to this degree. We previously fix an order for all the monomials up to degree 2 in two variables, for example $(1, X, Y, X^2, XY, Y^2)$ and, using this order, the polynomial f will be encoded as $(3, 0, 0, 1, -2, 1)$.

This way of encoding polynomials is called the *dense encoding*.

Let f be a polynomial of degree bounded by d ($d \geq 2$) in n variables and let us consider how many coefficients it has, that is to say how many numbers will be needed to encode it (i.e. its size when considered as an input), provided we are given a previous monomial ordering. According to our definition, we have to compute how many monomials in n variables of degree bounded by d there are, and the exact number is $\binom{d+n}{d}$. If we consider that we are working with a fixed number of variables n but that the degrees can change, taking $d \geq 2$, we have that

$$\binom{d+n}{d} = \prod_{1 \leq i \leq n} \frac{d+i}{i} \leq 2d^n.$$

Furthermore, asymptotically in d we have that these two quantities are of the same order because

$$\frac{d^n}{\prod_{1 \leq i \leq n} \frac{d+i}{i}} \leq n!$$

and this is why we say that a polynomial of degree $d \geq 2$ in n variables has $\mathcal{O}(d^n)$ coefficients.

6.3.1 Hilbert's Nullstellensatz and dense encoding

As we have seen in Section 6.1.1, an effective Hilbert's Nullstellensatz is any algorithm that, given as input the polynomials $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$, decides whether there exist polynomials $g_1, \dots, g_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that

$$\sum_{1 \leq i \leq s} g_i \cdot f_i = 1 \tag{6.1}$$

and computes a particular solution $(g_1, \dots, g_s)$ to this identity.

The first step may be to find a bound for the possible degrees of some polynomial solutions $g_1, \dots, g_s$ to Equation (6.1) as a function of s, n and a bound d for the degrees of the polynomials $f_1, \dots, f_s$. If we are able to do so, our problem can be easily transformed into a linear algebra problem: we could write new variables for the coefficients of the polynomials $g_1, \dots, g_s$ up to the degree we found as a bound and Equation (6.1) would turn into a linear system by identifying the coefficients on the left with those on the right. That is why some authors consider the following problem an effective Hilbert's Nullstellensatz:

Show explicitly a function $\varphi : \mathbb{N}^3 \to \mathbb{N}$ satisfying the following property:

Let $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that $\deg(f_i) \leq d$ $(1 \leq i \leq s)$. *If* $1 \in (f_1, \dots, f_s)$, *there exist polynomials $g_1, \dots, g_s \in \mathbb{Q}[X_1, \dots, X_n]$ with* $\deg(g_i) \leq \varphi(n, s, d)$ $(1 \leq i \leq s)$ *such that* $\sum_{1 \leq i \leq s} g_i \cdot f_i = 1$.

In the case the polynomials we obtain by homogenizing $f_1, \dots, f_s$ have no common zeros at infinity, the Fundamental Theorem of Elimination Theory (see [Laz77] and Chapter 1, for example) shows that $\varphi(n,s,d) \le n(d-1)+1$, but in the general case this bound does not work.

Just as an example, we are going to show a very elementary result of this kind, where we obtain bounds similar to the ones obtained by G. Hermann [Her26], whose proof was corrected in [MW83].

Theorem 6.3.1. *Let* $f_1,\dots,f_s \in \mathbb{Q}[X_1,\dots,X_n]$ *such that* $\deg(f_i) \le d$ $(1 \le i \le s)$. *If* $1 \in (f_1,\dots,f_s)$, *there exist polynomials* $g_1,\dots,g_s \in \mathbb{Q}[X_1,\dots,X_n]$ *with* $\deg(g_i) \le (3d)^{2^{n-1}}$ $(1 \le i \le s)$ *such that* $\sum_{1\le i\le s} g_i.f_i = 1$.

Proof. We shall prove this theorem using induction on n.

For $n=1$, let $f_1,\dots,f_s \in \mathbb{Q}[X]$ and suppose $\deg f_1 = d \ge \deg f_i$ $(2 \le i \le s)$. If $1 = \sum_{1\le i\le s} h_i.f_i$, applying the division algorithm by f_1 in $\mathbb{Q}[X]$, we have

$$h_i = f_1.q_i + r_i \qquad (2 \le i \le s).$$

Then we obtain, rearranging the sum, that

$$1 = f_1.(h_1 + \sum_{2\le i\le s} q_i.f_i) + \sum_{2\le i\le s} f_i.r_i.$$

As $\deg r_i \le d-1$ $(2 \le i \le s)$, we have that $\deg(f_1.(h_1+\sum_{2\le i\le s} q_i.f_i)) \le 2d-1$. Therefore, calling $g_1 = h_1 + \sum_{2\le i\le s} q_i.f_i$ and $g_i = r_i$ $(2 \le i \le s)$ we get that $1 = \sum_{1\le i\le s} g_i.f_i$ and $\deg g_i \le d-1$ $(1 \le i \le s)$.

Suppose now the result is true for n. Let $f_1,\dots,f_s \in \mathbb{Q}[X_1,\dots,X_{n+1}]$ be such that $\deg(f_i) \le \deg(f_1) = d$ $(2 \le i \le s)$.

We want to deal with polynomials which are monic with respect to a variable. To do so, consider the following change of variables (where $\lambda_2,\dots,\lambda_n$ are new parameters): $X_1 = Y_1, X_2 = Y_2 + \lambda_2 Y_1, \dots, X_n = Y_n + \lambda_n Y_1$. The polynomials we obtain when applying this change of variables have maximum degree in Y_1 and their leading coefficients in this variable are the homogeneous parts of maximum degree of the original polynomials evaluated in $(1,\lambda_2,\dots,\lambda_n)$. Choosing a suitable $n-1$-tuple such that these homogeneous parts do not vanish, we get the desired linear change of variables.

So, without loss of generality, we can suppose every polynomial f_i is monic in X_1. Introduce new variables $U_1,\dots,U_s,V_1,\dots,V_s$ and consider the polynomials

$$F := \sum_{1\le i\le s} U_i f_i \quad \text{and} \quad G := \sum_{1\le i\le s} V_i f_i \quad \text{in} \quad \mathbb{Q}[U,V][X_1,\dots,X_{n+1}].$$

The resultant of these polynomials with respect to the variable X_1,

$$\mathrm{Res}_{X_1}(F,G) \in \mathbb{Q}[U,V][X_2,\dots,X_{n+1}]$$

is bi-homogeneous in the groups of variables (U,V) of bi-degree (d,d). We are going to prove that, if we write

$$\mathrm{Res}_{X_1}(F,G)=\sum_{\alpha,\beta} h_{\alpha,\beta}(X_2,\dots,X_{n+1})U^\alpha V^\beta,$$

$f_1,\dots,f_s$ have a common root in $\mathbb{C}^{n+1}$ if and only if $(h_{\alpha,\beta})_{|\alpha|=d,|\beta|=d}$ have a common root in $\mathbb{C}^n$.

If $(x_1,\dots,x_{n+1})\in\mathbb{C}^{n+1}$ is a common root of $f_1,\dots,f_s$, then

$$\mathrm{Res}_{X_1}(F,G)(x_2,\dots,x_{n+1})(U,V)=0$$

and therefore, $(h_{\alpha,\beta})_{|\alpha|=d,|\beta|=d}$ have a common root in $\mathbb{C}^n$.

On the other hand, if $(x_2,\dots,x_{n+1})$ is a common root of $(h_{\alpha,\beta})_{|\alpha|=d,|\beta|=d}$, consider the polynomials F and G in $\mathbb{Q}(U_1,\dots,U_s,V_1,\dots,V_s)[X_1,X_2,\dots,X_{n+1}]$. Then, $F(X_1,x_2,\dots,x_{n+1})$ and $G(X_1,x_2,\dots,x_{n+1})$ share a common root in $\overline{\mathbb{Q}(U,V)}$. But, as the roots of $F(X_1,x_2,\dots,x_{n+1})$ lie in $\overline{\mathbb{Q}(U)}$ and the roots of $G(X_1,x_2,\dots,x_{n+1})$ lie in $\overline{\mathbb{Q}(V)}$, the common root must be in $\mathbb{C}$. That is, there exists $x_1\in\mathbb{C}$ such that $F(x_1,\dots,x_{n+1})=0$ and $G(x_1,\dots,x_{n+1})=0$. As the variables U,V are algebraically independent, we conclude that $(x_1,\dots,x_{n+1})$ is a common root of the polynomials $f_1,\dots,f_s$.

Then we have reduced the number of variables by one. Note that, because of Hilbert's Nullstellensatz, we have shown that

$$1\in(f_1,\dots,f_s)\iff 1\in(h_{\alpha,\beta})_{|\alpha|=d,|\beta|=d}.$$

$\mathrm{Res}_{X_1}(F,G)$ can be written as a linear combination of F and G. Taking into account the degrees of the polynomials involved, we can state that there exist polynomials R and S in $\mathbb{Q}[U,V][X]$ of degree bounded by $2d^2$ in the variables X such that $\mathrm{Res}_{X_1}(F,G)=RF+SG$. Rewriting this identity into powers of U and V, we have that

$$h_{\alpha,\beta}=\sum_{1\le i\le s} p_i^{(\alpha,\beta)}f_i$$

where the polynomials $p_i^{(\alpha,\beta)}$ have degrees bounded by $2d^2$. Using the inductive hypothesis for the polynomials $(h_{\alpha,\beta})_{|\alpha|=d,|\beta|=d}$ whose degrees are bounded by $2d^2$, the theorem follows. □

Evidently, this kind of bound is not good for algorithmic purposes. There are much better bounds for the degrees of the polynomials appearing in the Nullstellensatz but the proofs are beyond the scope of this survey. Brownawell, in [Bro87], obtained the first single exponential bound $\varphi(d,n,s)=3\min\{n,s\}nd^{\min\{n,s\}}$ in the characteristic zero case. Then, in [Kol88] and [FG90] the most precise bounds known up to now for any characteristic were found: $\varphi(d,n,s)=(\max\{3,d\})^n$. In [SS95] a better bound for the particular case when $d=2$, namely $\varphi(2,n,s)=n2^{n+2}$, was shown.

More precise bounds involving other parameters than d, n and s were obtained in [Som97], [KSS97] and [GHM+98] (see Section 6.6.2).

Let us make a final comment on the complexity of an algorithm that, using the dense encoding of polynomials, decides whether the variety they define is empty or not and, if it is empty, gives as output a linear combination of the input polynomials equal to 1.

If the input polynomials $f_1, \dots, f_s$ have degrees bounded by d and the bound for the degrees of the polynomials involved in the linear combination given by the Nullstellensatz is $\varphi(d, n, s)$, then we only need to solve a system of $\mathcal{O}\left((\varphi(d,n,s)+d)^n\right)$ linear equations in $\mathcal{O}(s\varphi(d,n,s)^n)$ variables (or to prove that this system has no solution). The complexity of doing this, using the techniques in [Ber84] and [Mul87], is of order $\mathcal{O}(s^4.(\varphi(d,n,s)+d)^{4n})$.

Therefore, using the best Effective Nullstellensäzte known up to now, that essentially state $\varphi(d,n,s) = d^n$, the complexity of any algorithm using dense encoding will be at least of order $\mathcal{O}(sd^{n^2})$ (see Proposition 6.3.4 below).

6.3.2 Quantifier elimination and dense encoding

Suppose now we are given $s+t$ polynomials in $\mathbb{Q}[X_1, \dots, X_n][Y_1, \dots, Y_m]$ of degrees bounded by d and we want to give algorithmically a quantifier-free formula equivalent to

$$\exists y \in \mathbb{C}^m : f_1(x,y) = 0 \wedge \cdots \wedge f_s(x,y) = 0 \wedge g_1(x,y) \neq 0 \wedge \cdots \wedge g_t(x,y) \neq 0. \tag{6.2}$$

Rabinowicz's trick allows us to consider only equalities by means of a new indeterminate Z and therefore, the previous formula is equivalent to

$$\exists y \in \mathbb{C}^m \; \exists z \in \mathbb{C} : f_1(x,y) = 0 \wedge \cdots \wedge f_s(x,y) = 0 \wedge \big(1 - z. \prod_{1 \le i \le t} g_i(x,y)\big) = 0.$$

For a fixed $x \in \mathbb{C}^n$, using Hilbert's Nullstellensatz, this last formula is equivalent to

$$\nexists p_1, \dots, p_s, p_{s+1} \in \mathbb{C}[Y_1, \dots, Y_m, Z] \; / \; 1 = \sum_{1 \le i \le s} p_i f_i + p_{s+1}\big(1 - Z. \prod_{1 \le i \le t} g_i\big).$$

Any effective Hilbert's Nullstellensatz providing upper bounds for the degrees of the polynomials p_i involved allows us to translate this last formula into a quantifier-free formula in the coefficients of the polynomials f_i and g_j by means of linear algebra. Suppose the linear system involved is $A.X^t = B$ where $A \in \mathbb{C}^{\ell \times k}$ and $B \in \mathbb{C}^{\ell}$. The non-existence of solutions is equivalent to the condition $\operatorname{rank}(A) \neq \operatorname{rank}(A|B)$. Using that the rank of a matrix can be computed by means of the determinants of its minors, this last condition can be translated into a (very long) formula involving $\wedge$, $\vee$, equalities and inequalities to zero. This formula works for every $x \in \mathbb{C}^n$ and therefore, this formula is equivalent to (6.2).

It is evident that the better the effective Nullstellensatz we are using, the smaller the complexity of this kind of algorithm will be, provided we compute the rank of the matrices involved in a smart way (for example, using the algorithm in [Mul87]).

Given a first order prenex formula φ ('prenex' meaning that there are several blocks of existential and universal quantifiers placed at the beginning of the formula) with coefficients over an algebraically closed field, let $|\varphi|$ be its length, i.e. the number of symbols needed to encode φ, let n be the number of indeterminates involved, let D be one plus the sum of the degrees of the polynomials that appear in φ and let r be the number of blocks of quantifiers. Heintz and Wüthrich (see [Hei83] and [HW75]) exhibited elimination algorithms for algebraically closed fields of given characteristic with complexity bounded by $|\varphi|D^{n^{O(n)}}$. In fact, in the 1940s, Tarski already knew the existence of elimination algorithms but he did not describe them explicitly (see [Tar51]). Later, using the fundamental techniques described in [CG83] and [Hei83], Chistov and Grigor'ev considered the problem for prenex formulae and obtained in [CG84] and [Gri87] more precise complexity bounds of order $|\varphi|D^{n^{O(r)}}$. However, these bounds depend on arithmetic properties of the base field involved because polynomial factorization algorithms are used as subalgorithms. None of the algorithms mentioned before are efficiently well-parallelizable. Finally, in [FGM90], a well-parallelizable elimination algorithm within the same sequential complexity bounds obtained in [CG84] and [Gri87] is constructed combining the methods in [Hei83] with some effective versions of Hilbert's Nullstellensatz (see Section 6.3.1). Moreover, the complexity of this algorithm does not depend on particular properties of the base field k. Later, the same result was obtained in [Ier89]. In the context of quantifier elimination, it is also worth mentioning the work of Renegar (see [Ren92]) on elimination over real closed fields since the bounds obtained there are very sharp and imply the bounds for elimination over complex numbers.

6.3.3 Equidimensional decomposition and dense encoding

Different algorithms describing decompositions of an algebraic variety V have been given. Chistov and Grigor'ev (see [CG83]) exhibit an algorithm for the computation of the irreducible decomposition provided an algorithm that factorizes multivariate polynomials with coefficients in the base field is given. Giusti and Heintz (see [GH91]) present an algorithm for the equidimensional decomposition of algebraic varieties which is well-parallelizable. Although we do not include the proof of this last result here, we can state their main theorem and the complexity obtained:

Theorem 6.3.2. *Let $f_1, \dots, f_s$ be polynomials in $\mathbb{Q}[X_1, \dots, X_n]$ of degree bounded by d and let V be the variety they define. There exists an algorithm of complexity $s^5 d^{O(n^2)}$ which computes, for every $0 \le i \le n$, $d^{O(n^2)}$ polynomi-*

als of degree bounded by d^n defining the equidimensional component of V of dimension i.

A more recent algorithm to decompose an algebraic variety using Bézoutian matrices can be found in [EM99a]. However, the decomposition obtained there may not be minimal (embedded components may appear) and the algorithm is probabilistic (see Section 6.6.1).

6.3.4 A lower bound

In this section, we are going to show that the better bounds already obtained (and mentioned before) for the efficient Hilbert's Nullstellensatz are of the best possible order.

To do so, we are going to state a very well-known example by Masser and Philippon (see [Bro87]) that gives a very high lower bound for the degrees of the polynomials appearing in the Nullstellensatz:

Example 6.3.3. Take the following polynomials in $\mathbb{Q}[X_1, \dots, X_n]$:

$$f_1 = X_1^d, f_2 = X_1 - X_2^d, \dots, f_{n-1} = X_{n-2} - X_{n-1}^d, f_n = 1 - X_{n-1}X_n^{d-1}.$$

If $g_1, \dots, g_n \in \mathbb{Q}[X_1, \dots, X_n]$ are polynomials such that $1 = \sum_{1 \le i \le n} g_i f_i$, consider a new variable T and evaluate the polynomials in the following vector of elements in $\mathbb{Q}(T)$:

$$(T^{(d-1)d^{n-2}}, \dots, T^{d-1}, 1/T).$$

Note that, under such evaluation, all the polynomials f_i vanish for $2 \le i \le n$ and so we have that

$$1 = g_1\big(T^{(d-1)d^{n-2}}, \dots, T^{d-1}, 1/T\big)T^{(d-1)d^{n-1}}.$$

This identity implies that $\deg_{X_n}(g_1) \ge (d-1)d^{n-1}$ and therefore $\deg g_1 \ge (d-1)d^{n-1}$.

This simple example shows that, with the notation above, a lower bound for the degrees of the polynomials g_i appearing in the expression $1 = \sum_{1 \le i \le s} g_i.f_i$ is $d^{\mathcal{O}(n)}$, and therefore we have

Proposition 6.3.4. *Any general algorithm that, from an input of s polynomials $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$ of degrees bounded by d, computes (provided they exist) polynomials $g_1, \dots, g_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that $1 = \sum_{1 \le i \le s} g_i.f_i$ and encodes them in dense form must have complexity of order at least $\mathcal{O}(d^{n^2})$.*

Moreover, in [FGM90], it is shown that, from the point of view of overall complexity, the complexities they attain for the quantifier elimination algorithm are optimal when using dense encoding. In fact, they prove the following

Theorem 6.3.5. *There exists a sequence of first order formulae (containing quantifiers and two free variables)* φ_k $(k \in \mathbb{N})$ *over an algebraically closed field with the following properties:*

- $|\varphi_k| = \mathcal{O}(k)$
- *For each quantifier free formula* θ *equivalent to* φ_k *involving the polynomials* $F_1, \ldots, F_s$*, there exists* i*,* $1 \leq i \leq s$*, such that* $\deg F_i \geq 2^{2^{ck}}$*, where* $c > 0$ *is a suitable constant.*

Note that this theorem states lower bounds for the degrees of the polynomials appearing in the output formula, and the greater the degrees, the greater the number of nodes needed to encode them.

6.4 Straight-line Program encoding for polynomials

6.4.1 Basic definitions and examples

The comments in Section 6.3.4 show us that it is impossible to obtain more efficient *general* algorithms when dealing with dense encoding of polynomials. There are at least two ways of avoiding this problem: the first one is to change the form the polynomials are encoded (that is to say, to try to find a shorter way for encoding polynomials) while the second one is to design non-general algorithms which can only solve special problems but within a lower complexity. We will now discuss the first of these: changing the way we encode polynomials.

One attempt that has been made to change the representation of polynomials is the so-called ‘sparse’ encoding, which consists in specifying which monomials of a given polynomial have non-zero coefficients and which are these coefficients. Suppose a polynomial P has only a few monomials with respect to its degree. The sparse encoding will consist of a number of vectors which specify the (non-zero) coefficient of every monomial appearing in P. For example, if $P = 2X^{15}Y^4 + 2X^7Y^3 - 3X^2 + 1$, it can be encoded by a vector of four three-tuples, one for each of the monomials appearing in P. In each three-tuple, the first coefficient would stand for the degree of the monomial in X, the second one for the degree of the monomial in Y and the third one would be the coefficient of the monomial, that is, P would be encoded in the following way

$$P := ((15, 4, 2); (7, 3, 2); (2, 0, -3); (0, 0, 1))$$

instead of using a vector of $\binom{21}{2} = 210$ coordinates.

This way of encoding polynomials has proved to be efficient when dealing with particular families of polynomials (see, for example, Chapter 7 and Chapter 3) and there is a lot of theory and many algorithms that use the sparse encoding. For a complete background of this theory (including sparse resultants, Newton polytopes, toric varieties and Bernstein theorem, among other interesting and very useful notions) we suggest the reader refer to [CLO98], [GKZ94] and [Ful93].

However, it is not clear whether it is worth it to use this sparse encoding in a *general* algorithm: the output polynomials may have too many monomials. Moreover, the sparse encoding does not behave well under linear changes of coordinates in the sense that a 'short' polynomial in the sparse form can change into a very 'long' one by means of a linear change of variables: note that

$$(X+Y)^{100} = \sum_{0\le i\le 100} \binom{100}{i} X^i Y^{100-i}$$

(that is, a single monomial may turn into a polynomial with many monomials under a linear change of variables).

An alternative way to encode polynomials (the one we are going to study here) is based on the following idea:

Let P be the polynomial $P := (X+Y)^{100} - 1$. Why can we define this polynomial so easily (that is to say, using a small number of symbols) but it takes so much space to encode it for a machine (in both the sparse and the dense encoding)?

The answer perhaps is that we are used to thinking of a polynomial as a 'formal expression' rather than a function that can be evaluated. But, as far as fields of characteristic zero are concerned, polynomial functions and polynomials can be considered as the same objects. Therefore, if we define a polynomial function by defining its exact value at every point (that is to say, by means of describing how to evaluate it), we will be defining a polynomial. In the previous example, the polynomial P would be the only polynomial in $\mathbb{Q}[X,Y]$ such that, to evaluate it at a pair (x,y), you have to compute the sum of x and y to the 100-th power and subtract 1 from the result. This way of encoding a polynomial will be called a *straight-line program.* Let us put these ideas more precisely:

Definition 6.4.1. *Let $X_1,\dots,X_n$ be indeterminates over $\mathbb{Q}$ and let $R \in \mathbb{N}$. An element $\beta := (Q_1,\dots,Q_R) \in \mathbb{Q}(X_1,\dots,X_n)^R$ is a* **straight-line program** *(slp for short) if each Q_ρ satisfies one of the following two conditions:*

- $Q_\rho \in \mathbb{Q} \cup \{X_1,\dots,X_n\}$ *or*
- $\exists \rho_1,\rho_2 < \rho$ *and* $* \in \{+,-,\cdot,\div\}$ *such that* $Q_\rho = Q_{\rho_1} * Q_{\rho_2}$.

We say β is a **division-free slp** *if* $Q_\rho = Q_{\rho_1} \div Q_{\rho_2} \Rightarrow Q_{\rho_2} \in \mathbb{Q} - \{0\}$.

From now on, we are only going to deal with **division-free** slp's. Note that, in this case, each element Q_ρ is a polynomial in $\mathbb{Q}[X_1,\dots,X_n]$. If $F \in \{Q_\rho \;/\; 1 \le \rho \le R\}$, we say that β *computes* or *calculates* F.

There are several measures of complexity that can be taken into account when considering slp's. For example:

- The *total length* of β (denoted by $L(\beta)$) is the quantity of operations performed during the slp β (more precisely, it is the number of coordinates Q_ρ defined as the result of an operation between two previous coordinates).

- The *additive length* of β ($L_{\pm}(\beta)$) is the quantity of sums and subtractions performed during the slp.
- The *non-scalar length* of β ($L_{\mathbb{Q}}(\beta)$) is the number of products between two non-rational elements performed during the slp.

Given any polynomial $F \in \mathbb{Q}[X_1, \ldots, X_n]$ we will define its *total length* (also called *total complexity*) as

$$L(F) := \min\{L(\beta) \ / \ \beta \text{ is an slp computing } F\}.$$

We can respectively define $L_{\pm}(F)$ and $L_{\mathbb{Q}}(F)$.

Exercise 6.4.2. Prove that, for any $F \in \mathbb{Q}[X_1, \ldots, X_n]$, $L(F) = \mathcal{O}((L_{\mathbb{Q}}(F))^2)$.

From now on, unless it expressly stated, we will only consider the total length of an slp of a polynomial and, for the sake of shortness, we will simply call it *its length.*

As an example, we are going to show an slp that calculates the polynomial $F(X) = 1 + X + X^2 + X^3 + \cdots + X^{2^j - 1}$ efficiently. Of course, we can compute every power of X and then add them up, but it would yield an slp of length $2^{j+1} - 3$. A better slp computing F, based on the binary expansion of any positive integer up to $2^j - 1$ is the following one :

$$\beta := \Big(1, X, X^2, X^4, \ldots, X^{2^{j-1}}, 1 + X, 1 + X^2, 1 + X^4, \ldots, 1 + X^{2^{j-1}},$$
$$(1 + X)(1 + X^2), (1 + X)(1 + X^2)(1 + X^4), \ldots, \prod_{0 \le i \le j-1} (1 + X^{2^i})\Big)$$

and $L(\beta) = 3j - 2$.

Another well-known example of slp encoding a polynomial is Horner's rule for univariate polynomials:

$$a_0 + a_1 X + a_2 X^2 + \cdots + a_d X^d = (a_0 + X(a_1 + X(a_2 + X(\ldots(a_{d-1} + a_d X)\ldots)))).$$

The length of this slp is $2d$ and it involves d products and d sums. It can be proved that the number of sums and the number of products involved in *any* slp computing this polynomial are bounded by d when the elements $X, a_0, \ldots, a_d$ are algebraically independent (see, for example, [BCS97]).

Exercise 6.4.3. Let $P \in \mathbb{Q}[X, Y]$ be a polynomial whose sparse encoding is $P = ((m_1, n_1, c_1), \ldots, (m_s, n_s, c_s))$. Find a bound for $L(P)$.

Exercise 6.4.4. Find an infinite family of polynomials in $\mathbb{Q}[X, Y]$ such that the number of nodes needed to encode each of them into the sparse form is (much) greater than its length.

Exercise 6.4.5. Given a generic polynomial of degree d in $\mathbb{Q}[X, Y]$, find an upper bound for its length.

Exercise 6.4.6. Try to generalize the three previous exercises to the case of n-variate polynomials.

A last comment has to be made about the complexity of algorithms when dealing with straight-line programs. An slp can be obviously considered as a directed acyclic graph without branchings and, therefore, it has nodes. The complexity of an algorithm using the slp encoding will be the total number of nodes, that is to say, the ones arising as operations or comparisons *plus* the internal nodes of the slp's involved.

6.4.2 Some apparent disadvantages

When we are dealing with slp's to encode polynomials, we face a fundamental problem: the same polynomial may be encoded by means of many different slp's. So, it is not straightforward to verify a polynomial identity.

Suppose you are given an slp of length L that evaluates a polynomial F in n variables of degree bounded by d. If you want to know whether $F \equiv 0$, a naive attempt would be to interpolate F, but it would take so many points to do so that the complexity of doing this would again be too large (within the same order as the number of nodes needed in the dense representation of F).

Another way to solve the problem is to find a smaller particular set of points such that two polynomials of bounded length and degree coincide if and only if they coincide when evaluated in all these points. Luckily, there is a result due to Heintz and Schnorr stating the existence of this set:

Theorem 6.4.7. *(see [HS82]) Let $\widetilde{W}(d,n,L) \subset \mathbb{Q}[X_1,\dots,X_n]$ be the set of polynomials of degree bounded by d that can be calculated by means of an slp of length L. Let $\Gamma \subset \mathbb{Q}$ be a set of $2L(1+d)^2$ elements. Then, there exists a set of points $\{\alpha_1,\dots,\alpha_m\} \subset \Gamma^n$ with $m = 6(L+n)(L+n+1)$ satisfying*

$$F \in \widetilde{W}(d,n,L) \text{ such that } F(\alpha_i) = 0 \ \forall\ 1 \le i \le m \Rightarrow F \equiv 0.$$

The set $\{\alpha_1,\dots,\alpha_m\}$ is called a *correct test sequence* or a *set of questors.* Unfortunately, we do not know how to construct such a set within a reasonable cost. A way to avoid this problem is to consider probabilistic algorithms (which we will briefly discuss later in Section 6.6.1).

Another question we can ask is how many polynomials can be evaluated easily (that is to say, can be calculated by means of short slp's). The answer again, as we are going to see now, is not very encouraging (see [Sch78] and [HS80]):

For fixed n, d and L, let us consider the set of all the polynomials $F \in \mathbb{Q}[X_1,\dots,X_n]$ with $\deg(F) \le d$ and non-scalar length $L_{\mathbb{Q}}(F) \le L$.

Observe that each of these polynomials can be computed by a 'non-scalar' slp (that is to say, the only coordinates we are taking into account in this slp are the products between non-scalar elements)

$$\beta := (\beta_{-n+1}, \ldots, \beta_0, \beta_1, \ldots, \beta_L)$$

where $\beta_{-n+i} = X_i$ $(1 \leq i \leq n)$ and, defining $\beta_{-n} := 1$,

$$\beta_k = \left(\sum_{-n \leq j \leq k-1} a_j^{(k)} . \beta_j \right) \cdot \left(\sum_{-n \leq j \leq k-1} b_j^{(k)} \beta_j \right).$$

Considering new variables $A_j^{(k)}$ and $B_j^{(k)}$ $(1 \leq k \leq L;\ -n \leq j \leq k - 1)$, there exist polynomials $Q_\alpha^{(k)} \in \mathbb{Q}[A_j^{(k)}, B_j^{(k)}]$ such that the coefficients of any polynomial F_k that can be computed in the k-th step of β are the specializations of these polynomials in some rational vectors a and b, that is to say

$$F_k = \sum_\alpha Q_\alpha^{(k)}(a, b) X^\alpha.$$

So we have

Proposition 6.4.8. *For every L, $n \in \mathbb{N}$ there exist $Q_\alpha \in \mathbb{Z}[T_1, \ldots, T_m]$ polynomials with $m = (L+n)(L+n+1)$, $\alpha \in (\mathbb{N}_0)^n$, $|\alpha| \leq 2^L$, $\deg Q_\alpha \leq 2|\alpha|L$ such that for every $F \in \mathbb{Q}[X_1, \ldots, X_n]$ satisfying $L_\mathbb{Q}(F) \leq L$,*

$$F = \sum_\alpha Q_\alpha(t) X^\alpha \textit{ for some } t \in \mathbb{Q}^m.$$

Now, we can consider the morphism obtained by evaluating the family of polynomials $(Q_\alpha\ :\ |\alpha| \leq d)$:

$$(Q_\alpha\ :\ |\alpha| \leq d) : \mathbb{C}^{(L+n)(L+n+1)} \rightarrow \mathbb{C}^{\binom{n+d}{n}}.$$

Therefore, if $F := \sum_\alpha c_\alpha X^\alpha \in \mathbb{Q}[X_1, \ldots, X_n]$ is any polynomial that has $\deg(F) \leq d$ and non-scalar length $L_\mathbb{Q}(F) \leq L$, considering it as the vector $(c_\alpha) \in \mathbb{Q}^{\binom{n+d}{n}}$, it turns out that $F \in Im(Q_\alpha\ :\ |\alpha| \leq d)$.

As a consequence, we have that, for fixed $d, n, L \in \mathbb{N}$, the set

$$W(n, d, L) := \overline{Im(Q_\alpha\ :\ \alpha \in (\mathbb{N}_0)^n,\ |\alpha| \leq d)} \subset \mathbb{C}^{\binom{n+d}{n}}$$

is a closed set that contains all the vectors of coefficients of polynomials $F \in \mathbb{Q}[X_1, \ldots, X_n]$ such that $\deg F \leq d$ and $L_\mathbb{Q} \leq L$.

A very important remark is that, as $W(n, d, L)$ is defined by means of a polynomial function in $(L+n)(L+n+1)$ variables, its dimension is bounded by $\dim W(n, d, L) \leq (L+n)(L+n+1)$. This can be interpreted in the following way: the polynomials of degree bounded by d with non-scalar complexity bounded by L considered in $\mathbb{C}^{\binom{n+d}{n}}$ (a space of dimension $\binom{n+d}{n}$), lie in a variety of dimension $(L+n)(L+n+1)$. So, as long as L satisfies $(L+n)(L+n+1) < \binom{n+d}{n}$, there are very few polynomials easy to evaluate since the complement of the variety they lie in is a non-empty open set in the Zariski topology. Therefore, most polynomials are difficult to evaluate.

Taking these last observations into account, one may wonder if it would be useful to deal with slp's when trying to solve polynomial equations. The answer is affirmative as we will see in the following sections.

6.4.3 A fundamental result

In [GH93], Giusti and Heintz obtain a fundamental result using for the first time straight-line programs to solve algorithmically a problem related to solving a system of polynomial equations. In that paper, they give a **polynomial algorithm** that can decide whether a given algebraic variety V is empty or not from the polynomials defining V encoded in dense form. In fact, they go a little further: given polynomials, encoded in dense form and defining a variety V, they can find the dimension of V algorithmically in polynomial time.

In a first step, they design an algorithm that, given polynomials defining a variety V, computes a variety Z, either zero-dimensional or empty, satisfying the following conditions (V_0 will denote, as usual, the zero-dimensional equidimensional component of V):

- $V_0 \subset Z \subset V$ (that is to say, all the isolated points of V are in Z and all the points of Z are points in the variety.)
- The way Z is presented makes it 'easy' to decide whether it is empty or not (a more precise description of this way of presenting Z will be given in Section 6.4.4).

Note that, if we already know that the variety V is either empty or has dimension 0, we can decide if it is empty by means of this result ($V = \emptyset \iff Z = \emptyset$).

The general idea of the algorithm computing the dimension of V is the following: suppose the variety V is defined by $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$. Generally, if it is not empty, when we cut it with a hyperplane H_1, we will obtain a variety $V \cap H_1$ of dimension $\dim V - 1$. Continuing this process with 'generic' hyperplanes, we have that, after $\dim V + 1$ steps, by reducing the dimension by one in each step, we get the empty set. Then, as $\dim V \leq n$, when we cut it with $n+1$ 'generic' hyperplanes we obtain the empty set:

$$V \cap H_1 \cap \cdots \cap H_{n+1} = \emptyset.$$

So, we have that $V \cap H_1 \cap \cdots \cap H_n$ is either the empty set or a variety consisting only of isolated points and we are under the required hypotheses to decide whether it is empty or not. If it is not empty, then $\dim V = n$. If it is empty, we consider the variety $V \cap H_1 \cap \cdots \cap H_{n-1}$ and repeat the process. After at most $n+1$ steps we will know the dimension of V (because it is equal to the minimum number of 'generic' hyperplanes we have to cut V with to obtain the empty set minus one).

The sets of $n+1$ hyperplanes that do not satisfy the desired conditions can be considered as elements of a proper closed set in a proper affine space $\mathbb{C}^N$, that is to say the whole construction we have made works for almost every set of $n+1$ hyperplanes. This is what we meant by 'generic' hyperplanes in the last paragraph.

The proof of the result by Giusti and Heintz is beyond the scope of this survey, but we are going to take into account some of the ideas used there.

6.4.4 An old way of describing varieties: the Shape lemma

In [GH93], Giusti and Heintz use a particular way of defining zero-dimensional varieties which was already used by Kronecker (see [Kro82]). This way of presenting the variety is called *a shape lemma presentation* or a *geometric resolution* of the variety. (This same description is presented under different names in other chapters of this book: single variable representation in Chapter 2, univariate representation in Chapter 3 and shape lemma in Chapter 4.) The idea of this presentation is quite simple:

Suppose we are given a zero-dimensional variety $Z \subset \mathbb{C}^n$ defined by polynomials in $\mathbb{Q}[X_1, \ldots, X_n]$ and consisting of D points

$$x^{(1)} = (x_1^{(1)}, \ldots, x_n^{(1)}), \ldots, x^{(D)} = (x_1^{(D)}, \ldots, x_n^{(D)}).$$

Suppose also that their first coordinates are all different from one another. Therefore, we can obtain a polynomial $Q \in \mathbb{Q}[T]$ of degree D whose zeroes are exactly these first coordinates; namely

$$Q = \prod_{1 \le i \le D} (T - x_1^{(i)}).$$

Moreover, using interpolation, fixing an index j, $(2 \le j \le n)$, there exists a unique polynomial $P_j \in \mathbb{Q}[T]$ of degree bounded by $D-1$ such that $P_j(x_1^{(i)}) = x_j^{(i)}$ for every $1 \le i \le D$. Then,

$$Z = \{x \in \mathbb{C}^n \ / \ Q(x_1) = 0 \wedge x_2 - P_2(x_1) = 0 \wedge \cdots \wedge x_n - P_n(x_1) = 0\}.$$

This parametric description of Z (note that all coordinates are parametrized as functions of x_1) has the additional property of telling us how many points are in Z (this quantity coincides with the degree of Q).

The only inconvenience of this description is that we need the first coordinates of the points to be different from one another and this is not always the case. The way to solve this is to consider an affine linear form $\ell(X) = u_0 + u_1X_1 + \cdots + u_nX_n$ in $\mathbb{Q}[X_1, \ldots, X_n]$ such that $\ell(x^{(i)})$ are all different from one another (in this case we say either that ℓ is a *primitive element* of Z or that ℓ *separates the points* in Z).

Now, we are able to define what we call a *geometric resolution* of a zero dimensional variety:

Definition 6.4.9. *Let $Z = \{x^{(1)}, \dots, x^{(m)}\} \subset \mathbb{C}^n$ be a zero-dimensional variety defined by polynomials in $\mathbb{Q}[X_1, \dots, X_n]$. A* **geometric resolution** *of Z consists of an affine linear form $\ell(X) = u_0 + u_1X_1 + \cdots + u_nX_n$ in $\mathbb{Q}[X_1, \dots, X_n]$, and polynomials Q, $P_1, \dots, P_n \in \mathbb{Q}[T]$ (where T is a new variable) such that:*

- *$\ell(x^{(i)}) \neq \ell(x^{(k)})$ if $i \neq k$.*
- *$Q(T) = \prod_{1 \leq i \leq D}(T - \ell(x^{(i)}))$*
- *For $1 \leq j \leq n$, $\deg P_j \leq D - 1$ and*

$$Z = \{(P_1(\xi), \dots, P_n(\xi)) \ / \ \xi \in \mathbb{C} \text{ such that } Q(\xi) = 0\}.$$

As this description of Z is uniquely determined up to ℓ we call it **the** geometric resolution of Z associated to ℓ.

For the sake of simplicity, we also define the notion of geometric resolution for the empty set, and in this case, the polynomial Q is 1.

Although this definition is quite easy to understand, the problem underlying it is to find (given the zero-dimensional variety $Z \subset \mathbb{C}^n$ defined by polynomials $f_1, \dots, f_s$) a proper linear form and the polynomials $Q, P_1, \dots, P_n$ (note that our definition is based on the coordinates of the points in Z!).

In [GH93] Giusti and Heintz do not find the exact geometric resolution of the isolated points of a variety V but they are able to find a linear form ℓ which separates the isolated points of V, a polynomial which vanishes over the specialization of ℓ in the isolated points of V and, by means of them, they find a geometric resolution of a zero-dimensional variety Z, satisfying $V_0 \subset Z \subset V$. Given the polynomials defining V, in a first step they introduce a new variable to make a deformation in order to reduce the problem to the case of a zero-dimensional projective variety. Then, using some ideas and results of [Laz77] about the regularity of the Hilbert function of a suitable graded ring and some linear algebra algorithms ([Ber84] and [Mul87]), they obtain the characteristic polynomials of several linear maps which allow them to get the desired geometric resolution.

Note that, if the variety V is zero-dimensional, the algorithm of [GH93] computes a geometric resolution of V. For an improved and more detailed version of this construction of a geometric resolution of a zero-dimensional variety from polynomials defining it, see [KP96].

6.4.5 Newer algorithms, lower bounds

The paper we have already mentioned ([GH93]) is a milestone in the development of algorithms solving polynomial equations symbolically. The main theorem proved there is:

Theorem 6.4.10. *There exists an algorithm that, given polynomials $f_1, \dots, f_s$ in $\mathbb{Q}[X_1, \dots, X_n]$ of degrees bounded by d in the dense encoding defining an algebraic variety $V \subset \mathbb{C}^n$, computes $\dim V$ within complexity $s^{\mathcal{O}(1)}d^{\mathcal{O}(n)}$.*

Note that this result allows us to answer the first question concerning a polynomial equation system (whether its set of solutions is empty or not) just by computing its dimension.

Some of the problems stated have since been solved within polynomial time (that is to say, by means of polynomial algorithms), by using different tools.

In [GHS93], given polynomials $f_1, \dots, f_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that the variety they define is empty, a family of polynomials $g_1, \dots, g_s \in \mathbb{Q}[X_1, \dots, X_n]$ such that $1 = \sum_{1 \le i \le s} g_i . f_i$ holds is constructed. The polynomials $g_1, \dots, g_s$ have degree bounded by $d^{\mathcal{O}(n^2)}$ and are obtained in an slp encoding. The complexity of the whole algorithm is $s^{\mathcal{O}(1)} d^{\mathcal{O}(n)}$ (compare with the end of Subsection 6.3.1). In [FGS95] the same problem is re-considered by using duality theory and a complete different algorithm is designed so that the new polynomials $g_1, \dots, g_s$ obtained have degree bounded by $d^{\mathcal{O}(n)}$.

A quantifier elimination algorithm using slp's was obtained in [PS98]. The main result there is more general than the one we have stated above, but adapted to our case it would essentially mean that the elimination stated before can be done in polynomial time in the size of the input.

We can also mention polynomial algorithms for the equidimensional decomposition of varieties(see [JS02] and [Lec00]). However, these algorithms are probabilistic (see Section 6.6.1 for a brief account on probabilistic algorithms).

6.5 The Newton-Hensel method

The use of slp's as a way of encoding polynomials made it possible to adapt algorithmically a very well-known concept, the Newton-Hensel method, which can be seen as a particular version of the implicit function theorem. (Compare with the Hensel operator defined in Chapter 9.)

Let $T_1, \dots, T_m, X_1, \dots, X_n$ be indeterminates over a field $\mathbb{Q}$. Given $t \in \mathbb{C}^m$, $T - t$ will represent the vector $(T_1 - t_1, \dots, T_m - t_m)$.

Let $f_1, \dots, f_n \in \mathbb{Q}[T, X]$ be polynomials. We will denote by f the vector of polynomials $(f_1, \dots, f_n)$, by Df the Jacobian matrix of f with respect to the indeterminates X and by Jf its determinant.

Lemma 6.5.1. *Let $f_1, \dots, f_n \in \mathbb{Q}[T, X]$ and let $(t, \xi) \in \mathbb{C}^m \times \mathbb{C}^n$ such that*

$$f_1(t, \xi) = 0, \dots, f_n(t, \xi) = 0 \text{ and } Jf(t, \xi) \neq 0.$$

Then, there exists a unique n-tuple of formal power series $\mathcal{R} = (\mathcal{R}_1, \dots, \mathcal{R}_n) \in \mathbb{C}[[T - t]]^n$ such that:

- $f_1(T, \mathcal{R}) = 0, \dots, f_n(T, R) = 0$
- $\mathcal{R}(t) := (\mathcal{R}_1(t), \dots, \mathcal{R}_n(t)) = \xi.$

Proof. (This is only a sketch; for a very detailed proof of this fact, see for example [HKP+00].)

Given $f(X) = (f_1(T,X), \dots, f_n(T,X))$, we define the Newton-Hensel operator associated to it as

$$N_f(X)^t := X^t - Df(X)^{-1}.f(X)^t.$$

Note that $Jf(X)$ is not the zero polynomial (from our hypothesis, $Jf(t,\xi) \neq 0$) and, therefore, our definition makes sense.

We define the following sequence of rational functions:

$$\begin{cases} \mathcal{R}^{(0)} := \xi \\ \mathcal{R}^{(k)} := N_f(\mathcal{R}^{(k-1)}) = N_f^k(\xi) \quad \text{for} \quad k \in \mathbb{N} \end{cases}$$

The first thing to take into account is whether we can define this sequence (that is to say, if we do not try to divide by zero) but this fact can be inductively proved using that $\mathcal{R}^{(k)}(t) = \xi$.

The following conditions are fulfilled (this can be proved recursively):

- $f_i(T, \mathcal{R}^{(k)}) \in (T-t)^{2^k} \subset \mathbb{C}[[T-t]]$ for every $1 \leq i \leq n$
- $\mathcal{R}_j^{(k+1)} - \mathcal{R}_j^{(k)} \in (T-t)^{2^k} \subset \mathbb{C}[[T-t]]$ for every $1 \leq j \leq n$

where $(T-t)$ indicates the ideal in $\mathbb{C}[[T-t]]$ generated by $T_1 - t_1, \dots, T_m - t_m$.

Therefore, the sequences $(\mathcal{R}_j^{(k)})_{k \in \mathbb{N}}$ are convergent $(1 \leq j \leq n)$ and the n-tuples of their limits $\mathcal{R} := (\mathcal{R}_1, \dots, \mathcal{R}_n)$ is the vector we are looking for. □

Just to show how this works, we are going to discuss an example briefly.

Example 6.5.2. Given n polynomials of degrees $d_1, \dots, d_n$ in n variables defining a zero-dimensional variety V and for a generic linear form ℓ, we show how to compute, in many cases at least, the polynomial $Q(T)$ of Definition 6.4.9 that leads to a geometric resolution of V:

We consider generic polynomials $f_1, \dots, f_n$ of degrees $d_1, \dots, d_n$ in the variables $X_1, \dots, X_n$:

$$f_1(T,X) = \sum_{|\alpha| \leq d_1} T_\alpha^{(1)} X^\alpha$$
$$\dots$$
$$f_n(T,X) = \sum_{|\alpha| \leq d_n} T_\alpha^{(n)} X^\alpha$$

(note that each coefficient of f_i is a new variable $T_\alpha^{(i)}$).

Consider the variety $W := \{(x_1, \dots, x_n) \in \mathbb{C}^n \ / \ x_1^{d_1} - 1 = 0, \dots, x_n^{d_n} - 1 = 0\}$. Of course, we know all the points in this set: they are n-tuples of roots of unity. Let t be the vector of coefficients of the polynomials defining W. Therefore we are under the conditions needed to apply Lemma 6.5.1 because we have that, for every $\xi \in W$, (t, ξ) is a particular instance of (T, X) that satisfy the needed hypotheses (it is easy to see that in this instance, $Jf(t, \xi) \neq 0$). Then, by applying the Newton-Hensel algorithm we can approximate vectors of power series in $T - t$ which will be roots of the original system and we can do it as precisely as we want.

We will have then $\prod_{1 \leq i \leq n} d_i$ different (approximations of) vectors of power series that should be all the roots of the original system in $\overline{\mathbb{C}(T)}$ (it can be seen that the system we are dealing with has dimension zero when we think of T as a set of parameters and Bézout's theorem states that the number of solutions is bounded by $\prod_{1 \leq i \leq n} d_i$).

Suppose that, from every $\xi \in W$, we obtain the associated solution $\mathcal{R}_\xi \in \mathbb{C}[[T - t]]^n$ of the original system. Then

$$\prod_{\xi \in W} (Y - \ell(\mathcal{R}_\xi))$$

is a polynomial in $\mathbb{Q}[[T - t]][Y]$ that vanishes at every point $\mathcal{R}_\xi$. In fact, this polynomial is the polynomial of minimal degree defining the image of our original variety under the morphism

$$\begin{aligned} \overline{\mathbb{Q}(T)}^n &\to \overline{\mathbb{Q}(T)} \\ w &\mapsto \ell(w) \end{aligned}$$

As our original variety is definable with polynomials in $\mathbb{Q}(T)[X]$, this polynomial we obtain must be in $\mathbb{Q}(T)[Y]$ and therefore, by multiplying it by a fixed polynomial $h \in \mathbb{Q}[T]$ we obtain a polynomial $M \in \mathbb{Q}[T][Y]$ satisfying the following:

$$M(T, \ell(X_1, \ldots, X_n)) \in (f_1, \ldots, f_n)$$

(here we are using that the ideal the polynomials $f_1, \ldots, f_n$ define is radical).

Therefore, given n polynomials in n variables defining a zero-dimensional variety V, provided the vector of their coefficients t_0 do not lie in a hypersurface, we can obtain by evaluating $M(T, Y)$ in t_0 a non-zero polynomial $M(t_0, Y) \in \mathbb{Q}[Y]$ which specialized in the linear form ℓ vanishes over the zeroes of V. This is a fundamental step we mentioned before (see Sections 6.4.3 and 6.4.4).

Of course a lot of work has to be done to succeed in finding this polynomial. For example one should know somehow up to what precision the Newton-Hensel algorithm is needed, how to compute the polynomial h, and so on, but this is just an example of how things work.

There are two main features to be taken into account when considering the Newton-Hensel algorithm. The first one is that an approximation of the power series vector up to a given precision can be obtained in very few steps (note that to obtain the series we are looking for up to degree θ we only have to apply $\log_2 \theta$ steps of our iteration). The second one is that the Newton-Hensel method deals essentially with slp's. In fact, an algorithmic statement of the Newton-Hensel method is the following lemma (see [GHH$^+$97] for a proof):

Lemma 6.5.3. *Under the same hypotheses and notation of Lemma 6.5.1, suppose the polynomials $f_1, \ldots, f_n$ have degree bounded by d and are given by an slp of length L. Let $\kappa \in \mathbb{N}$, then there exists an slp of length $O(\kappa d^2 n^7 L)$ which evaluates polynomials $g_1^{(\kappa)}, \ldots, g_n^{(\kappa)}, h^{(\kappa)} \in \mathbb{Q}[T][X]$ with $h^{(\kappa)}(t, \xi) \neq 0$ which represent the numerators and the denominator of the rational functions obtained in the κ-th iteration of the Newton-Hensel operator.*

The Newton-Hensel method has been successfully used to obtain more efficient algorithms to solve polynomial equation systems. This tool has been introduced in this framework for the first time in [GHM$^+$98], where an algorithm solving zero-dimensional systems was designed and an effective Nullstellensatz was stated. However, these procedures required computing with algebraic numbers. In [GHH$^+$97], the first completely rational algorithm using the Newton-Hensel method was obtained and the complexity bounds were improved in [GLS01] and in [HMW01]. The Newton-Hensel method has been extensively applied to other problems: for example, to solve parametric systems (see [HKP$^+$00] and [Sch03b]) and to obtain equidimensional decompositions of varieties (see [Lec00] and [JKSS04]). Some of these algorithms work under certain particular hypotheses while the others work for any given input probabilistically (see Section 6.6.1).

Moreover, in [Lec02] an extension of the Newton-Hensel operator adapted to the non-reduced case was presented. Then, this extension was applied to obtain an algorithm that computes the equidimensional decomposition of a variety (see [Lec03]).

All these algorithms share an important feature: they all use the Newton-Hensel operator, and therefore they can deal with input polynomials codified by means of slp's.

6.6 Other trends

In this last section, we would like to discuss briefly some ideas involved in algorithmic procedures which have been mentioned earlier.

6.6.1 Probabilistic algorithms

Sometimes our algorithms may depend on the choice of an object satisfying certain conditions (a linear form separating points, a point where a polynomial does not vanish, etc). These choices may be very expensive from the algorithmic point of view. Think of a polynomial f in n indeterminates of degree d. If we want to get a n-tuple v such that $f(v) \neq 0$ we have to check through many points. Sometimes, they may even involve a procedure we do not know how to accomplish (for example, we know we have to look for a point that is not a root of certain polynomial of bounded degree, but we cannot compute exactly the involved polynomial). To avoid this, one can choose a random point v to go on. Of course, this may lead to an error. Then, a probabilistic algorithm would be an algorithm that 'generally' performs the task we want accurately, but with a bounded probability of error.

Most algorithms involving slp's can be considered as probabilistic algorithms if we do not know an adequate correct test sequence for the kind of slp's involved. In this case, if we want to decide whether an slp represents the 0 polynomial or not, we just choose a random point and evaluate the slp in it. If the result is not zero, we are sure that the polynomial is not the zero polynomial but if it is zero, we can suppose that the polynomial is the zero one.

A clear example of this is the following (already mentioned in Section 6.4.3): We have a non-empty variety V and we want to compute its dimension. We cut it with a random hyperplane and consider what happens. Suppose that this intersection is empty. We would assume that the original variety is of dimension 0. It is generally the case, but if we are unlucky and the original variety was lying in a hyperplane parallel to the one we chose, our deduction would be false.

In most of the probabilistic algorithms we consider, the generic condition a random point should satisfy is that it is not a zero of a given polynomial $f \in \mathbb{Q}[X_1, \dots, X_n]$ of bounded degree. The random point we choose has integer coordinates taken from a finite subset of $\mathbb{N}$ big enough. The estimation of the probability of success is done by means of the following well-known result (see [Sch80] and [Zip79]):

Lemma 6.6.1. *Let $R \subset \mathbb{N}$ be a finite subset. Let $f \in \mathbb{Q}[X_1, \dots, X_n] - \{0\}$ be a polynomial. Then, for random choices of elements $a_1, \dots, a_n \in R$, we have that*

$$\mathrm{Prob}(f(a_1, \dots, a_n) = 0) \leq \frac{\deg f}{\#R}.$$

For example, some of the equidimensional decomposition algorithms already mentioned ([Lec00], [JS02], [JKSS04]) are probabilistic.

6.6.2 Non-general algorithms

In Section 6.4, we have mentioned that a possible way to avoid the high complexities involved in dense encoding was to design specific algorithms that would not work for every polynomial system but only for some of them. This is already being done, in the sense that some of the algorithms being produced in computer algebra may be general but work better (have lower complexity) in special cases. This leads us to consider other invariants (not only the degree, quantity and number of variables of the polynomials involved) to compute the complexity of the algorithms. Roughly speaking, the new invariants involved have to do somehow with the geometry of the varieties involved (that is the *semantic* features of the problem) and not with the way the variety is presented (the *syntactic* ones). For a further discussion on this topic see, for example [GHM$^+$98]. Many of the previously mentioned results deal with this new kind of invariants (see, for example, [GHH$^+$97], [KSS97], [HKP$^+$00], [Lec00] and [JKSS04]).

Acknowledgment

I would like to thank Alicia Dickenstein and Ioannis Emiris for inviting me to take part in the writing of this book. I would also like to thank Gabriela Jeronimo for her help in the writing of this chapter, and Mike Stillman for having read it thoroughly and for his comments and advice.