Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik

Skript Computeralgebra I

Wintersemester 1995/96

Jürgen Gerhard

April 17, 1996

Contents

Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik
Skript Computeralgebra I, Teil 1
Wintersemester 1995/96

In science and engineering, a successful attack on a problem will usually lead to some equations (or inequalities) that have to be solved. There are many types of such equations: differential equations, linear equations, polynomial equations, recurrences, equations in groups, etc. In principle, there are two ways of solving such equations: approximately or exactly. *Numerical analysis* is a well-developed field that provides highly successful mathematical methods and computer software to compute *approximate* solutions.

Computer algebra is a fairly recent area of computer science, where mathematical methods and computer software are developed for the *exact* solution of equations.

Why use approximate solutions at all if we can have exact solutions? The answer is that in many cases, an exact solution is not possible. This may have various reasons: for certain (simple) ordinary differential equations, one can prove that no closed-form solution (of a specified type) is possible. More important are questions of efficiency: any system of linear equations, say with rational coefficients, can be solved exactly, but for the huge linear systems that arise in meteorology, nuclear physics, geology or other areas of science, only approximate solutions can be computed efficiently. The exact methods, run on a supercomputer, would not yield answers within a few days or weeks (which is not really acceptable for weather prediction).

However, within its range of exact solvability, computer algebra usually provides more interesting answers than traditional numerical methods. Given a differential equation or a system of linear equations with a parameter t, the scientist gets much more information out of a closed form solution in terms of t than from several solutions for specific values of t.

Many of today's students may not know that the *slide rule* was an indispensable tool of engineers and scientists until the 1960's. *Electronic pocket calculators* made them obsolete within a short time. In the coming years, *computer algebra systems* will similarly replace calculators for many purposes. Although still bulky and expensive (usually requiring a full-blown computer system), these systems can easily perform exact (or arbitrary precision) arithmetic with numbers, matrices, polynomials etc. They will become an indispensable tool for the scientist and engineer, from students to the work place. Eventually these systems will be integrated with other software like numerical packages, CAD/CAM, and graphics.

The goal of this course is to give an introduction to the basic methods and techniques of computer algebra.

1. Introduction to computer algebra systems

An application of computers that is gaining more and more importance is the task of solving algebraic problems that arise in fields such as physics or engineering. Computer scientists have put together programs that can be used to solve these algebraic problems, often algebraic or differential equations.

Computer algebra systems have historically evolved in several stages. The *first generation* comprised MACSYMA from MIT in 1972, SCRATCHPAD from IBM in 1971, REDUCE by A. C. Hearn. These systems provided algebraic engines capable of doing amazing *exact* or *formal* or *symbolic* computations: differentiation, integration, factorization, etc. (The French term for computer algebra is *calcul formel*). The *second generation* started with MAPLE from the University of Waterloo in 1985 and MATHEMATICA by Steven Wolfram. They began to provide modern interfaces and graphic capabilities, and the hype surrounding the launching of MATHEMATICA did much to make these systems widely known. The *third generation* is on the market now: AXIOM by IBM and NAG, a successor of SCRATCHPAD, MAGMA by Cannon at the University of Sydney, and MUPAD by Fuchssteiner here at the University of Paderborn. These systems incorporate a categorical approach and operator calculations. MUPAD is designed to also work in a parallel environment. In July 1995, the DFG (Deutsche Forschungsgemeinschaft) agreed to fund the first "Sonderforschungsbereich" SFB 376 "Massive Parallelität" in Paderborn directed by Friedhelm Meyer auf der Heide; two of 13 projects of this multi-million-DM enterprise deal with computer algebra.

Today's research and development is driven by three goals, which sometimes conflict: wide functionality (the capability of solving a large range of different problems), ease of use (user interface, graphics display), and speed (how big a problem can you solve with a routine calculation, say a day's computing?). This course will concentrate on the latter goal. We will see the basics of the fastest algorithms available today, mainly for some problems in polynomial manipulation. Several groups have developed software for these basic operations: PARI by Cohen in Bordeaux, THALES by Buchmann in Saarbrücken, software by Lenstra and Manasse at Bellcore, by Shoup, by Kaltofen, and in development here at Paderborn (Gerhard).

A central problem is the factorization of polynomials. Enormous progress has been achieved in the last few years, say for this problem over finite fields. In 1991, the largest problems amenable to routine calculation were about 2 KB in size, while in 1995 Shoup's software can handle problems of about 0.5 MB. Almost all the progress here is due to new algorithmic ideas, and this problem will serve as a guiding light for this course.

When designing a computer algebra system, one must consider the algorithms one is going to use and how they are going to be implemented. Important implementation issues concern things like the user interface, memory management, how to use flags on subexpressions for garbage collection, and which representations or simplifications are allowed for the various algebraic objects.

In this course, we will consider arithmetic algorithms in some basic domains.

The arithmetic tasks which we will analyze include conversion between representations, addition, subtraction, multiplication, division, division with remainder, greatest common divisors, and factorization. The basic domains for computer algebra are the natural numbers $\mathbb{Z}$, the rationals $\mathbb{Q}$, finite fields, the algebraic number fields $\mathbb{Q}[x]/(f)$, the finite fields $\mathbb{Z}_p[x]/(f)$, the fields of p-adic numbers, polynomial rings $F[x_1, \ldots, x_n]$ and power series $F[[x_1, \ldots, x_n]]$. We will only consider some of these. Many other interesting algebraic problems are considered by computer algebraists, but we won't have time to look at them. They include matrix computations and the solution of systems of polynomial equations, computing with groups, character tables and various group representations, integration and summation in finite terms, isolating roots of polynomials, solving systems of polynomial equations and inequalities, combinatorial and number-theoretic questions, and quantifier elimination. A few of these will be studied in the "Computeralgebra II" course.

Computer algebra systems have a wide variety of applications in fields that require computations that are tedious, lengthy and difficult to get right when done by hand. In particular, in physics computer algebra systems are used in high energy physics, for quantum electrodynamics, quantum chromodynamics, satellite orbit and rocket trajectory computations and celestial mechanics in general. For a comparison, consider Delaunay's results on lunar theory in which he calculated the orbit of the moon under the influence of the sun and a non-spherical earth with a tilted eclipse. This work took twenty years to complete and was published in 1867. It was shown, in 20 hours on a small computer in 1970, to be correct to nine decimal places.

2. Introductory concepts

This is a review of some basic algebra concepts[1]. The simplest algebra we will be concerned with is the group.

2.1. Group theory.

DEFINITION 2.1. *A group is a set G with a binary operator $\cdot : G \times G \to G$ satisfying*

1. *Associativity:* $\forall a, b, c \in G \quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$,
2. *Identity exists:* $\exists 1 \in G \quad \forall a \in G \; a \cdot 1 = 1 \cdot a = a$,
3. *Inverse exists:* $\forall a \in G \quad \exists a^{-1} \in G \; a \cdot a^{-1} = a^{-1} \cdot a = 1$.

The group is denoted by the set name, G. When more care is needed the group is denoted by $(G; \cdot, ^{-1}, 1)$.

It is usual, for convenience of notation, to omit the symbol $\cdot$ from products. Thus $a \cdot b$ becomes the simpler ab. We will also frequently need to distinguish between two group operations. The alternate notation $+$ for $\cdot$, 0 for 1 and $-a$ instead of a^{-1} is used. The first representation is called a *multiplicative group* and the new one is called an *additive group* . The additive group could be denoted by $(G; +, -, 0)$.

EXAMPLE 2.2. *Here are some examples.*

Group Set	**Group Operation**
$\mathbb{Z} =$ *Integers*	$+$
$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$	$+$ *mod* n
$\mathbb{Z}_n^\times = \{a \in \mathbb{Z} \colon 1 \le a < n, \gcd(a, n) = 1\}$	*multiplication mod* n
$\mathbb{R}^{n \times n} =$ *real* $n \times n$*-matrices*	*matrix addition*
invertible real $n \times n$*-matrices*	*matrix multiplication*

A subset S of a group G is called a set of *generators* of G if every element of G can be written as a finite product of elements in S and their inverses. For instance consider $\mathbb{Z}_{12} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ under addition mod 12. $S = \{3, 8\}$ is a generating set for $\mathbb{Z}_{12}$. In fact $\mathbb{Z}_{12}$ has a generating set consisting of just one generator, 1. If a group has one element that generates the entire group then the group is called *cyclic* . $\mathbb{R}^{n \times n}$ has no finite set of generators.

A subset H of a group G is called a *subgroup* of G if it is closed unter multiplication and inversion, i.e., if $ab \in H$ and $a^{-1} \in H$ for all $a, b \in H$. (Exercise: Show that any subgroup contains the neutral element 1.) We write $H \le G$.

The number of elements in a group is called its *order* , written $\#G$. If G is a finite group and $H \le G$, then *Lagrange's Theorem* says that $\#G = \#H \cdot \#(G : H)$,

[1]Please report any error in these notes to the instructors. You should be able to answer the little questions in the notes on the fly. If you have a difficulty, be sure to clarify your problem (with the help of a textbook, the tutor, or an instructor).

where $G : H = \{gH : g \in G\}$ is the set of *right cosets* of H. In particular, $\#H$ is a divisor of $\#G$.

A special type of group of interest is a *commutative group* (also called *abelian group*) . These are groups with the additional property

4. Commutativity: $\forall a, b \in G\ ab = ba$.

The integers $\mathbb{Z}$ under addition form an abelian group , while the orthogonal $n \times n$ matrices under multiplication do not. (Small exercise: Show that cyclic groups are abelian .) $\mathbb{Z}_{12}^{\times}$ is a group that is not cyclic. It is abelian in fact $\mathbb{Z}_{12}^{\times} \cong \mathbb{Z}_2 \times \mathbb{Z}_2$. The proof is an exercise in unraveling the definitions.

A *group homomorphism* is a map $f : G \to H$ between two multiplicative groups G and H that respects the group operations: $f(1) = 1$, and $f(g_1 g_2) = f(g_1) f(g_2)$ for all $g_1, g_2 \in G$. A map $f : A \to B$ between two sets A and B is one-to-one or *injective* if $f(a_1) = f(a_2)$ implies $a_1 = a_2$; f is onto or *surjective* if $\forall b \in B\ \exists a \in A\ f(a) = b$. If f is both 1-1 and onto then f is called a *bijection*. A group homomorphism $f : G \to H$ that is also a bijection is called an *isomorphism* between G and H, and then G and H are *isomorphic*. As far as group theory is concerned, G and H are representations of the same group. A group homomorphism $f : G \to H$ is an isomorphism if and only if there exists a group homomorphism $g : H \to G$ so that $f \circ g = \mathrm{id}_H$ and $g \circ f = \mathrm{id}_G$.

As an exercise, show that any cyclic group G of order n is isomorphic to $\mathbb{Z}_n$ (addition mod n is understood). This is written $G \cong \mathbb{Z}_n$. There is thus essentially only one cyclic group of order n and it is usually denoted by C_n instead of $\mathbb{Z}_n$.

EXERCISE 2.3. *Let $f : G \to H$ be a homomorphism of multiplicative groups . Show that* $\ker f = \{g \in G : f(g) = 1\}$, *the* kernel *of f, is a subgroup of G and* $\operatorname{im} f = \{f(g) : g \in G\}$, *the* image *of f, is a subgroup of H.*

If G and H are groups, $f : G \longrightarrow H$ is a homomorphism , and $K = \ker f$ is the kernel of f, then $G : K$, the set of right cosets of K, forms again a group by letting $(gK)(g'K) = (gg')K$ for any $g, g' \in G$. This group is called the *factor group* of G modulo K and denoted by G/K. The *homomorphism theorem* for groups states that $G/K = G/\ker f$ is isomorphic to $\operatorname{im} f$.

Given two groups G and H, a new group $G \times H$—called the *direct product* —can be constructed. As a set $G \times H = \{(g, h) : g \in G, h \in H\}$, and multiplication is defined by $(g_1, h_1) \cdot (g_2, h_2) = (g_1 g_2, h_1 h_2)$. (What is the identity element of $G \times H$, and what is the inverse of some (g, h)?)

2.2. Ring theory. A ring is an algebraic structure with two operations, as follows.

DEFINITION 2.4. *A ring is a set R with two binary operations $\cdot, + : R \times R \to R$ satisfying*

1. *R together with $+$ is an abelian group with identity 0,*

2. *$\cdot$ is associative,*

3. *R has an identity element 1 for $\cdot$,*

4. *$\cdot$ distributes over $+$: $\forall a,b,c \in R$, $a(b+c) = (ab)+(ac)$ and $(b+c)a = (ba)+(ca)$.*

A ring is commutative if it satisfies the additional property

5. *$\forall a, b \in R$ $ab = ba$.*

EXAMPLE 2.5. *Here are some examples.*

Ring set	**$+$ Operation**	**$\cdot$ Operation**
$\mathbb{Z}$ = *set of integers*	$+$	$\cdot$
$\mathbb{R}$ = *set of real numbers*	$+$	$\cdot$
$\mathbb{Z}_n$	$+$ *mod* n	$\cdot$ *mod* n
$n \times n$ *integer matrices*	*matrix addition*	*matrix multiplication*

Only the matrices, of the above examples, form a non-commutative ring.

DEFINITION 2.6. *A ring homomorphism is a map f from a ring R to a ring S where $f(r_1 + r_2) = f(r_1) + f(r_2)$, $f(r_1 r_2) = f(r_1)f(r_2)$ for all $r_1, r_2 \in R$, and $f(0_R) = 0_S$ and $f(1_R) = 1_S$. f is an isomorphism if f is also a bijection.*

Thus a ring homomorphism is a homomorphism for both the additive and the multiplicative structure. Again isomorphic rings are considered to be essentially the same structure. Of interest are special subsets called ideals.

DEFINITION 2.7. *A* left ideal *I is a subset of the ring R satisfying*

1. *$\forall a, b \in I$ $a + b \in I$,*

2. *$\forall a \in I$ and $r \in R$ $ra \in I$.*

When R is commutative, this is called an ideal .

(Question: is every ideal a ring?) As an example, the even numbers form an ideal $I = \{.. -2, 0, 2, ..\} = \{2a : a \in \mathbb{Z}\} \subseteq \mathbb{Z}$. In general, we write $(a) = \{ra : r \in R\}$ for the *principal ideal* generated by $a \in R$. Suppose $I \subseteq R$ is an ideal, and $a, b \in R$. We say that a and b are *congruent modulo* I (written as "$a \equiv b \bmod I$") if $a - b \in I$. As an example, with $R = \mathbb{Z}$, we have $14 \equiv 2 \bmod (12)$, which we also write as $14 \equiv 2 \bmod 12$. If $a, b \in R$, we write $a|b$ ("a divides b") iff there exists $c \in R$ with $ac = b$. (Exercise: $a|b \Leftrightarrow b \in (a)$.)

A set of elements $S \subseteq R$ is a *system of representatives* if $\forall a \in R$ $\exists!$ $b \in S$ $a \equiv b \bmod I$. ("$\exists!$ b" means: there exists exactly one b.) Example: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ is a system of representatives for $(12) \in \mathbb{Z}$. There are many other such systems. The system of representatives can be made into a ring again, by using multiplication and addition modulo I. The usual notation is: R/I, read "R modulo I". There is a natural ring homomorphism $f: R \to R/I$ mapping $a \in R$ to the unique $b = f(a) \in R/I$ with $a \equiv b \bmod I$. For instance, $f(14) = 2$ in the above example. (We note that there is a more natural definition of R/I as the set of equivalence classes modulo I, but the one given above is somewhat simpler.)

As for groups, there is also a *homomorphism theorem* for rings. If R and S are commutative rings, $f: R \longrightarrow S$ is a ring homomorphism, and $I = \ker f = \{r \in R: f(r) = 0\}$ is the kernel of f, then I is an ideal of R, and R/I is isomorphic to the subring $\operatorname{im} f = \{f(r): r \in R\}$ of S, the image of f.

We can add more and more restrictions to rings, to get rings in which more interesting things can be done, for instance factoring or computing greatest common divisors. The first restriction to consider are integral domains. An *integral domain* is a non-trivial commutative ring with no zero divisors. Non-trivial means that 1 and 0 are distinct. A *zero divisor* is any non-zero element $a \in R$ for which there is another non-zero element $b \in R$ such that $ab = 0$. (Is $\mathbb{Z}_{12}$ or $\mathbb{Z}_7$ an integral domain?) In an integral domain we have the useful fact that if $a \neq 0$ and $ab = ac$ then $b = c$. This is known as the cancellation law.

For the ring of integers, we have two interesting properties.

1. Unique factorization : Every positive integer (not 0 or 1) has a unique factorization as a product of primes.

2. Division property : $\forall a, b \in \mathbb{Z}, b \neq 0, \exists! q, r \in \mathbb{Z}\ a = qb + r,\ 0 \leq r < |b|$.

We would like to generalize these results to other rings. For unique factorization, we need the concept of irreducible elements. A *unit* in an integral domain R is any element with a multiplicative inverse: u is a unit $\iff \exists v \in R\ uv = 1$. For $\mathbb{Z}$, the units are $+1$ and -1. An element a is an *associate* of an element b if there is a unit u such that $a = ub$. In $\mathbb{Z}$, $+4$ and -4 are associates. An *irreducible* is any non-unit $p \in R$ such that if $p = bc$, with $b, c \in R$, then b or c is a unit. A *unique factorization domain* (UFD) is an integral domain in which every non-zero element can be expressed as a unique (up to associates and ordering of the irreducibles) product of irreducibles.

(What are the units of the ring $\mathbb{Z} \times \mathbb{Z}$? Factor $(18, 27)$ in $\mathbb{Z} \times \mathbb{Z}$ and figure out how many different ways there are to factor it when associates are not considered the same.)

To talk about the division property we need an extra function to satisfy the role played by the absolute value function in the integer case. An integral domain D is called a *Euclidean domain* if there is a *degree function* $d: D^* \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers and $D^* = D - \{0\}$, satisfying

1. $d(ab) \geq d(a)$, if $a, b \neq 0$

2. Division property : $\forall a, b \in D\ b \neq 0, \exists\ q, r \in D$ satisfying $a = qb + r$ and $r = 0$ or $d(r) < d(b)$. q is called a *quotient* and r is called a *remainder* .

(What is a degree function on $\mathbb{Z}$? Is there a degree function for $\mathbb{Z} \times \mathbb{Z}$?) There are slight differences between the general concept for rings and the division theorem for integers stated above, but this is to be expected.

Euclidean domains are the domains in which the Euclidean algorithm for computing greatest common divisors and the Chinese Remainder Algorithm work. It is an important theorem that every Euclidean domain is a unique factorization domain.

2.3. Polynomial rings.

DEFINITION 2.8. *Given a ring R, we define the ring of polynomials $R[x]$ to be the set of vectors $a = (a_0, a_1, \ldots)$ with entries from R, where all but a finite number of the a_i's are zero. The polynomials become a ring if addition is defined by $(a_0, a_1, \ldots) + (b_0, b_1, \ldots) = (a_0 + b_0, a_1 + b_1, \ldots)$ and multiplication is defined by $(a_0, a_1, \ldots)(b_0, b_1, \ldots) = (c_0, c_1, \ldots)$, where $c_n = \sum_{i=0}^{n} a_i b_{n-i}$. The degree $\deg a$ of a nonzero polynomial a is the largest integer n such that $a_n \neq 0$. Its leading coefficient is a_n. It is more usual to denote the polynomial $(a_0, a_1, a_2, \ldots)$ by $a_0 + a_1x^1 + a_2x^2 + \cdots + a_nx^n$ where $a_k = 0$ for all $k > n$. A power series is the same as a polynomial without the restriction that at most a finite number of terms be non-zero.*

If R is a commutative ring then $R[x]$ is commutative. If R is an integral domain then so is $R[x]$. We might hope that the same holds for Euclidean domains . In fact, if R is a Euclidean domain then the polynomial degree function obeys property 1. of the Euclidean domain degree function . However, the division property goes away (say, in $\mathbb{Z}[x]$, as can be seen when you try $a = x^2 + 3$, $b = 3x + 1$). The division property holds if the leading coefficient of b is a unit of R. (Is $\mathbb{Z}_3[x]$ a Euclidean domain?)

The units of $R[x]$ are simply the units of R, where we use the natural identification of R with polynomials of degree 0. Irreducibles are a bit trickier. For instance, $x^2 + 1$ is irreducible in $\mathbb{Z}[x]$ and $\mathbb{Z}_3[x]$, but in $\mathbb{Z}_5[x]$, $x^2 + 1 = (x + 2)(x - 2)$.

There are some ring homomorphisms of interest on polynomials. The easiest is the evaluation homomorphism $f_c: R[x] \to R[x]/(x-c) \cong R$ which takes a polynomial $p \in R[x]$ to $f_c(p) = p(c) \in R$ where $c \in R$. Another ring homomorphism is based on the ring homomorphism defined by modular arithmetic on the integers $r_m: \mathbb{Z} \to \mathbb{Z}_m$. This homomorphism can be applied to polynomials to get another homomorphism $r_m: \mathbb{Z}[x] \to \mathbb{Z}[x]/(m) \cong \mathbb{Z}_m[x]$ by applying r_m to the coefficients of the polynomials.

A third homomorphism—generalizing both above—is to polynomials what modulo is to integers. If $m \in R[x]$, then we have the principal ideal $(m) \subseteq R[x]$ of multiples of m: $(m) = \{am: a \in R[x]\}$. Suppose that the leading coefficient of m is 1 (or, more generally, a unit in R). Then the polynomials $f \in R[x]$ of degree less than $\deg m$ form a system of representatives for (m), and hence they form the quotient ring $R[x]/(m)$, with addition and multiplication modulo m. We then get a homomorphism $r_m: R[x] \to R[x]/(m)$ by taking the remainder of any polynomial in $R[x]$ with divisor m. As indicated above, the evaluation morphism is really the special case with $m = x - c$.

The following fact is useful. If R and S are rings, $\varphi: R \to S$ is a ring homomorphism , $f \in R[x_1, \ldots, x_n]$ is a polynomial in n variables, and $r_1, \ldots, r_n$ are elements of R, then

$$\varphi(f(r_1,\ldots,r_n)) = \overline{f}(\varphi(r_1),\ldots,\varphi(r_n)),$$

where $\overline{f} \in S[x_1,\ldots,x_n]$ arises from f by application of φ to the coefficients. Similarly, if $\equiv$ is a congruence relation on R, f is as above, and $r_1,\ldots,r_n,r'_1,\ldots,r'_n$ are elements of R satisfying $r_i \equiv r'_i$ for $1 \le i \le n$, then $f(r_1,\ldots,r_n) \equiv f(r'_1,\ldots,r'_n)$. (Exercise: Prove this!) We say that ring homomorphisms and congruence relations *commute* with polynomial expressions.

It is easy to show that the polynomials over a Euclidean domain R form a Euclidean domain only when every non-zero element of R is a unit. This leads us to the following definition.

Definition 2.9. *A field is an integral domain in which every non-zero element is a unit, i.e., has a multiplicative inverse.*

The familiar examples are the fields $\mathbb{Q}$ of the rational numbers, the field $\mathbb{R}$ of the real numbers, and the field $\mathbb{C}$ of the complex numbers. We have $\mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$.

The number of elements in a field or ring is called its order. The above fields all have infinite order, however, there are fields of finite order too. Among them are the fields $\mathbb{Z}_p$, where p is a prime. The existence of the inverse of a non-zero element a of $\mathbb{Z}_p$ follows from the fact that from the Euclidean algorithm (as discussed later in these notes) we get $s,t \in \mathbb{Z}$ such that $1 = sa + tp$, since the $gcd(a,p)$, the greatest common divisor of a and p, is 1 for $a = 1,\ldots,p-1$. Then $sa \equiv 1 \bmod p$.

If we consider the ring $\mathbb{Z}_3 \times \mathbb{Z}_3$, then $(1,1)+(1,1)+(1,1) = (0,0)$, and the ring has order 9. This leads us to define the *characteristic* of a ring or field to be the minimum number of times the identity element can be added to itself to get 0. In the case where this can never produce zero, the ring or field is said to have characteristic zero . The reals and rationals are fields with characteristic zero

2.4. "Big Oh" notation. In many situations in algorithmics it is convenient to express the cost of an algorithm only "up to a constant factor". What is needed is not the *exact* cost for increasing input sizes but merely the "rate of growth". We say for example that the familiar multiplication algorithm for $n \times n$ matrices over a field "is an n^3 algorithm", which precisely means that the number $f(n)$ of field operations needed to multiply two matrices of dimension n is less than $c \cdot n^3$ for all $n \in \mathbb{N}$, where c is a real constant that we are not really interested in. This "sloppiness" can be formalized by means of the "Big Oh" notation, introduced by P. Bachmann and E. Landau in number theory at the turn of the century, and popularized in computer science by D. Knuth (1970). We first give a precise mathematical definition and then discuss a more common and convenient but less clean use.

Definition 2.10.
1. *A partial function $f\colon \mathbb{N} \longrightarrow \mathbb{R}$, i.e. one that need not be defined for all $n \in \mathbb{N}$, is called* eventually positive , positive almost everywhere , *or* positive a.e. *for short, if there is a constant $N \in \mathbb{N}$ such that $f(n)$ is defined and strictly positive for all $n \ge N$.*

2. *Let $g\colon \mathbb{N} \longrightarrow \mathbb{R}$ be positive a.e. Then we define*

- $O(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} : f$ *positive a.e. and* $\exists N \in \mathbb{N}, c \in \mathbb{R}_{>0}\ \forall n \geq N$ f, g *are defined and* $f(n) \leq cg(n)\}$,
- $\Omega(g) = \{f : \mathbb{N} \longrightarrow \mathbb{R} : f$ *positive a.e. and* $\exists N \in \mathbb{N}, c \in \mathbb{R}_{>0}\ \forall n \geq N$ f, g *are defined and* $f(n) \geq cg(n)\}$,
- $\Theta(g) = O(g) \cap \Omega(g)$.

Here $\mathbb{R}_{>0}$ *denotes the set of all positive real numbers.*

If $f(n)$ denotes the cost for matrix multiplication as above and $g(n) = n^3$, we may write $f \in O(g)$. In computer science and mathematics, however, the notation $f = O(g)$ or $f(n) = O(n^3)$ has been established for this. Some comments on this notation should be made.

1. The use of the equality sign is misleading because it suggests symmetry. But $f = O(g)$ is a binary predicate (it is in fact a pre-order, as we will see below) rather than an equation, with a loss of information from left to right. In particular, one should not write chains of equations with O. As an example, let f, g be as above and $h(n) = n^4$. Then $f = O(g)$ and $g = O(h)$, but we do not write $f = O(g) = O(h)$.

2. A second abuse of notation is that we often write, e.g., $n^3 = O(n^4)$. For each $n \in \mathbb{N}$, n^3 is just a number and the O-notation makes little sense for single numbers. What is meant is that $g \in O(h)$, with g, h as above. There is a notation to avoid this abuse, called the λ-calculus, but it is somewhat clumsy and we do not use it.

Analogously, we write $f = \Omega(g)$ or $f(n) = \Omega(g(n))$ for $f \in \Omega(g)$ and $f = \Theta(g)$ or $f(n) = \Theta(g(n))$ for $f \in \Theta(g)$, respectively. But we should always keep in mind the exact definition when using the equality notation.

Example 2.11. *1. Let* $f : \mathbb{N} \longrightarrow \mathbb{R}$ *with* $f(n) = 1 + 2n + 27n^4$. *Then* $f(n) = O(n^4)$, *but also* $f(n) = O(n^e)$ *for any real* $e \geq 4$ *("O" does not imply "as accurate as possible"!).*

2. $f(n) = O(n^3)$ *is definitely wrong, but* $f(n) = \Omega(n^3)$, *in fact* $f(n) = \Omega(n^e)$ *for any real* $e \leq 4$.

3. $f(n) = \Theta(n^4)$ *is a consequence of the above, and here 4 is indeed the only possible exponent, so* Θ *mirrors the "real" rate of growth of the function* f, *while* O *and* Ω *just give upper and lower bounds for the rate of growth, respectively.*

4. $h(n) = O(1)$ *if and only if* h *is bounded from above.*

Lemma 2.12. *Rules of manipulation.*

1. $f \in O(f)$ *(reflexivity).*

2. *$f \in O(g)$ and $g \in O(h) \Rightarrow f \in O(h)$ (transitivity).*
3. *$f \in O(g) \Rightarrow cf \in O(g)$ for any $c \in \mathbb{R}_{>0}$.*

   The first three rules also hold for Ω and Θ instead of O.

4. *$f \in O(g) \Leftrightarrow g \in \Omega(f)$.*
5. *$f \in \Theta(g) \Leftrightarrow g \in \Theta(f)$ (symmetry).*

The proof is left as an exercise.

The reflexivity, transitivity and symmetry properties show that Θ partitions the set of eventually positive functions into equivalence classes, where two functions are equivalent if and only if they have the same rate of growth.

Often the O is used in a more extended form, where it may appear *anywhere* on the righthand side of an equation. For instance, $f(n) = h(n) + O(g(n))$ is simply a shorthand for $f(n) = g(n) + k(n)$ and $k \in O(h)$, or more briefly $f - g \in O(h)$. Similarly, $f(n) = g(n) \cdot O(h(n)) \Leftrightarrow \frac{f}{g} \in O(h)$, $f(n) = g(n)^{O(h(n))}$ if $f(n) = g(n)^{k(n)}$ and $k \in O(h)$, and more generally $f(n) = g(n, O(h(n)))$ if $f(n) = g(n, k(n))$ and $k \in O(h)$.

Now let's see how O and some familiar operations on functions are related.

LEMMA 2.13. *Let $f, g\colon \mathbb{N} \longrightarrow \mathbb{R}$ be positive a.e.*

1. *$c \cdot O(f) = O(f)$ for any $c \in \mathbb{R}_{>0}$.*
2. *$O(f) + O(g) = O(f + g) = O(\max(f, g))$.*
3. *$O(f) \cdot O(g) = O(f \cdot g) = f \cdot O(g)$.*
4. *$O(f)^m = O(f^m)$ for any $m \in \mathbb{R}_{>0}$, where f^m denotes the function for which $f^m(n) = f(n)^m$ (and not $\underbrace{f(f(\cdots f}_{m}(n))$).*
5. *$\frac{1}{O(f)} = \Omega(\frac{1}{f})$.*
6. *$f(n) = g(n)^{O(1)} \Leftrightarrow f$ is bounded by a polynomial in g.*

Similar results hold for Ω and Θ. All but the last of the equations above are set equations, where e.g. $O(f) + O(g)$ denotes the set $\{h + k\colon h \in O(f) \text{ and } k \in O(g)\}$.

Be cautious with exponentiation of O! At first glance, you might consider $e^{O(f)} = O(e^f)$ to be valid. But the counterexample $f(n) = n$ shows that $e^{2n} = e^{O(n)}$, but not $e^{2n} = (e^n)^2 = O(e^n)$. The constant hidden within the "O" *does* influence the rate of growth when it occurs as an exponent.

For further reading on this topic, the excellent book "Concrete Mathematics" (Graham *et al.* 1989) is recommended. See also Brassard & Bratley (1988).

In some situations, the O carries still too much information. For instance, the fastest known algorithm for multiplying two n bit integers needs $O(n \log n \operatorname{loglog} n)$

bit operations (it will be discussed later in this course). So it is, *up to logarithmic factors*, essentially a *linear* algorithm like the addition algorithm for n-bit integers. To cover this, von zur Gathen invented in 1985 the so called "soft O" notation.

DEFINITION 2.14. *Let $f, g\colon \mathbb{N} \longrightarrow \mathbb{R}$ be positive a.e. Then we write $f \in O^{\sim}(g)$ (pronounce "soft Oh of ..."), or $f = O^{\sim}(g)$, or $f(n) = O^{\sim}(g(n))$) if $f(n) = O(g(n))(\log g(n))^{O(1)}$, or equivalently, if there are constants $N \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}_{>0}$ such that $f(n) \le c_1 g(n)(\log g(n))^{c_2}$ for all $n \ge N$.*

Thus $n \log n \operatorname{loglog} n = O^{\sim}(n)$, and all the ugly log-factors are swallowed by the "soft O".

2.5. Arithmetic circuits. In this subsection, we introduce the main computational model in computer algebra: arithmetic circuits. It puts algebraic algorithms on a formal basis, thus permitting their detailed cost analysis. On the other hand, algorithms are often nicely illustrated by such circuits.

In the sequel, R is some fixed ring (commutative, with 1). An *algebraic computation problem* over R is a function

$$\varphi\colon R^m \supseteq I \longrightarrow R^k$$

for some $m, k \in \mathbb{N}_{>0}$. The set I is also called the set of *instances* for φ. Examples are

- Addition of n elements of R: $m = n$, $k = 1$, $I = R^n$, and

$$\varphi\colon (x_1, \ldots, x_n) \longmapsto x_1 + \cdots + x_n.$$

- Addition of two polynomials in $R[x]$ of degree less than n: $m = 2n$, $k = n$, $I = R^{2n}$, and

$$\varphi\colon (f_0, \ldots, f_{n-1}, g_0, \ldots, g_{n-1}) \longmapsto (f_0 + g_0, \ldots, f_{n-1} + g_{n-1}).$$

- Evaluation of a polynomial in $R[x]$ of degree less than n at an element of R: $m = n + 1$, $k = 1$, $I = R^{n+1}$, and

$$\varphi\colon (f_0, \ldots, f_{n-1}, a) \longmapsto f_0 + f_1 a + \cdots + f_{n-1} a^{n-1}.$$

- Multiplication of two polynomials in $R[x]$ of degree less than n: $m = 2n$, $k = 2n - 1$, $I = R^{2n}$, and

$$\varphi\colon (f_0, \ldots, f_{n-1}, g_0, \ldots, g_{n-1}) \longmapsto (f_0 g_0, f_0 g_1 + f_1 g_0, \ldots, f_{n-1} g_{n-1}).$$

- Division of a polynomial of degree less than $2n$ by a monic polynomial of degree n: $m = 3n$, $k = 2n$, $I = R^{3n}$, and

$$\varphi\colon (f_0, \ldots, f_{2n-1}, g_0, \ldots, g_{n-1}) \longmapsto (q_0, \ldots, q_{n-1}, r_0, \ldots, r_{n-1})$$

such that

$$\sum_{0\le i<2n} f_i x^i = \left(\sum_{0\le i<n} q_i x^i\right)\left(x^n + \sum_{0\le i<n} g_i x^i\right) + \sum_{0\le i<n} r_i x^i.$$

An *arithmetic circuit* $\mathcal{C}$ over R is a labelled directed acyclic graph (DAG) with the following properties:

- Each vertex with indegree zero (*input vertex*) has nonzero outdegree and is labelled with an indeterminate or a constant from R, in such a way that no two of them carry the same label.
- Each vertex with outdegree zero (*output vertex*) has indegree one and is labelled with a positive integer in such a way that each integer in $\{1,\ldots,n\}$ occurs exactly once if there are n output vertices.
- Each other vertex (*computation vertex* or *inner vertex*) is labelled with an arithmetic operation $+,-,*$ (or $/$ if R is a field), has indegree two, and the two incoming edges are distinguishable (at least for the noncommmutative operations $-$ and $/$), e.g., by labelling them 1 and 2.

EXAMPLE 2.15. *Figure 2.1 shows two versions of an arithmetic circuit. On the left hand side, the circuit is drawn as described above. On the right hand side, the edges have no arrows and we implicitly assume that they are directed downwards. Furthermore, outgoing edges from the same vertex are combined and split at the points marked by a $\bullet$. This corresponds to a more "electrical" view of the circuit: the edges are lines, where elements of R "flow", with "contact" crossings with a $\bullet$, and no contact at the other crossings, and the vertices are "gates". The input vertices are only represented by their labels, the output vertex is not shown (in general, if there is more than one output vertex, we omit their labels and assume that they are ordered from left to right, with labels starting at 1 and increasing by 1). Finally, we implicitly assume that the left incoming edge of a $-$-vertex (or $/$-vertex) is labelled 1 and the right one 2. Mathematically, of course, both sides represent the same circuit.*

We inductively define the rational function (or polynomial, if no $/$-vertices occur) in the input variables with coefficients in R that a vertex in an arithmetic circuit computes: An input vertex computes its label, a $+$, $-$, $*$- or $/$-vertex computes the sum, difference, product, or quotient, respectively, of the two rational functions that the source vertices of the two incoming edges compute (with the obvious order of operands in the noncommutative case), and an output vertex computes the same rational function as the source vertex of its incoming edge. A condition is that no division by the rational function 0 may occur.

Figure 2.2 shows another arithmetic circuit, together with the rational functions that each vertex computes. The polynomial computed at the output vertex of the

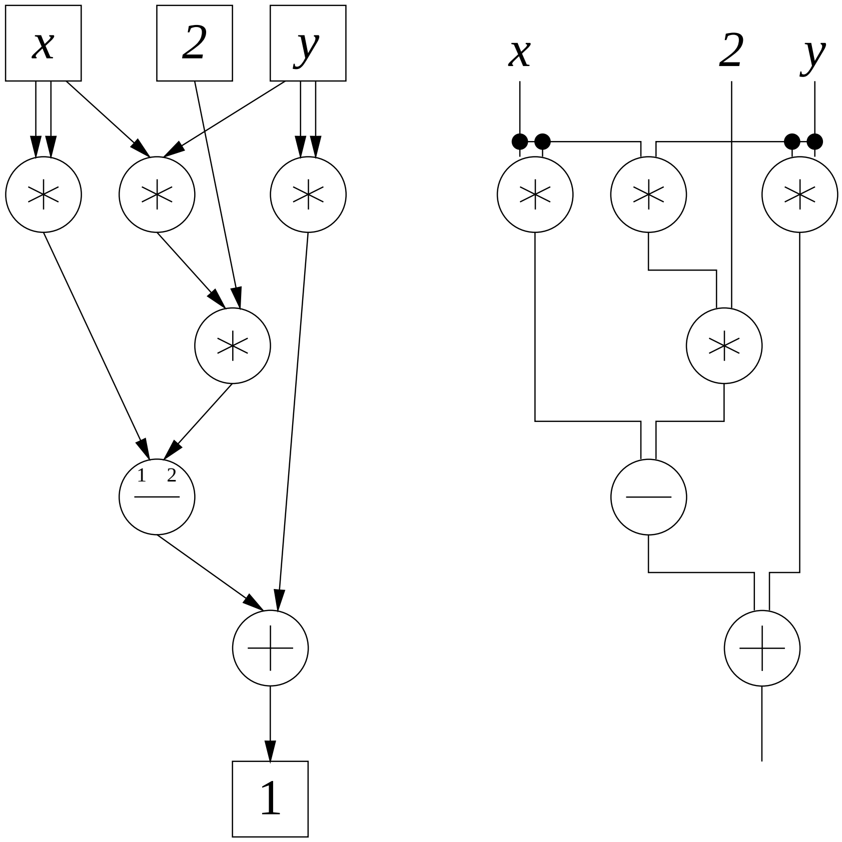


Figure 2.1: An arithmetic circuit

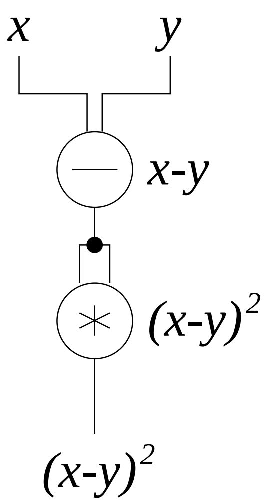


Figure 2.2: An arithmetic circuit with the polynomials computed at the individual vertices

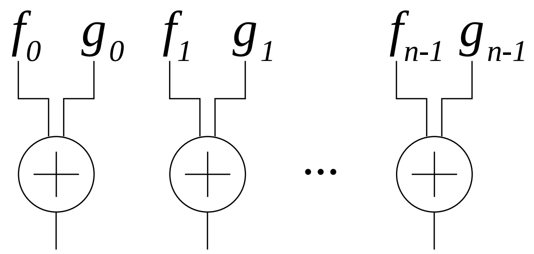
circuit in Figure 2.1 is $(x^2 - 2xy + y^2)$, which equals the polynomial at the output vertex in Figure 2.2.

Now we are ready to connect syntax and semantics: we say that an arithmetic circuit $\mathcal{C}$ *solves* an algebraic computation problem $\varphi\colon R^m \supseteq D \longrightarrow R^k$ if

- $\mathcal{C}$ has m input vertices labelled with variables, $x_1, \ldots, x_m$ say,
- $\mathcal{C}$ has k output vertices, where the output vertex labelled j computes a rational function $u_j \in R(x_1, \ldots, x_m)$ for $1 \leq j \leq k$, and
- for all instances $(a_1, \ldots, a_m) \in I$, $u_j(a_1, \ldots, a_m)$ is defined for $1 \leq j \leq k$ and

$$\varphi(a_1, \ldots, a_m) = (u_1(a_1, \ldots, a_m), \ldots, u_k(a_1, \ldots, a_m)).$$

EXAMPLE 2.16. *An arithmetic circuit for the addition of two polynomials in $R[x]$ of degree less than n:*



The two most important cost measures for arithmetic circuits are the *size* , i.e., the number of *internal* vertices of a circuit, and the *depth* , which is defined as the maximum number of internal vertices on paths from input vertices to output vertices. The size of a circuit corresponds to the sequential running time of an algorithm implementing the circuit on one processor: one might, e.g., arrange the vertices in a fixed order which respects the topology of the underlying DAG, and then the actual running time would be proportional to the number of computation vertices. The depth corresponds to parallel running time: if there are as much processors as vertices running in parallel and some their connections are as in the circuit, the running time is proportional to the length of the longest path in the circuit.

Both cost measures are a significant abstraction in two respects. First, additive operations $+, -$ cost the same as multiplicative operations $*, /$, while in many important applications multiplications and divisions are much more expensive than additions. The second abstraction refers to the actual size of the ring elements: the cost for an addition of two integers, e.g., depends on the number of digits, but the unit cost measure arithmetic operations on *arbitrary* operands ignores these differences.

EXAMPLE 2.17. *Figure 2.3 shows two arithmetic circuits for the addition of n ring elements. The first one mirrors the way how one would add the n elements by hand*

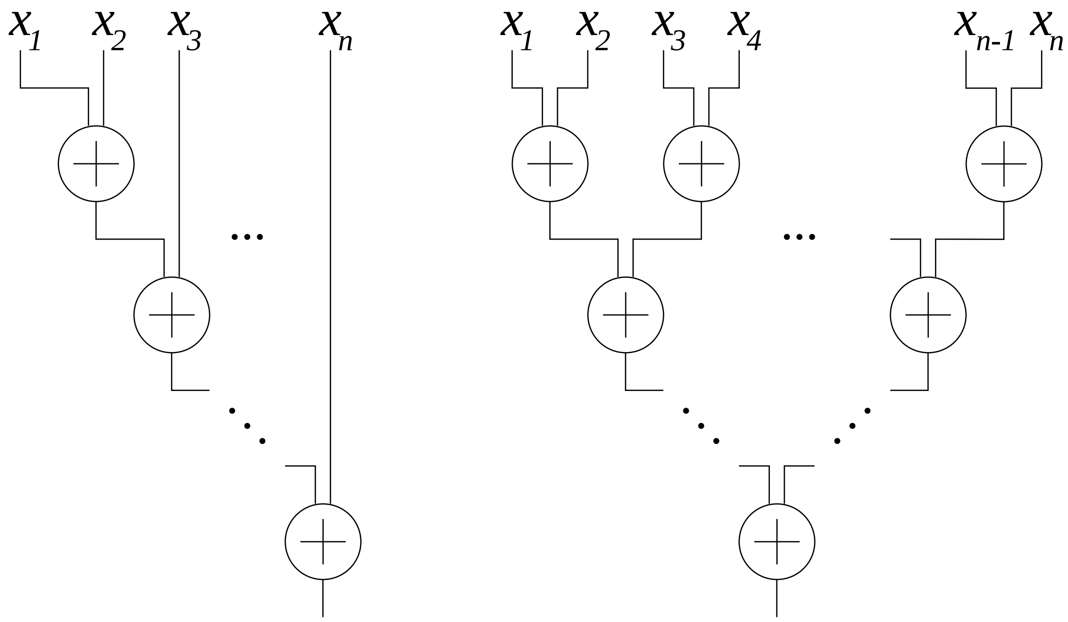


Figure 2.3: Two arithmetic circuits for addition of n elements

by summing them up one after the other. It has size and depth equal to $n-1$. *The second one arranges the additions in form of a binary tree and has the same size, but depth only* $\lceil \log_2 n \rceil$.

EXAMPLE 2.18. *Figure 2.4 shows an arithmetic circuit for the division of a polynomial of degree less than* $2n$ *by a monic polynomial of degree* n *with remainder in the case* $n = 4$, *using the "school method". The first* 8 *input vertices correspond to the coefficients of the dividend, the last* 4 *input vertices to the nonleading coefficients of the divisor (since the divisor is monic, its leading coefficient need not be specified). The first* 4 *of the output vertices yield the coefficients of the quotient and the last* 4 *output vertices the coefficients of the remainder.*

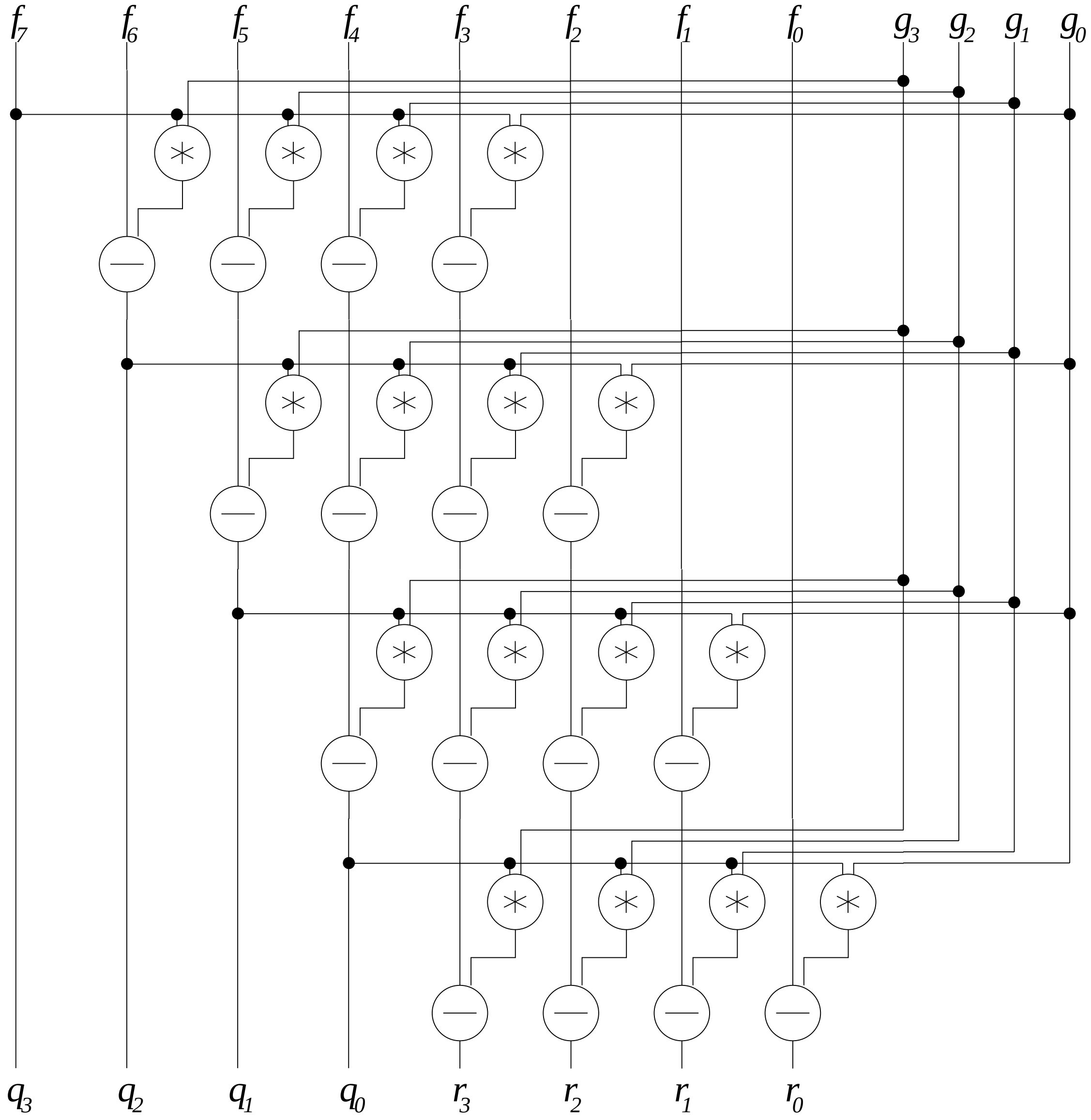


Figure 2.4: An arithmetic circuit for polynomial division

3. A few easy algorithms

3.1. Evaluation of polynomials with respect to different cost measures.

PROBLEM 3.1. *Given $n \in \mathbb{N}$, find an algorithm that, on input $a_0, \cdots, a_n$ and x, computes $y = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$. This is the problem of polynomial evaluation. The inputs and the output are elements of some ring R.*

The obvious algorithm to evaluate the polynomial is to compute $x^2, x^3, \ldots, x^n$, requiring $n-1$ multiplications, and then computing the terms $a_i x^i$, requiring a further n multiplications. Finally, adding the terms together requires n additions.

We can improve the number of multiplications by a factor of two if we change the order in which we do our adding and multiplying. Consider rewriting the polynomial as

$$y = (\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots)x + a_0.$$

Computing the innermost bracket, $a_n x + a_{n-1}$ requires one multiplication and one addition. This process is repeated n times to get y. The cost of the algorithm using this algorithm known as *Horner's rule* is n multiplications and n additions as opposed to $2n-1$ multiplications and n additions for the above "naïve" method.

This algorithm comes from Horner (1819), but Newton (1969), p. 222, already gives an example of the method which *maximè expeditum putat*.

The birthday of *algebraic complexity theory* was in 1954 when Ostrowski asked whether or not Horner's rule was optimal. Besides considering total arithmetic operations, Ostrowski also introduced a different notion of cost. He posed the following problem: Suppose that F is a field, and $R = F[x, a_0, \ldots, a_n]$ the ring of polynomials in indeterminates $x, a_0, \ldots, a_n$. Can you then improve the number of multiplications in polynomial evaluation by allowing, at no cost, multiplication of some fixed elements of the field F, called *scalars* , with any other element of R, and also addition of any pair of elements of R? In general, the scalars can be any fixed set of elements. Over $\mathbb{R}$, the algorithm might consider such numbers as 2, $\sqrt{2}$ or π as scalars.

Pan in 1966 showed the answer to be no: Horner's rule is optimal! He showed that at least n multiplications are needed to evaluate general polynomials of degree n. Belaga (1958) had shown a corresponding result for additions. However, if the coefficients $a_0, \ldots, a_n$ of the polynomial are not indeterminates, but fixed field elements (scalars), then the following holds.

THEOREM 3.2. *(Paterson & Stockmeyer 1973) Let $a \in F[x]$, F any field, have degree n. Then $a(x)$ can be evaluated with $2\lceil\sqrt{n}\,\rceil + 1$ non-scalar multiplications.*

PROOF. Begin by partitioning a into roughly $\sqrt{n}$ blocks of length $\sqrt{n}$. Let $m = \lceil\sqrt{n}\,\rceil$ and $k = \lceil n/m\rceil + 1$. Then

$$\begin{aligned}\sum_{0\le i\le n} a_i x^i &= (a_{km-1}x^{m-1} + \cdots + a_{(k-1)m})x^{(k-1)m} + \cdots \\ &\quad + (a_{2m-1}x^{m-1} + \cdots + a_m)x^m + (a_{m-1}x^{m-1} + \cdots + a_0).\end{aligned}$$

(The a_i with $i > n$ are set to zero.)

Evaluation then can be done by first computing $x, x^2, \cdots, x^m$ in $m-1$ non-scalar operations. We can compute $y_i = \sum_{0 \le j \le m} a_{im+j} x^j$, for $0 \le i \le k$, at no cost because we only multiply by scalars and add. Horner's rule, applied to evaluating $\sum_{0 \le i \le k} y_i x^{im}$, requires a further k non-scalar multiplications and some additions which we don't count. The total cost is $m + k \le 2\lceil \sqrt{n} \rceil + 1$. □

In practice, this result is useful only in very special cases, e.g., when trying to evaluate polynomials at matrices in which the coefficients may be integers and the indeterminate x may vary over $m \times m$ matrices. It makes sense when evaluating such a polynomial to reduce, as much as possible, the number of matrix multiplications used. If we regard the integers as scalars then the above result implies that the polynomial can be evaluated using about $2\sqrt{n}$ matrix multiplications. The "naïve" algorithm above and Horner's rule give algorithms that use $2n-1$ and n matrix multiplications, respectively.

3.2. Multiplication of polynomials.

PROBLEM 3.3. *Input: the coefficients of two polynomials a and b.*
Output: the coefficients of the product $c = ab$.

The obvious method for two polynomials of degree at most n uses $O(n^2)$ operations: $(n+1)^2$ multiplications $a_i b_j$, and n^2 additions for all $c_k = \sum_{i+j=k} a_i b_j$. For instance, multiplying $(ax+b)(cx+d) = acx^2 + (ad+bc)x + bd$ uses four multiplications ac, ad, bc, bd, and one addition $ad + bc$. Note that we are not calculating the value of ab at x (as in 3.1), but are only computing the coefficient sequence.

Surprisingly, there is an easy method of doing better. We assume that the degrees of a and b are less than $n = 2^k$, for some $k \in \mathbb{N}$. Set $m = n/2$ and rewrite a and b in the form $a = A_1 x^m + A_0$ where $A_1 = a_{n-1} x^{m-1} + \cdots + a_m$ and $A_0 = a_{m-1} x^{m-1} + \cdots + a_0$, and similarly $b = B_1 x^m + B_0$. (If $\deg a < n-1$, then some of the top coefficients are zero.) Now $ab = A_1 B_1 x^n + (A_0 B_1 + A_1 B_0) x^m + A_0 B_0$. In this form, multiplication of a and b has been reduced to 4 multiplications of polynomials of degree less than m and four additions of polynomials with degree less than $2n$. Note that multiplication by a power of x does not count as a multiplication, since it corresponds merely to a shift of the coefficients.

So far we have not really achieved anything. But this new expression for ab can be rearranged to reduce the number of multiplications of the smaller polynomials at the expense of increasing the number of additions. Since multiplication is slower than addition, a saving is obtained when n is sufficiently large. The idea is due to Karatsuba & Ofman (1962). Rewrite the product as $ab = A_1 B_1 (x^n - x^m) + (A_1 + A_0)(B_1 + B_0) x^m + A_0 B_0 (1 - x^m)$. With this expression, we see that multiplication of a and b requires only three multiplications of polynomials of degree less than m and six additions of polynomials of degree at most $2n$. If $T(n)$ denotes the time necessary to multiply two polynomials of degree less than n, then $T(2n) \le 3T(n) + cn$, for some constant c. The linear term comes from the observation that addition of two polynomials of degree less than ℓ can be done with ℓ operations.

THEOREM 3.4. *If* $T(1) = 1$ *and* $T(2^k) \le 3T(2^{k-1}) + c2^k$ *if* $k > 0$, *then* $T(2^k) \le (1+2c)3^k - 2c \cdot 2^k$ *for* $k \ge 0$.

PROOF. By induction on k. The case $k = 0$ is clear. For $k > 0$, we have

$$T(2^k) \le 3T(2^{k-1}) + c \cdot 2^k \le 3 \cdot ((1+2c)3^{k-1} - 2c \cdot 2^{k-1}) + c2^k = (1+2c)3^k - 2c \cdot 2^k. \quad \square$$

COROLLARY 3.5. *The Karatsuba-Ofman algorithm for multiplying polynomials over a ring can be done with* $O(n^{\log_2 3})$ *or* $O(n^{1.59})$ *ring operations.*

PROOF. The above discussion and the theorem imply that the algorithm requires $O(3^{\log_2 n})$ steps. The result follows from noting that $3^{\log_2 n} = 2^{\log_2 3 \cdot \log_2 n} = n^{\log_2 3}$, and $\log_2 3 < 1.59$. $\square$

This is a substantial improvement over the classical method, since $\log_2 3 < 2$. This is illustrated in Figure 3.2.

(Question: Work out the smallest value of c. Check that the classical method requires $2(n+1)^2 - (2n+1)$ operations. For which values of $n = 2^k$ is $(1+2c)3^k - 2c2^k$ smaller than the classical $2(n+1)^2 - (2n+1)$?)

The Karatsuba-Ofman algorithm is used in computer algebra systems like Maple. In practice, the classical method is faster for polynomials of small degree . The challenge in an implementation is to determine the *break even point*, where the asymptotically faster algorithm actually beats the slower one.

3.3. Multiplication of integers.

PROBLEM 3.6. *Given two integers* a *and* b, *determine their product* $c = ab$.

If we represent integers in terms of powers of a base B, then $a = \sum_i a_i B^i$and $b = \sum_i b_i B^i$, where $0 \le a_i, b_i < B$. (The binary notation has $B = 2$.) There is a noticeable similarity between this representation of integers and the usual representation of polynomials. We might expect that the algorithms that work for polynomial multiplication work for integer multiplication except for a few details involving carries. In fact this is the case.

The classical algorithm for integer multiplication learned in school requires $O(n^2)$ bit operations when a and b have at most n digits ($B = 2$ and $a_i = B_i = 0$ for $i \ge n$). Karatsuba's algorithm works in much the same way for integers as for polynomials. Write $a = A_1 2^n + A_0$ and $b = B_1 2^n + B_0$ where the binary lengths of a and b are less than $2n$. The product ab can be rewritten as $(2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0$. As in the polynomial case, multiplication of two integers has been reduced to multiplication of three integers of at most half the size plus a few $O(n)$ operations. We conclude, as for polynomials, that multiplication of two n bit integers requires $O(n^{\log_2 3})$ digit operations.

In existing computer algebra systems, asymptotically fast algorithms may not be used or may be used only for polynomials and integers that are very large. The issue of representation plays a crucial role in determining at what point the "fast" methods beat the "classical" algorithms.

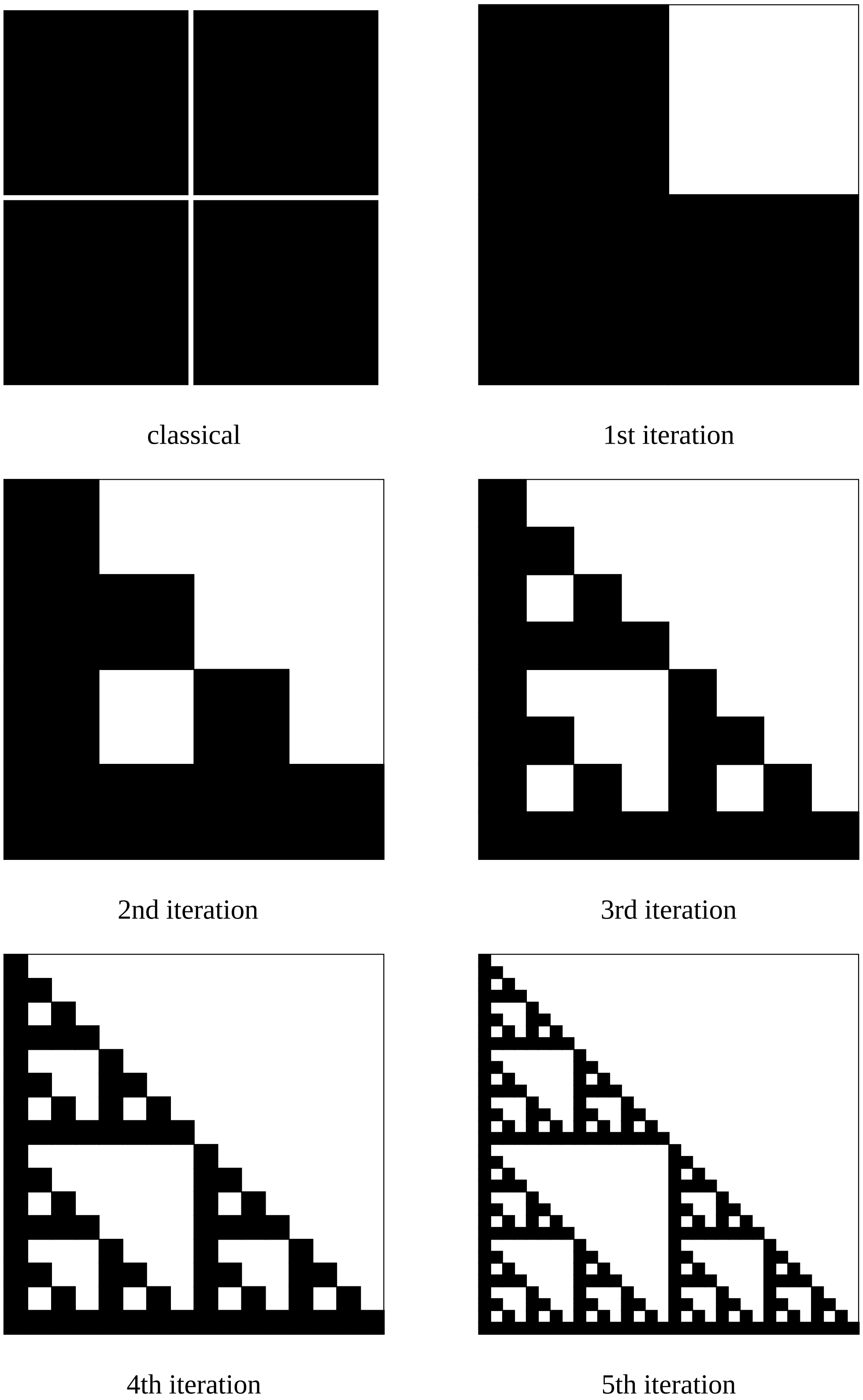


Figure 3.5: Cost of the Karatsuba-Ofman algorithm for variuos recursion depths. The image approaches a fractal of dimension $\log_2 3$.

Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik
Skript Computeralgebra I, Teil 2
Wintersemester 1995/96

4. Signals, filters, and convolution

Definition 4.1. *A* continuous signal *(or* analog signal*) is a function*

$$\sigma : D \to \mathbb{R}^n, \text{ where } D \subseteq \mathbb{R}^m \text{ and } m, n \in \mathbb{N}.$$

Sound is a signal that varies over time and has range amplitude and frequency. It is an example of a signal $\sigma : \mathbb{R} \to \mathbb{R}^2$. In the case of black and white images on a screen, the signal associates to each point an intensity value. Here $\sigma : D \subseteq \mathbb{R}^2 \to \mathbb{R}$. When color is represented by the constituent amounts of three basic colors (RGB) or four basic colors (CMYK), we have a signal mapping into $\mathbb{R}^3$ or $\mathbb{R}^4$, respectively.

Of particular importance are the sine signal $\sigma : [0,1] \to \mathbb{R}$ with $\sigma(t) = \sin(2\pi t)$, and its complex variant $\sigma : [0,1] \to \mathbb{R}^2 \cong \mathbb{C}$ with $\sigma(t) = e^{2\pi i t} = \cos(2\pi t) + i \sin(2\pi t)$, where $i = \sqrt{-1}$. They will play a central role in the Fourier analysis in section 5. More generally, we have the signal $\sigma : [0,1] \to \mathbb{C}$ with $\sigma(t) = e^{i(\omega t + \rho)}$, with *frequency* ω and *phase shift* ρ.

Definition 4.2. *A* discrete signal *is a function*

$$x : D \to \mathbb{R}^n, \text{ where } D \subseteq \mathbb{Z}^m \text{ and } m, n \in \mathbb{N}.$$

A discrete signal is often obtained by sampling a continuous signal at discrete intervals. For example, when we sample the complex exponential signal with frequency ω at n regularly spaced points in $[0,1)$, we obtain $x : \{0, 1, \cdots, n-1\} \to \mathbb{C}$ with $x(j) = e^{i\omega' j}$, where $\omega' = \omega/n$.

Discrete signals find applications in such areas as biomedical engineering, acoustics, sonar and radar imaging, seismology, speech communication, data communication, television satellite communication, satellite images and many more. Speech and telephone signals are examples of signals with only one dimensional domains, while radar imaging, satellite images, lunar images are processed with two dimensional domains. When modelling complicated problems, such as appear in seismology, the domain can have many dimensions.

It is important to perform certain operations on signals to extract relevant information from them, or to transform the signal to make it easier to use. For instance one may wish to extract some important parameters from the data such as danger signs from an electrocardiogram or electroencephalogram. One may want to compress the data contained in a telephone signal or recognize the words associated with speech signals. A common problem is to extract relevant information from masses of data associated with such things as television transmission or satellite images. Television satellites transmit about 85 Mbit/sec while LANDSAT satellites transmit about 100 Mbits/sec; this amounts to 4×10^{15} bits per year. Another application of signal processing in signal transmission is to try to remove signal interference contributed by transmission noise, fading or channel distortion.

For this course we will restrict attention to one dimensional domains. That is, discrete signals will be functions $x\colon \mathbb{Z} \to \mathbb{R}$ or $x\colon \mathbb{Z} \to \mathbb{C} \cong \mathbb{R}^2$. Signals form a vector space: they can be added together and multiplied by real scalars . Thus if x and y are two signals and $a, b \in \mathbb{R}$, we have the signal $z = ax + by$. There is a set of special signals that form a basis for the set of all signals. The *unit sample signal at 0* is defined to be

$$\delta(x) = \begin{cases} 1 & \text{if } x = 0; \\ 0 & \text{otherwise.} \end{cases}$$

Trivially, every signal satisfies the following relation: $x(n) = \sum_{k\in\mathbb{Z}} x(k)\delta(n-k)$, for all $n \in \mathbb{Z}$. Thus if we define the signals δ_k by $\delta_k(n) = \delta(n-k)$ for all $n \in \mathbb{Z}$, we can formally write $x = \sum_{k\in\mathbb{Z}} x(k)\delta_k$, where the $x(k) \in \mathbb{R}$ are coefficients. The unit sample signals δ_k form a basis for the space of discrete signals. (Caution: often we have a fixed finite subset $D \subseteq \mathbb{Z}$, say $D = \{0, \cdots, m\}$, and then this is the usual notion of a basis of a vector space over $\mathbb{R}$. However, in general infinitely many nonzero $x(k)$'s may be needed, and then this is a slightly different notion. We gloss over all questions of convergence in the integrals and infinite sums in this and the next section.)

Definition 4.3. *A filter is a function from signals to signals (a "functional"). A linear filter is a linear function on signals. Thus T is a linear filter if for all signals x, y and $a, b \in \mathbb{R}$, $T(ax + by) = aT(x) + bT(y)$.*

Definition 4.4. *The unit sample response of a linear filter T is its effect on the unit samples δ_k.*

A linear filter is characterized by its unit sample response. Suppose the unit sample response is $h(k, n) = T(\delta_k)(n)$ for $n, k \in \mathbb{Z}$. Then $T(x)(n) = \sum_{k\in\mathbb{Z}} x(k)h(k, n)$.

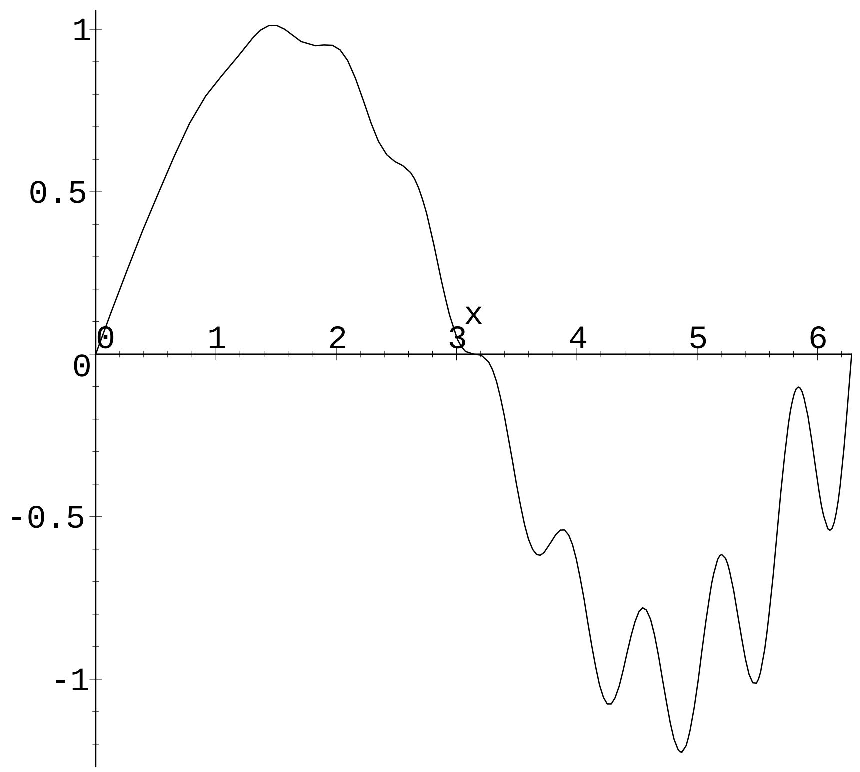
Definition 4.5. *Given a signal x and $k \in \mathbb{Z}$, the associated k-shifted signal, denoted by x_k, is the signal defined by $x_k(n) = x(n-k)$, for $n \in \mathbb{Z}$.*

For any $k \in \mathbb{Z}$, the filter T_k that takes a signal to its k-shifted signal is a linear filter. This follows from observing that

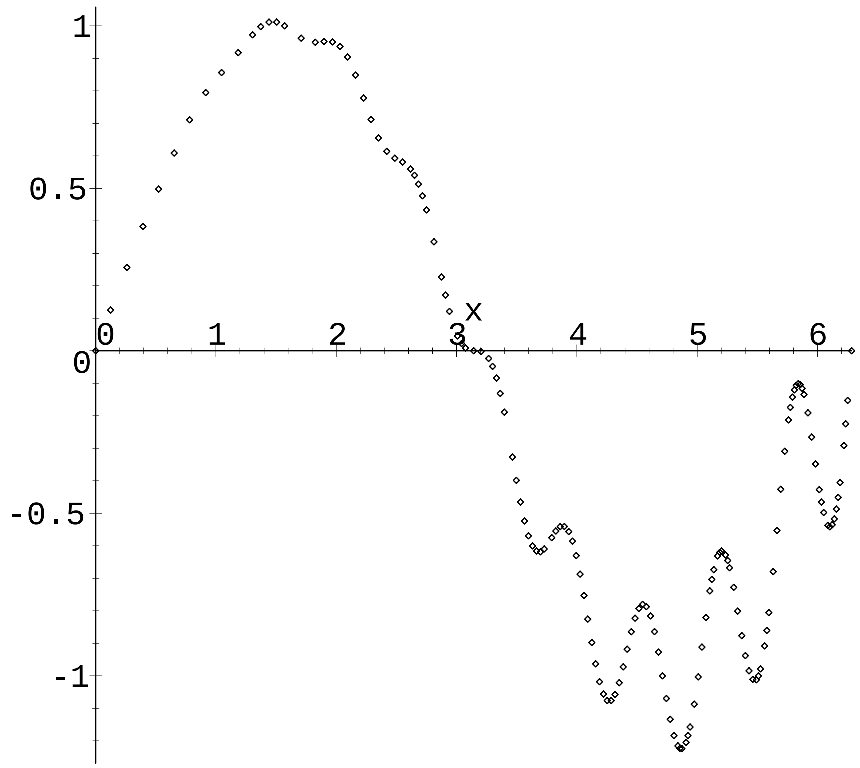
$$\begin{aligned} T_k(ax + by)(n) &= (ax + by)_k(n) = (ax + by)(n-k) = ax(n-k) + by(n-k) \\ &= aT_k(x)(n) + bT_k(y)(n) \\ &= (aT_k(x) + bT_k(y))(n), \ \forall n \in \mathbb{Z}, \ a, b \in \mathbb{R}. \end{aligned}$$

We have $T_k = \underbrace{T_1 \circ \cdots \circ T_1}_{k \text{ times}}$.

Figure 4.6: An analogue signal, and the corresponding discrete signal.



$$f(x) = \sin(x) + \frac{1}{100}x^2\sin(10x)$$



Definition 4.6. *A linear filter T is shift invariant if for all signals x, y and $k \in \mathbb{Z}$ with $y = T(x)$ we have $y_k = T(x_k)$. In words, T acting on the k-shift of a signal is the same as the k-shift of T acting on the signal. Equivalently, $T \circ T_k = T_k \circ T$ for all $k \in \mathbb{Z}$, or $T \circ T_1 = T_1 \circ T$.*

Definition 4.7. *The convolution of two signals x,y is the signal*

$$x * y = \sum_{k \in \mathbb{Z}} x(k) y_k, \text{ with } (x * y)(n) = \sum_{k \in \mathbb{Z}} x(k) y(n-k) \text{ for } n \in \mathbb{Z}.$$

Again, we are ignoring questions of convergence here. In many examples, e.g. when $x(n) = y(n) = 0$ for $n < 0$, the above sums are well defined.

Example 4.8. *For every signal x, we have*

$$x = \sum_{k \in \mathbb{Z}} x(k) \delta_k = x * \delta,$$

so that δ is the unit element w.r.t. $*$.

Remark 4.9. *The convolution $*$ is associative, commutative, and bilinear, so that the vector space of all discrete signals together with $*$ as multiplication is an $\mathbb{R}$-algebra.*

If we consider the unit sample response $h(n, k)$ of a linear shift invariant filter T and define the signal $h(n) = h(0, n)$, then we have $h_k(n) = h(k, n)$. It is clear then that the signal h characterizes T. Further we can write $T(x)(n) = \sum_{k \in \mathbb{Z}} x(k) h_k(n) = (x * h)(n)$. Thus for linear shift invariant filters, $T(x) = x * h$. If T is not shift invariant then the relation $h_k(n) = h(k, n)$ does not necessarily hold.

Example 4.10. *Let $0 < a < 1$ and $N \in \mathbb{N}$. Consider the linear shift invariant filter T defined by*

$$h(n) = \begin{cases} a^n & n \geq 0, \\ 0 & \text{otherwise}, \end{cases}$$

and the signal

$$x = \sum_{k \geq 0} \delta_k.$$

The response is $y = T(x) = x * h$, or equivalently

$$y(n) = \sum_{k \in \mathbb{Z}} x(k) h(n-k) = \sum_{0 \leq k \leq n} h(n-k) = \begin{cases} 0 & n < 0; \\ \frac{1-a^{n+1}}{1-a} & n \geq 0. \end{cases}$$

Definition 4.11. *Let x be a discrete signal. The* Z-transform *of x is the formal series*

$$\widehat{x} = \sum_{n \in \mathbb{Z}} x(n) z^n.$$

LEMMA 4.12. *For two signals x, y we have $\widehat{x * y} = \widehat{x} \cdot \widehat{y}$.*

This expresses the fact that multiplication of two (formal) powerseries is just the convolution of their coefficient sequences.

EXAMPLE 4.13. *(Example 4.10 continued)*

$$\begin{aligned}
\widehat{x} &= \sum_{n\in\mathbb{Z}} x(n)z^n = \sum_{n\geq 0} z^n = \frac{1}{1-z}, \\
\widehat{h} &= \sum_{n\in\mathbb{Z}} h(n)z^n = \sum_{n\geq 0} a^n z^n = \frac{1}{1-az}, \\
\widehat{x * h} &= \sum_{n\geq 0} \frac{1-a^{n+1}}{1-a} z^n = \frac{1}{1-a}\sum_{n\geq 0} z^n - \frac{a}{1-a}\sum_{n\geq 0} a^n z^n \\
&= \frac{1}{(1-a)(1-z)} - \frac{a}{(1-a)(1-az)} = \frac{(1-az)-a(1-z)}{(1-a)(1-z)(1-az)} \\
&= \frac{1}{(1-z)(1-az)} = \widehat{x} \cdot \widehat{h}.
\end{aligned}$$

As a further example, we now discuss a *low-pass filter*. Suppose that some apparatus has measured a discrete signal, but that the measurement x is distorted by random noise:

$$\text{measured signal } x = \text{smooth signal} + \text{random noise}.$$

We can smooth x by replacing $x(j)$ by the average of the values at j and $j+1$:

$$y(j) = \frac{x(j) + x(j+1)}{2}.$$

This rule defines a linear filter T, with $T(x) = y$. Note that $T = (T_{-1} + T_0)/2$. (Exercise: check linearity.) Let us apply T to the discrete sinusoidal signal x with frequency ω, so that $x(j) = e^{i\omega j}$:

$$\begin{aligned}
y(j) &= T(x)(j) = \frac{x(j) + x(j+1)}{2} \\
&= \frac{e^{i\omega j} + e^{i\omega(j+1)}}{2} = e^{i\omega j} \cdot \frac{1}{2}(1 + e^{i\omega}).
\end{aligned}$$

It is remarkable that the second factor does not depend on j.

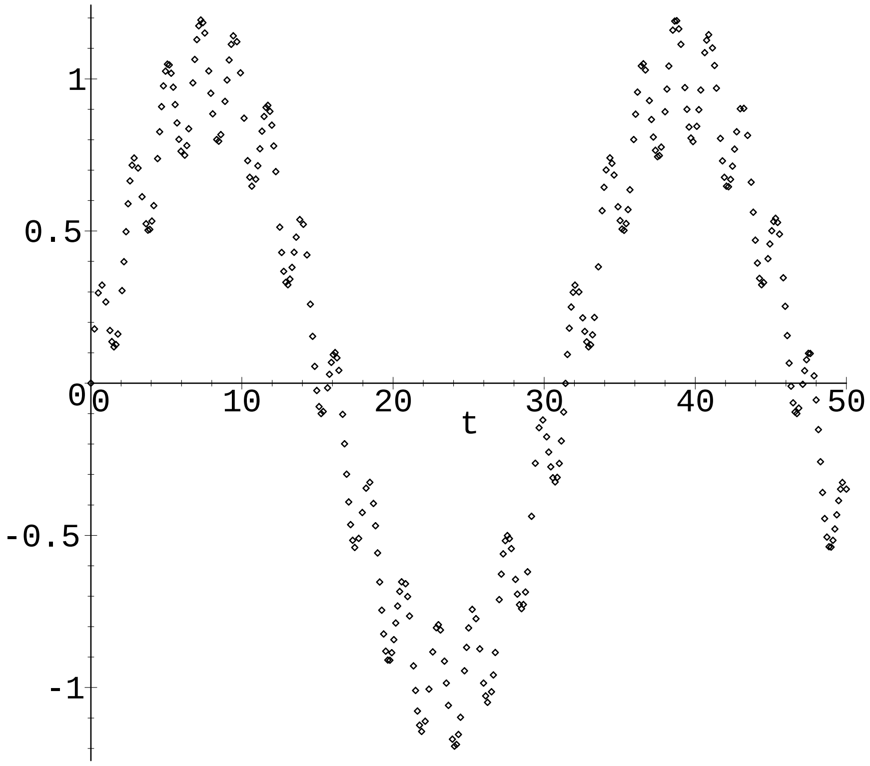
Let us define the complex number

$$\eta = \frac{1}{2}(1 + e^{i\omega}) = |\eta| \cdot e^{i\cdot arg(\eta)},$$

where the right hand expression gives the polar coordinates of η. Then

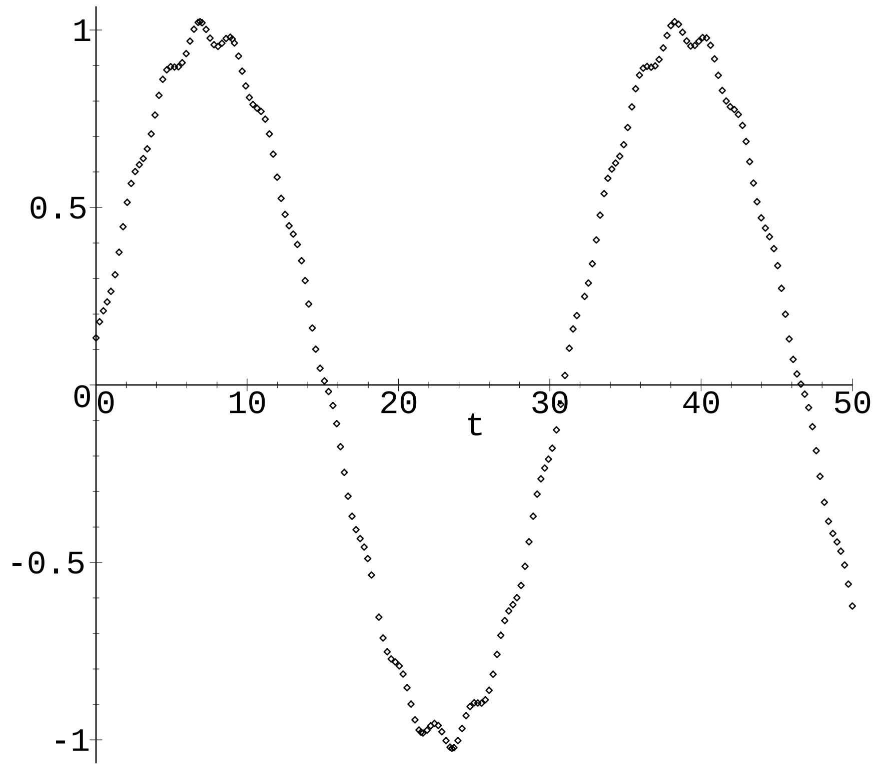
$$\begin{aligned}
\forall j \quad T(x)(j) &= \eta \cdot x(j), \\
T(x) &= \eta \cdot x = |\eta| \cdot e^{i(\omega j + arg\ \eta)}.
\end{aligned}$$

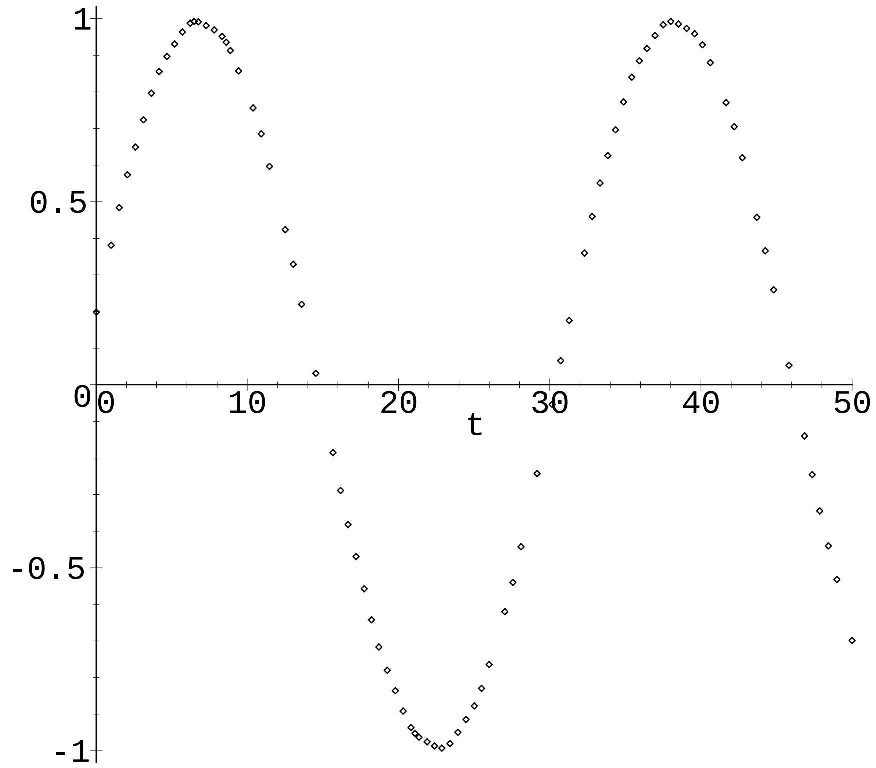
Figure 4.7: A discrete signal with noise



$$f(t) = \sin(0.2t) + 0.2\sin(2.8t)$$

Figure 4.8: The signal from figure 4.7 after one and two applications of the low-pass filter T





So the effect of applying T to a sinusoidal signal x is to multiply the amplitude by $|\eta|$ and to shift the phase by $arg\ \eta$.

$$\begin{aligned} |\eta| &= \frac{1}{2} \mid 1+e^{i\omega} \mid = \frac{1}{2} \mid e^{i\omega/2} \mid \cdot \mid e^{-i\omega/2}+e^{i\omega/2} \mid \\ &= \frac{1}{2} \cdot 1 \cdot 2\ \cos(\omega/2) = \cos(\omega/2). \end{aligned}$$

We have the following approximate equivalences:

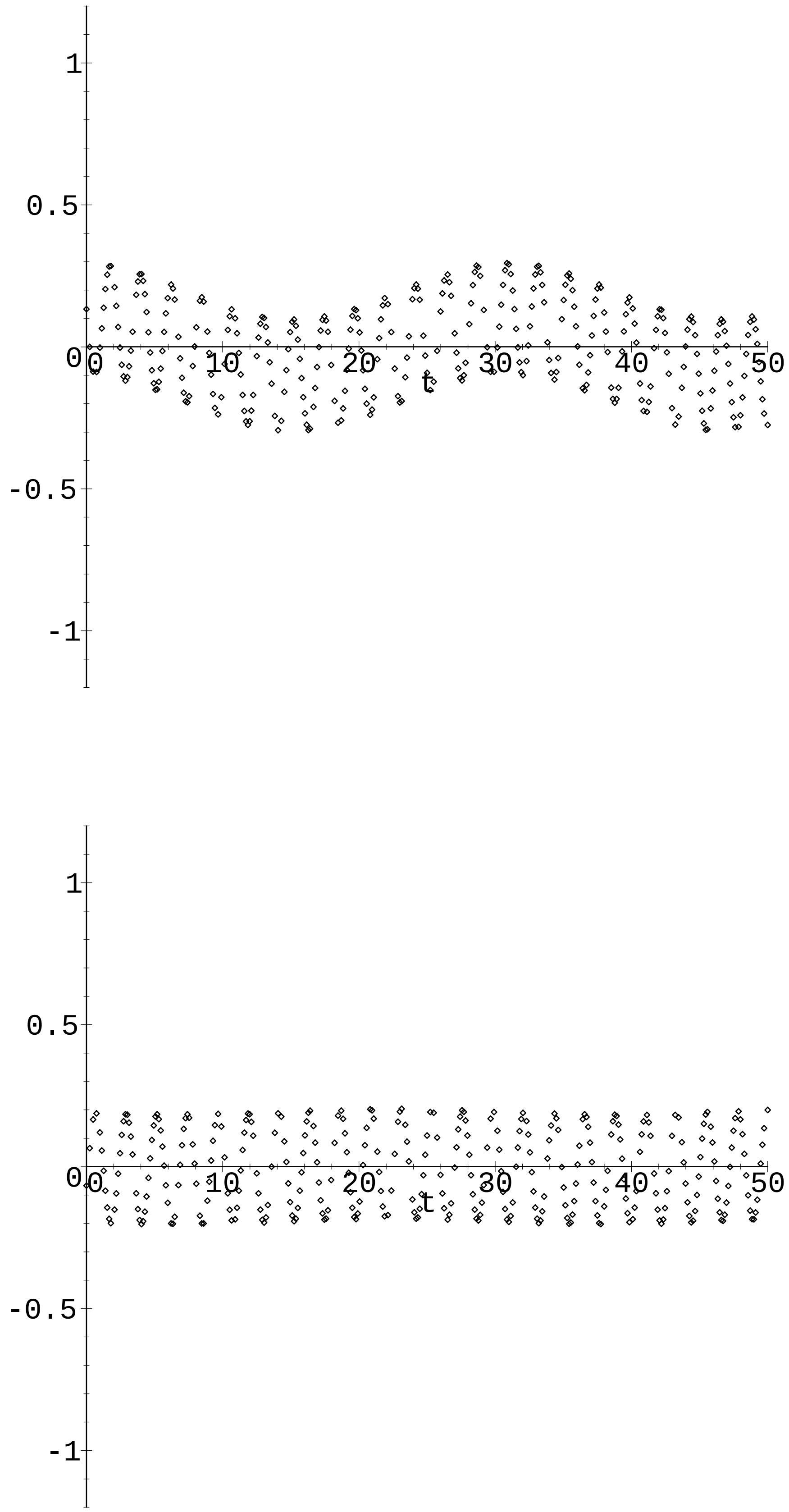
signal remains intact $\longleftrightarrow \cos(\omega/2)$ is close to 1
$\longleftrightarrow \omega/2$ is close to 0 $\longleftrightarrow \omega$ is close to 0
$\longleftrightarrow$ frequency is small;

signal is almost killed $\longleftrightarrow \cos(\omega/2)$ is close to 0
$\longleftrightarrow \omega/2$ is close to $\pi/2 \longleftrightarrow \omega$ is close to π
$\longleftrightarrow$ frequency is large.

Hence the name *low-pass filter*. If our measurement x consists of a low-frequency signal x_0 plus high-frequency noise x_1, the filtered signal $T(x)$ will reflect x_0 rather well, while the effect of the noise x_1 will be greatly reduced.

Analogously, one finds that the *high-pass filter* $U = (T_{-1} - T_0)/2$ reduces the influence of the low frequencies. The effect of the low-pass filter T and the high pass filter U on the discrete signal from figure 4.7 is shown in figure 4.8 and figure 4.9, respectively.

Figure 4.9: The signal from figure 4.7 after one and two applications of the high-pass filter U



5. The Fourier transform

5.1. The continuous and the discrete Fourier transform. In the previous section, we discussed discrete signals. In this section, we will deal with periodic contiuous signals, i.e., functions $f : \mathbb{R} \longrightarrow \mathbb{C}$ for which $a \in \mathbb{R}_{>0}$ exists such that $f(t+a) = f(t)$ for all $t \in \mathbb{R}$. Applying the transformation $t \longmapsto \frac{2\pi t}{a}$ if necessary, we may assume that $a = 2\pi$.

If f, g are two 2π-periodic signals that are "smooth", we define (in analogy to the discrete case) the *convolution* $f * g \colon \mathbb{R} \longrightarrow \mathbb{C}$ of f and g by

$$(f * g)(s) = \int_0^{2\pi} f(t)g(s-t)dt \text{ for } s \in \mathbb{R}.$$

("Smooth" implies that the integral exists.) As in the discrete case, the set of all "smooth" 2π-periodic signals together with $*$ as multiplication is a $\mathbb{C}$-algebra.

DEFINITION 5.1. *The* Fourier transform *of a 2π-periodic signal f is $\widehat{f} \colon \mathbb{Z} \longrightarrow \mathbb{C}$ with*

$$\widehat{f}(k) = \int_0^{2\pi} f(x)e^{-ikt}dt \textit{ for } k \in \mathbb{Z},$$

where $i = \sqrt{-1} \in \mathbb{C}$ as usual.

The Fourier transform of a continuous signal is the analogue of the Z-transform of a discrete signal, and a similar formula for the transform of the concolution holds:

$$\widehat{f * g} = \widehat{f} \cdot \widehat{g}, \tag{5.1}$$

i.e.,

$$(\widehat{f * g})(k) = \widehat{f}(k) \cdot \widehat{g}(k) \text{ for all } k \in \mathbb{Z}.$$

Its proof is left as an exercise.

Furthermore, the following inversion formula holds.

$$f(t) = \frac{1}{2\pi} \sum_{k \in \mathbb{Z}} \widehat{f}(k) e^{ikt}$$

for all $t \in \mathbb{R}$, the series converges uniformly to f. This series is called the *Fourier series* for f, and the numbers $\beta_k = \frac{1}{2\pi}\widehat{f}(k) = \frac{1}{2\pi}\int_0^{2\pi} f(t)e^{-ikt}dt$ for $k \in \mathbb{Z}$ are the *Fourier coefficients* of f. The inversion formula says that the function f is uniquely determined by the sequence of its Fourier coefficients $(\beta_k)_{k\in\mathbb{Z}}$. The special functions e^{ikt}, for $k \in \mathbb{Z}$, are a "basis" for the complex vector space of all 2π-periodic functions; however, there are in general infinitely many nonzero coefficients.

Popular examples of 2π-periodic functions are obtained by taking a function defined on a bounded interval, normalizing the interval to be $[0, 2\pi]$, and extending the function by periodicity. A typical example is the following, where we actually leave out the periodic extension.

EXAMPLE 5.2. *Consider the square wave*

$$f(t) = \begin{cases} +1, & 0 < t < \pi; \\ -1, & \pi < t < 2\pi; \\ 0, & t = 0, \pi, 2\pi. \end{cases}$$

Now, $\beta_0 = 0$ *and for* $k \neq 0$*, the* k*th Fourier coefficient is*

$$\begin{aligned}
\beta_k &= \frac{1}{2\pi}\int_0^{2\pi} f(t)e^{-ikt}dt \\
&= \frac{1}{2\pi}[\int_0^{\pi} e^{-ikt}dt - \int_{\pi}^{2\pi} e^{-ikt}dt] \\
&= \frac{1}{2\pi}\frac{i}{k}[e^{-ikt}|_0^{\pi} - e^{-ikt}|_{\pi}^{2\pi}] \\
&= \frac{i}{2\pi k}[e^{-ik\pi} - e^0 - e^{-ik\cdot 2\pi} + e^{-ik\pi}] \\
&= \frac{i}{\pi k}(e^{-ik\pi} - 1) \\
&= \begin{cases} \frac{-2i}{k\pi}, & k \text{ odd;} \\ 0, & k \text{ even.} \end{cases}
\end{aligned}$$

So

$$f(t) = \sum_{\substack{k \in \mathbb{Z} \\ k \text{ odd}}} \frac{-2i}{k\pi} e^{ikt}.$$

Now,

$$\begin{aligned}
f(t) &= \sum_{\substack{k>0 \\ k \text{ odd}}} (\beta_k e^{ikt} + \beta_{-k}e^{-ikt}) + \beta_0 \\
&= \sum_{\substack{k>0 \\ k \text{ odd}}} \frac{-2i}{k\pi}(e^{ikt} - e^{-ikt}) + 0 \\
&= \sum_{\substack{k>0 \\ k \text{ odd}}} \frac{-2i}{k\pi}((\cos(kt) + i\sin(kt)) - (\cos(kt) - i\sin(kt))) \\
&= \sum_{\substack{k>0 \\ k \text{ odd}}} \frac{4}{k\pi}\sin(kt).
\end{aligned}$$

So,

$$f(t) = \frac{4}{\pi}(\sin(t) + 1/3\sin(3t) + 1/5\sin(5t) + \cdots).$$

From $f(\pi/2) = 1$ *we may deduce that*

$$(1 - 1/3 + 1/5 - 1/7 + \cdots) = \pi/4.$$

This result can be verified by considering

$$\begin{aligned}\tan^{-1}(x) &= \int_0^x \frac{dt}{1+t^2} \\ &= \int_0^x (1 - t^2 + t^4 - t^6 + \cdots)dt \\ &= 1 - x^3/3 + x^5/5 - x^7/7 + \cdots.\end{aligned}$$

Since $\tan^{-1}(1) = \pi/4$,

$$1 - 1/3 + 1/5 - 1/7 + \cdots = \pi/4,$$

as before.

The Discrete Fourier Transform can be obtained from the (continuous) Fourier series by approximating the integral by step functions with steps of size $2\pi/n$ on the interval $[0, 2\pi]$.

$$\beta_k \sim b_k = \frac{1}{2\pi} \sum_{0 \le j < n} f\left(\frac{2\pi j}{n}\right) \left(e^{-\frac{2\pi i}{n}}\right)^{jk} \cdot \frac{2\pi}{n} = \frac{1}{n} \sum_{0 \le j < n}^{n-1} a_j \omega^{jk},$$

where $a_j = f(2\pi j/n)$ and the argument $\omega = e^{-2\pi i/n} \in \mathbb{C}$ is an nth primitive root of unity.

5.2. Formal DFT theory.

DEFINITION 5.3. *Let R be a ring, $n \in \mathbb{N}$ and $\omega \in R$. ω is a primitive nth root of unity (n-PRU) if*

1. $\omega^n = 1$,
2. *n is a unit in R,*
3. *$\omega^k - 1$ is not zero or a zero divisor for $1 \le k < n$.*

The last condition implies, among other things that $\omega^k \neq 1$, $1 \le k < n$.

EXAMPLE 5.4. *Suppose $R = \mathbb{C}$ and $\omega = e^{2\pi i/8}$. Then ω is an 8-PRU.*

EXAMPLE 5.5. *$\mathbb{Z}_8$ has no primitive square root of unity, despite the fact that $3^2 \equiv 1$.*

EXAMPLE 5.6. *Consider the "Fermat prime" $m = 2^4 + 1$. 3 is a 16-PRU.*

EXERCISE 5.7. *If ω is an n-PRU, then show that ω^{-1} is also. If n is even, then show that ω^2 is an $\frac{n}{2}$-PRU. If n is odd, then show that ω^2 is an n-PRU.*

Let R be a ring, $n \in \mathbb{N}$, and $\omega \in R$ an n-PRU. In the sequel, we identify polynomials $a = \sum_{0 \le i < n} a_i x^i \in R[x]$ of degree less than n with their coefficient vectors $(a_0, \ldots, a_{n-1}) \in R^n$.

Definition 5.8. *The R-linear map*

$$\mathrm{DFT}_\omega \colon \begin{cases} R^n & \longrightarrow \quad R^n \\ \quad a & \longmapsto \quad \big(a(1), a(\omega), a(\omega^2), \ldots, a(\omega^{n-1})\big) \end{cases}$$

which evaluates a polynomial at the powers of ω is called the Discrete Fourier Transform *(DFT).*

Definition 5.9. *The* convolution *of two polynomials $a = \sum_{0 \le i < n} a_i x^i$ and $b = \sum_{0 \le j < n} b_j x^j$ in $R[x]$ is the polynomial*

$$c = a *_n b = \sum_{0 \le k < n} c_k x^k \in R[x],$$

where

$$c_k = \sum_{i+j \equiv k \bmod n} a_i b_j = \sum_{0 \le i < n} a_i b_{k-i} \text{ for } 0 \le k < n,$$

with index arithmetic modulo n. If n is clear from the context, we will simply write $*$ for $*_n$. If we regard the coefficients as vectors in R^n, then c is called the cyclic convolution of vectors a and b.

This notion of convolution is equivalent to polynomial multiplication in the ring $R[x]/(x^n - 1)$. The kth coefficient of the product polynomial $a \cdot b$ is $\sum_{i+j=k} a_i b_j$, and hence $a *_n b \equiv ab \bmod (x^n - 1)$. This relationship between convolution and multiplication can be exploited to get fast polynomial multiplication algorithms.

Lemma 5.10. *For polynomials $a, b \in R[x]$ of degree less than n, we have*

$$\mathrm{DFT}_\omega(a *_n b) = \mathrm{DFT}_\omega(a) \cdot \mathrm{DFT}_\omega(b),$$

where $\cdot$ denotes the pointwise multiplication of vectors.

Proof. We have $a *_n b = ab + g \cdot (x^n - 1)$ for some $g \in R[x]$, so that

$$(a *_n b)(\omega^i) = a(\omega^i) b(\omega^i) + g(\omega^i)(\omega^{in} - 1) = a(\omega^i) b(\omega^i)$$

for $0 \le i < n$. $\Box$

If we write $\widehat{a}$ for $\mathrm{DFT}_\omega(a)$, then Lemma 5.10 becomes $\widehat{a * b} = \widehat{a} \cdot \widehat{b}$, as in (5.1). On a higher level, this says that $\mathrm{DFT}_\omega \colon R[x]/(x^n - 1) \longrightarrow R^n$ is an R-algebra homomorphism, where multiplication in R^n is pointwise multiplication of vectors. In fact, DFT_ω is an isomorphism. This a special case of a more general theorem, the Chinese Remainder Theorem, that we will discuss later in this course.

A general principle in algorithm design is the *change of representation*. For many mathematical objects, there exist several possibilities to describe and represent them, e.g., as data type in a computer program. For example, graphs may be represented by adjacency matrices or linked edge lists. Each representation often has its advantages and disadvantages, so that one kind of object manipulation is "easy" in one but "complicated" in another representation. This is where a change of representation comes in.

There are at least three possibilities to represent univariate polynomials. The familiar "dense" representation is by coefficient lists, where each coefficient is mentioned regardless whether it is zero or not. For random polynomials, where in general very few coefficients are actually zero, this is a sensible way. In practice, however, many polynomials are of high degree but have only very few nonzero coefficients, e.g., binomials $x^n + a$ or trinomials $x^n + ax + b$. For such so-called "sparse" polynomials, storing every coefficient would be a waste of space. The better way of representing them is by lists of pairs (i, a_i), where only those indices i with $a_i \neq 0$ occur. A third way of representing polynomials is functions for their evaluation. This is called the "black box" representation. It is easy to convert between dense and sparse representation and from each of them into the black box representation, and interpolation gives a way to construct the coefficients of a polynomial from its black box representation. Each of the above representations can analogously also be used for matrices.

In the following, we will use a fourth representation for polynomials, the *value representation*. Given a polynomial $f = \sum_{0 \leq i < n} f_i x^i \in R[x]$ of degree less than n over an integral domain R in its dense representation and n pairwise distinct elements $u_0, \ldots, u_{n-1} \in R$, we may represent f by its values $f(u_0), \ldots, f(u_{n-1})$ at the n points. The coefficients of f may be recovered from the n values by *interpolation*, and f is the only polynomial of degree less than n having those values at the n points (Exercise: Prove uniqueness). The conversion from the dense to the value representation of a polynomial is called *multipoint evaluation*, and interpolation is the conversion in the reverse direction.

The reason for considering the value representation of a polynomial is that multiplication in that representation is easy: If $f(u_0), \ldots, f(u_{2n-1})$ and $g(u_0), \ldots, g(u_{2n-1})$ are the values of two polynomials f and g of degree less than n at $2n$ distinct points, then the values of the product polynomial fg at those points are given by $f(u_0)g(u_0), \ldots, f(u_{2n-1})g(u_{2n-1})$ (we have to choose $2n$ evaluation points in order to have a unique representation of the product polynomial). Hence polynomial multiplication in the value representation is linear in the degree, while we do not know how to multiply polynomials in the dense representation in linear time. Thus a fast way of doing multipoint evaluation and interpolation would lead to a fast polynomial multiplication algorithm: evaluate the two input polynomials, multiply the results pointwise, and finally interpolate to get the product polynomial.

The Discrete Fourier Transform is a special multipoint evaluation at the powers $1, \omega, \ldots, \omega^{n-1}$ of a primitive nth root of unity ω, and we will now show that both the DFT and its inverse, the interpolation at the powers of ω, can be computed with $O(n \log n)$ operations in R and thus yield an $O(n \log n)$ multiplication algorithm for polynomials.

First we will show that interpolation at the powers of ω is essentially again a Discrete Fourier Transform. Recall the definition of the *Vandermonde matrix*. For

$u_0, \ldots, u_{n-1} \in R$, we have

$$\mathrm{VDM}(u_0, \ldots, u_{n-1}) = \begin{pmatrix} 1 & u_0 & u_0^2 & \cdots & u_0^{n-1} \\ 1 & u_1 & u_1^2 & \cdots & u_1^{n-1} \\ 1 & u_2 & u_2^2 & \cdots & u_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u_{n-1} & u_{n-1}^2 & \cdots & u_{n-1}^{n-1} \end{pmatrix} \in R^{n \times n}.$$

This matrix is the matrix of the multipoint evaluation map

$$\begin{array}{ccc} R^n & \longrightarrow & R^n \\ (f_0, \ldots, f_{n-1}) & \longmapsto & \left(\sum_{0 \le i < n} f_i u_0^i, \sum_{0 \le i < n} f_i u_1^i, \ldots, \sum_{0 \le i < n} f_i u_{n-1}^i \right) \end{array}$$

which takes polynomials of degree less than n (represented by their coefficient vectors) to their values at $u_0, \ldots, u_{n-1}$. If $u_0, \ldots, u_{n-1}$ are pairwise distinct, then the matrix $\mathrm{VDM}(u_0, \ldots, u_{n-1})$ is invertible, and its inverse is the matrix of the interpolation map at the n points.

The Vandermonde matrix of DFT_ω is

$$V_\omega = \mathrm{VDM}(1, \omega, \ldots, \omega^{n-1}) = (\omega^{ij})_{0 \le i,j < n}.$$

Example 5.11. *For the 4-PRU $\omega = i \in \mathbb{C}$, we have*

$$V_i = \mathrm{VDM}(1, i, -1, -i) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

We will now determine V_ω^{-1}.

Lemma 5.12. *Let $n \in \mathbb{N}$, R be a ring, and $\omega \in R$ with $\omega^n = 1$ and $1 \le k < n$. Assume that $n \in R$ is not zero or a zero divisor in R. Then $\sum_{0 \le j < n} \omega^{jk} = 0$ if and only if $\omega^k - 1$ is not zero or a zero divisor.*

Proof. From the formula

$$(x-1) \sum_{0 \le j < n} x^j = x^n - 1$$

we see that

$$(\omega^k - 1) \sum_{0 \le j < n} (\omega^k)^j = \omega^{kn} - 1 = 0.$$

If $\omega^k - 1$ is not zero or a zero divisor, then $\sum_j \omega^{jk} = 0$. Otherwise, there exists a nonzero $a \in R$ such that $a(\omega^k - 1) = 0$. This implies that $a\omega^k = a$. We observe then that $a\omega^{kj} = (a\omega^k)\omega^{k(j-1)} = a\omega^{k(j-1)} = \cdots = a$. Then the sum equals na which is not zero since n is a unit and a is not zero. □

THEOREM 5.13. *Let ω be an nth PRU. Then $V_\omega \cdot V_{\omega^{-1}} = nI$, where I is the $n \times n$ identity matrix.*

PROOF. Let

$$u = (V_\omega \cdot V_{\omega^{-1}})_{ik} = \sum_{0 \le j < n} (V_\omega)_{ij} (V_{\omega^{-1}})_{jk} = \sum_{0 \le j < n} \omega^{ij} \omega^{-jk} = \sum_{0 \le j < n} (\omega^{i-k})^j.$$

If $i = k$, then $u = \sum_j 1 = n$. If $i \neq k$ then

$$u = \sum_{0 \le j < n} \omega^{(i-k)j} = \frac{\omega^{(i-k)n} - 1}{\omega^{i-k} - 1} = 0,$$

by the previous lemma. □

This result is particularly useful to compute the inverse of the matrix V_ω since

$$(V_\omega)^{-1} = n^{-1} V_{\omega^{-1}}.$$

This means that computing the inverse is fairly easy.

EXAMPLE 5.14. *(Example 5.11 continued)*

$$V_i^{-1} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

Below we discuss an important algorithm—the Fast Fourier Transform—that computes the DFT quickly. The relation between the DFT and the Vandermonde matrix shows that the inverse DFT can then also be computed quickly.

As an illustration consider the problem of evaluating a polynomial $f = \sum_k x^k \in R[x]$ at the points at $+1, -1$. Separate the polynomial into its even and odd powered terms. Then $f = a + bx$ where $a = \sum_k f_{2k} x^{2k}$ and $b = \sum_k f_{2k+1} x^{2k}$. Now $f(1) = a(1) + b(1)$ and $f(-1) = a(1) - b(1)$. The process of evaluating f at $+1, -1$ has been reduced to the problem of evaluating two polynomials of half the degree of f at 1. If we could continue this recursively with the square root of -1 etc., we can evaluate the polynomial quickly for more distinct points. This process was (re)discovered by Cooley and Tukey in 1965. The method became known as the *Fast Fourier Transform*, or *FFT* for short, and is probably the second most important nontrivial algorithm in practice. (The most important one is fast sorting.)

THEOREM 5.15. *Let n be a power of 2 and $\omega \in R$ be an nth PRU. Then DFT_ω can be computed using $O(n \log n)$ ring operations in R.*

PROOF. We give an algorithm for computing DFT_ω.

ALGORITHM 5.16. *Fast Fourier Transform (FFT)*
Input: $n = 2^k \in \mathbb{N}_{>0}$ *with* $k \in \mathbb{N}$, $f = \sum_{0\le i<n} f_i x^i \in R[x]$, *and the powers* $\omega, \omega^2, \ldots, \omega^{n-1}$ *of an n-PRU* $\omega \in R$.
Output: $\mathrm{DFT}_\omega(f) = (f(1), f(\omega), \ldots, f(\omega^{n-1})) \in R^n$.

1. *If* $n = 1$ *then return* (f_0).

2. *Write* $f = a(x^2) + x \cdot b(x^2)$ *with* $a, b \in R[x]$ *of degree less than* $\frac{n}{2}$.

3. *Recursively compute*

$$\begin{aligned} (\alpha_i)_{0\le i<n/2} &= \mathrm{FFT}(\frac{n}{2}, a, \omega^2, \omega^4, \ldots, \omega^n), \\ (\beta_i)_{0\le i<n/2} &= \mathrm{FFT}(\frac{n}{2}, b, \omega^2, \omega^4, \ldots, \omega^n). \end{aligned}$$

4. *For* $0 \le i < \frac{n}{2}$ *compute*

$$\begin{aligned} \gamma_i &= \alpha_i + \omega^i \beta_i, \\ \gamma_{i+n/2} &= \alpha_i - \omega^i \beta_i. \end{aligned}$$

5. *Return* $(\gamma_0, \ldots, \gamma_{n-1})$.

As in the example above, the algorithm splits the polynomial f into polynomials $a = f_{even}, b = f_{odd}$ whose coefficients are the odd numbered and even numbered coefficients of f, respectively:

$$f_{even} = \sum_{0\le j<n/2} f_{2j} x^j \text{ and } f_{odd} = \sum_{0\le j<n/2} f_{2j+1} x^j.$$

Then the two polynomials a, b of degree less than $\frac{n}{2}$ are recursively evaluated at all the powers of the $\frac{n}{2}$-PRU ω^2, and finally the results are combined.

To prove correctness, we have to show that $\gamma_i = f(\omega^i)$ for $0 \le i < n$. This is clear for $i < \frac{n}{2}$ from the above, and for $i \ge \frac{n}{2}$ we have

$$\begin{aligned} \gamma_i &= \alpha_{i-n/2} - \omega^{i-n/2}\beta_{i-n/2} = a(\omega^{2(i+n/2)}) - \omega^{i+n/2} b(\omega^{2(i+n/2)}) \\ &= a(\omega^{2i}\omega^n) + \omega^{n/2}\omega^{i+n/2} b(\omega^{2i}\omega^n) = a(\omega^{2i}) + \omega^i b(\omega^{2i}) \\ &= f(\omega^i). \end{aligned}$$

Note that $\omega^{n/2} = -1$ since

$$0 = \omega^n - 1 = (\omega^{n/2} - 1)(\omega^{n/2} + 1)$$

and $\omega^{n/2} - 1$ is neither 0 nor a zero divisor.

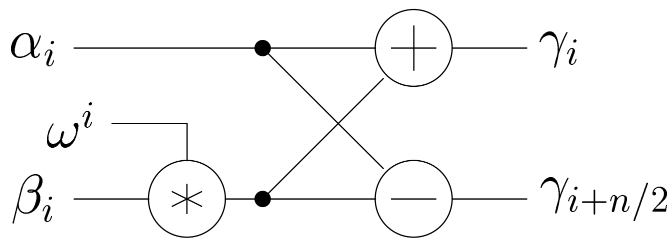
Let $T(n)$ denote the number of ring operations in R the algorithm uses on input n. The cost for the individual steps is: 0 in steps 1 and 2, $2T(\frac{n}{2})$ for step 3, and

$\frac{n}{2}$ multiplications and n additions in step 4. This yields $T(1) = 0$ and $T(n) = 2T(\frac{n}{2}) + \frac{3}{2}n$, and by unfolding the recursion we may deduce that

$$T(n) = \frac{3}{2} n \log_2 n \in O(n \log n). \quad \square$$

The FFT may be very well presented in the computational model of arithmetic circuits. The circuit is built from elementary blocks that execute step 4 of the above algorithm for one particular value of i, called a *butterfly operation*. One such building block is shown in figure 5.1, and the entire circuit for $n = 8$ in figure 5.2.

Figure 5.1: A butterfly operation (from: Cormen, Leiserson & Rivest 1989)



Now we use the FFT to compute convolutions and products of polynomials fast.

DEFINITION 5.17. *We say that a ring R supports the FFT if*

1. $\mathrm{char} R \neq 2$,
2. *R has a 2^lth PRU for any $l \in \mathbb{N}$,*

An example for a ring that supports the FFT is $R = \mathbb{C}$.

THEOREM 5.18. *Let R be a ring that supports the FFT, and $n = 2^k$ for some $k \in \mathbb{N}$. Then convolution in $R[x]/(x^n - 1)$ and multiplication of polynomials $a, b \in R[x]$ with $\deg(ab) < n$ can be performed using $\frac{9}{2} n \log_2 n + 3n \in O(n \log n)$ ring operations.*

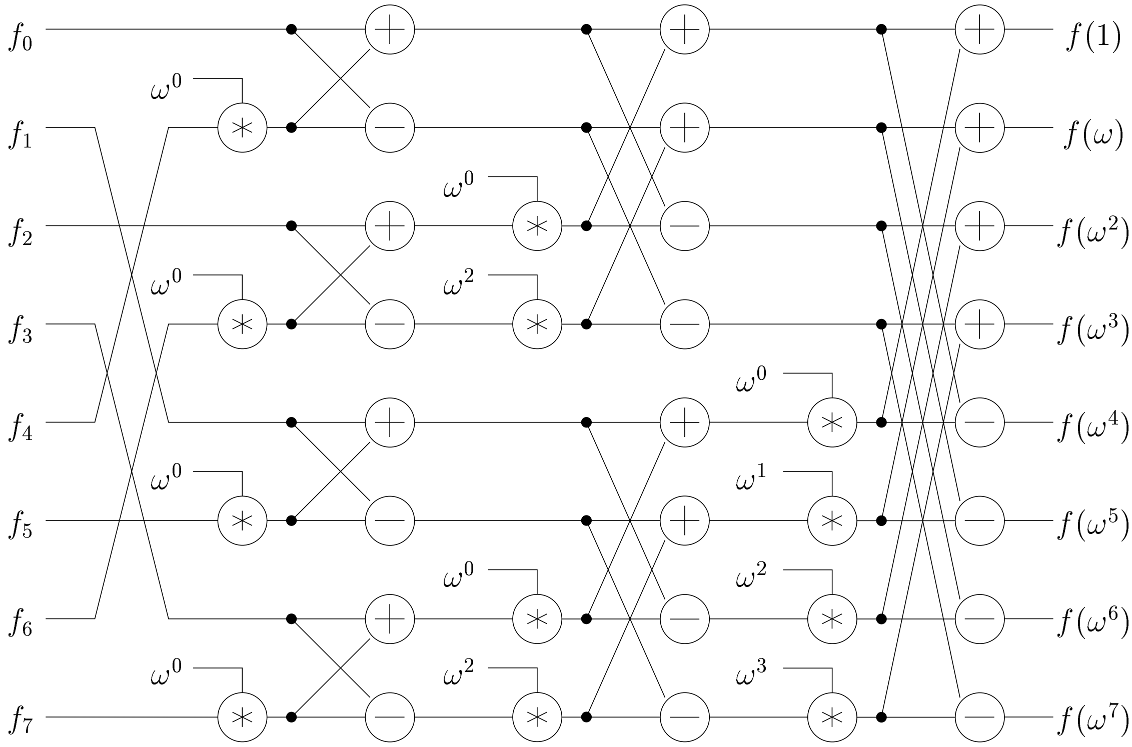
PROOF. Let a, b be vectors of polynomial coefficients in R^n, and let $c = a * b$. First note that c is uniquely determined by its value at n distinct points. Since the convolution $c = a * b$ satisfies

$$\mathrm{DFT}_\omega(c) = \mathrm{DFT}_\omega(a) \cdot \mathrm{DFT}_\omega(b),$$

by Lemma 5.10, where the multiplication is the componentwise product, we see that the following steps compute the convolution.

ALGORITHM 5.19. *Fast convolution and polynomial multiplication*
Input: *$a, b \in R[x]$ of degree less than $n = 2^k$ with $k \in \mathbb{N}$, and an n-PRU $\omega \in R$.*
Output: $a * b \in R[x]$.

Figure 5.2: An arithmetic circuit computing the FFT for $n = 8$ (from: Cormen, Leiserson & Rivest 1989)



1. *Compute* $\omega^2, \ldots, \omega^{n-1}$.
2. *Compute* $\alpha = \mathrm{DFT}_\omega(a) \in R^n$ *and* $\beta = \mathrm{DFT}_\omega(b) \in R^n$.
3. *Compute* $\gamma = \alpha \cdot \beta \in R^n$.
4. *Return* ${\mathrm{DFT}_\omega}^{-1}(\gamma) = \frac{1}{n}\mathrm{DFT}_{\omega^{-1}}(\gamma)$.

It is clear that the algorithm correctly computes the convolution of a and b. In particular, the output does not depend on the choice of ω. If furthermore $\deg(ab) < n$, then $a * b \equiv ab \bmod (x^n - 1)$ implies $ab = a * b$.

The cost for the individual steps is

1. at most n,
2. $2 \cdot \frac{3}{2} n \log_2 n$,
3. n,
4. $\frac{3}{2} n \log n + n$,

and the claim follows. $\square$

To multiply two arbitrary polynomials of degree less than m, we only need a 2^kth PRU, where $2^{k-1} < 2m \leq 2^k$. Then we have decreased the cost $\Omega(m^2)$ of the "naive" algorithm to $O(m \log m)$.

COROLLARY 5.20. *If R supports the FFT, then polynomials in $R[x]$ of degree at most n can be multiplied with $O(n \log n)$ field operations.*

5.3. Other results. Again the close relation between integers and polynomials leads us to ask whether these results can be extended to integer multiplication. In fact, the following result introduced the FFT into computer algebra.

THEOREM 5.21. *(Schönhage & Strassen 1971) Integer multiplication can be performed using $O(n \log n \operatorname{loglog} n)$ bit operations.*

The method uses the FFT. The extra factor $\operatorname{loglog} n$ is caused by the fact that the PRUs are not available in the ring of integers, but "virtual PRUs" have to be "constructed" within the algorithm. Schönhage and Strassen also applied their method to polynomial multiplication in time $O(n \log n \operatorname{loglog} n)$, without our assumption that "F supports the FFT". Schönhage (1977) solved the additional complication that occurs in characteristic two. These two papers, and also Cantor & Kaltofen (1991), showed:

THEOREM 5.22. *Over any ring R, polynomials of degree at most n can be multiplied using $O(n \log n \operatorname{loglog} n)$ operations.*

Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik
Skript Computeralgebra I, Teil 3
Wintersemester 1995/96

6. Fast polynomial evaluation and interpolation at subspaces of finite fields

In this course, we will discuss algorithms for many problems in computer algebra based on fast polynomial multiplication. In order to abstract from the underlying multiplication algorithm in our cost analyses, we introduce the following notation.

Definition 6.1. *Let R be a ring. A function* $\mathsf{M}\colon \mathbb{N}_{>0} \longrightarrow \mathbb{R}_{>0}$ *is called a* multiplication time *for $R[x]$ if polynomials in $R[x]$ of degree less than n can be multiplied using $O(\mathsf{M}(n))$ operations in R.*

Algorithm	$\mathsf{M}(n)$
classical	n^2
Karatsuba & Ofman (1962)	$n^{1.59}$
FFT multiplication (provided that R supports the FFT)	$n \log n$
Schönhage & Strassen (1971), Schönhage (1977) (FFT based)	$n \log n \operatorname{loglog} n$

Figure 6.3: Various polynomial multiplication algorithms and their running times

In principle, any polynomial multiplication algorithm leads to a multiplication time. Figure 6.3 summarizes the multiplication times for the algorithms discussed in the previous sections. In the rest of this course, we will assume that the multiplication time satisfies $\mathsf{M}(mn) \geq m \cdot \mathsf{M}(n)$ for all $m, n \in \mathbb{N}_{>0}$, and will mainly use $\mathsf{M}(n) = n \log n \operatorname{loglog} n$.

For the finite field $\mathbb{F}_2$, an algorihm by Schönhage (1977), using a ternary (base 3) FFT, works in time $O(n \log n \operatorname{loglog} n)$ but appears to be tedious to implement on a binary computer. However, with Schönhage's (1994) implementation of a Turing machine on modern processors (SPARC), this can be used for a highly efficient multiplication routine (see Reischert 1995). In this section, we will present a different algorithm for polynomial multiplication over a finite field $\mathbb{F}_q$ with q elements with a running time of $O(n \log_q^2 n)$ operations in $\mathbb{F}_q$ when $2n \leq q$. This will eventually lead to an $O(n \log^3 n)$ algorithm for polynomials over $\mathbb{F}_2$, which is slightly worse than Schönhage's result but also yields efficient multiplication procedures. Like the

FFT, the algorithm is based on fast multipoint evaluation and interpolation, but now at *additive* subsets (linear subspaces) rather than multiplicative subsets (roots of unity).

6.1. Multipoint evaluation in the general case. The Discrete Fourier Transform evaluates a polynomial at the powers of a primitive root of unity. We will now discuss an algorithm for solving the general multipoint evaluation problem.

We consider the following situation: Let F be a field, $n \in \mathbb{N}$, $u_0, \ldots, u_{n-1} \in F$ pairwise distinct, $m_i = x - u_i \in F[x]$, and $m = \prod_{0 \le i < n}(x - u_i)$. Then the map

$$\begin{array}{rcl} \chi\colon F[x] & \longrightarrow & F^n \\ f & \longmapsto & (f(u_0), \ldots, f(u_{n-1})) \end{array}$$

is a surjective ring homomorphism. $F[x]$ and F^n are also vector spaces over F, thus F-algebras, and χ is also a homomorphism of F-algebras.

We want to solve the first of the following problems. For simplicity, we assume that the number of points is a power of two.

PROBLEM 6.2. *(Multipoint evaluation) Given $n = 2^k$ for some $k \in \mathbb{N}$, $f \in F[x]$ of degree $< n$, and $u_0, \ldots, u_{n-1} \in F$ pairwise distinct, compute*

$$\chi(f) = (f(u_0), \ldots, f(u_{n-1})).$$

PROBLEM 6.3. *(Interpolation) Given $n = 2^k$ for some $k \in \mathbb{N}$, $u_0, \ldots, u_{n-1} \in F$ pairwise distinct, and $b_0, \ldots, b_{n-1} \in F$, compute $f \in F[x]$ of degree $< n$ with*

$$\chi(f) = (f(u_0), \ldots, f(u_{n-1})) = (b_0, \ldots, b_{n-1}).$$

In section 5, we saw that both problems can be solved with $O(n \log n)$ operations in F if F supports the FFT and $u_i = \omega^i$, where ω is a primitive nth root of unity. For arbitrary points $u_0, \ldots, u_{n-1}$, multipoint evaluation costs $O(n^2)$ operations in F by using Horner's rule n times. We have mentioned in section 3 Pan's (1966) result that Horner's rule is optimal: one evaluation requires at least n multiplications. One might be tempted to think that then n evaluations require at least n^2 multiplications. This is wrong, and our goal in this section is to see that mass-production of evaluations can be done much cheaper, provided that we have a fast way to compute polynomial divisions with remainder. In fact, it can be shown that interpolation can be done with the same asymptotic bound on the number of operations as multipoint evaluation, but we will not prove this here.

First, we state the following lemma, the proof of which is left as an exercise.

LEMMA 6.4. *Let $S, T : \mathbb{N} \longrightarrow \mathbb{N}$ be monotone functions with $S(2n) \ge 2S(n)$ for all $n \in \mathbb{N}$ and*

$$\begin{array}{rcl} T(1) & = & 0, \\ T(2^{i+1}) & = & 2T(2^i) + cS(2^i) \text{ for } i \ge 0, \end{array}$$

where $c \in \mathbb{N}$ is a constant. (We may think of S and T as being cost measures for certain algorithms, where the second algorithm is a recursive one dividing a problem of size 2^{i+1} into two problems of size 2^i plus c calls of the first algorithm on inputs of size 2^i.) Then

$$T(2^i) \leq \frac{c}{2} i S(2^i) = \frac{c}{2} S(n) \log n$$

for $n = 2^i$.

The idea of the algorithm is to cut the point set $\{u_0, \ldots, u_{n-1}\}$ into two halves of equal cardinality and to proceed recursively with each of the two halves. This leads to a binary tree of depth k with root $\{u_0, \ldots, u_{n-1}\}$ and the singletons $\{u_i\}$ for $0 \leq i < n$ at the leaves (see Figure 6.4 for an illustration).

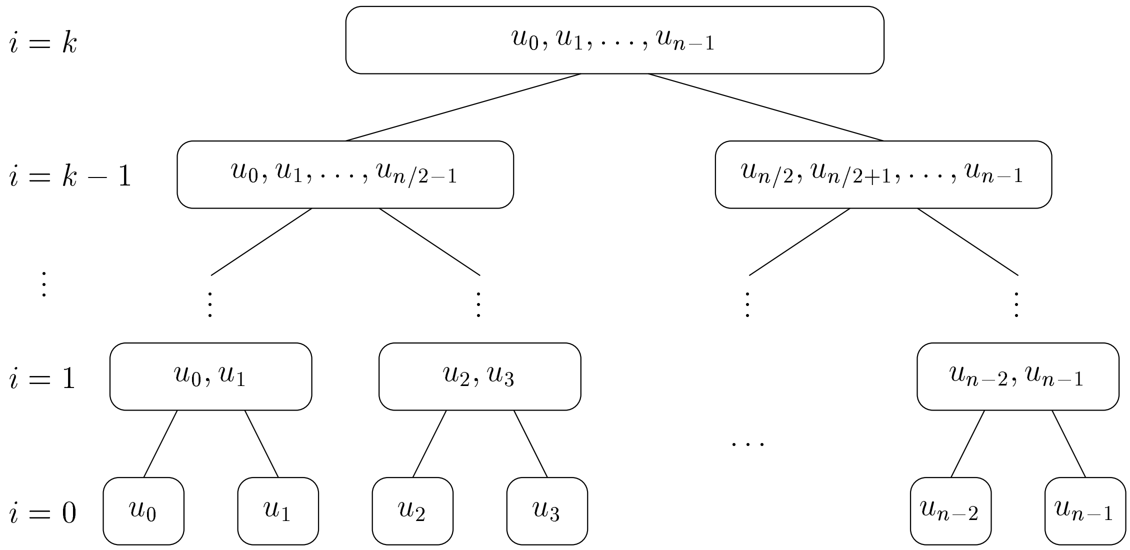


Figure 6.4: Binary tree for the multipoint evaluation algorithm.

We let $m_i = x - u_i$ as above, and define

$$M_{i,j} = m_{j \cdot 2^i} \cdot m_{j \cdot 2^i + 1} \cdots m_{j \cdot 2^i + (2^i - 1)} = \prod_{0 \leq l < 2^i} m_{j \cdot 2^i + l}$$

for $0 \leq i \leq k = \log_2 n$ and $0 \leq j < 2^{k-i}$. Thus each $M_{i,j}$ is a subproduct with 2^i factors of $m = \prod_{0 \leq l < n} m_l = M_{k,0}$ and satisfies for each i, j the recursive equations

$$\begin{aligned} M_{0,j} &= m_j, \\ M_{i+1,j} &= M_{i,2j} \cdot M_{i,2j+1}. \end{aligned}$$

Note that $M_{i,j}$ is the monic squarefree polynomial whose zero set is the jth node from the left at level i of the tree in Figure 6.4.

LEMMA 6.5. *All of the subproducts $M_{i,j}$ for $0 \le i \le k$ and $0 \le j < 2^{k-i}$ can be computed with $O(\mathsf{M}(n)\log n)$ operations in F, where $\mathsf{M}(n)$ is a multiplication time for $F[x]$.*

PROOF. We associate the polynomials $M_{i,j}$ with the vertices of the binary tree in Figure 6.4, as above. The polynomial at a vertex is the product of the polynomials at the two successors. There are $k+1$ levels, and the degree of each subproduct at level i is $\deg M_{i,j} = 2^i$. The computation along the tree proceeds from the bottom. Let $T(n) = T(2^k)$ denote the cost for computing all polynomials in the tree. Then T satisfies the recursive equations

$$\begin{aligned} T(1) &= 0, \\ T(2^k) &= 2T(2^{k-1}) + \mathsf{M}(2^{k-1}) \text{ for } k \ge 1, \end{aligned}$$

and hence $T(2^k) \le \frac{1}{2}k\mathsf{M}(2^k) \in O(\mathsf{M}(n)\log n)$, by Lemma 6.4. Less formally, the sum of all degrees at each level is n, and all multiplications at one level can be done in time $O(\mathsf{M}(n))$. Furthermore, there are $\log n$ levels. □

The computation of all the subproducts $M_{i,j}$ is a precomputation stage for the fast multipoint evaluation algorithm that we are going to present now. If several polynomials have to be evaluated at the same points $u_0, \ldots, u_{n-1}$, it is sufficient to carry out the precomputation stage only once in advance. We denote by $\mathsf{D}(n)$ the division time in $F[x]$, i.e., a polynomial in $F[x]$ of degree less than $2n$ can be divided with remainder by a polynomial of degree n using $O(\mathsf{D}(n))$ operations in F, and assume that $\mathsf{D}(2n) \ge 2 \cdot \mathsf{D}(n)$ for all $n \in \mathbb{N}$.

THEOREM 6.6. *Suppose that we have precomputed all the polynomials $M_{i,j}$. Then the multipoint evaluation problem can be solved with $O(\mathsf{D}(n)\log n)$ operations in F.*

PROOF. We present a divide-and-conquer algorithm that proceeds top down along the tree in Figure 6.4. We divide f by $M_{k-1,0}$ and $M_{k-1,1}$ with remainder, i.e., we compute $q_0, r_0, q_1, r_1 \in F[x]$ of degree $< \frac{n}{2}$ with

$$\begin{aligned} f &= q_0 M_{k-1,0} + r_0, \\ f &= q_1 M_{k-1,1} + r_1. \end{aligned}$$

Then

$$f(u_i) = \begin{cases} q_0(u_i)M_{k-1,0}(u_i) + r_0(u_i) = r_0(u_i) & \text{if } 0 \le i < \frac{n}{2}, \\ q_1(u_i)M_{k-1,1}(u_i) + r_1(u_i) = r_1(u_i) & \text{if } \frac{n}{2} \le i < n. \end{cases}$$

Hence we can continue recursively by evaluating r_0 at $u_0, \ldots, u_{\frac{n}{2}-1}$ and r_1 at $u_{\frac{n}{2}}, \ldots, u_{n-1}$.

We have reduced the multipoint evaluation problem of size n to two divisions with remainder plus two multipoint evaluation problems of size $\frac{n}{2}$. Let $U(n) = U(2^k)$ denote the cost for the recursive process. Then

$$\begin{aligned} U(1) &= 0, \\ U(2^k) &= 2U(2^{k-1}) + 2\mathsf{D}(2^{k-1}) \text{ for } k \ge 1, \end{aligned}$$

so that $U(2^k) \le k \cdot \mathsf{D}(2^k) \in O(\mathsf{D}(n)\log n)$, by Lemma 6.4. □

We will see later in this course that $\mathsf{D}(n) \in O(\mathsf{M}(n))$, and hence have the following consequence.

Corollary 6.7. *Evaluation of a polynomial $f \in F[x]$ of degree less than n at n points in F (including the precomputation stage) can be performed using $O(\mathsf{M}(n)\log n)$ operations in F.*

Exercise 6.8. *Let $m > n$, say $m = n^2$, and consider the problem of evaluating m polynomials of degree n at n points. What is the best running time you can find for this problem, using the above method? Same question for evaluating n polynomials of degree m at n points.*

Even without knowing how to do division of polynomials with remainder fast in general, we get a fast multipoint evaluation algorithm if the polynomials $M_{i,j}$ are sparse. This will be used at the end of this section.

6.2. Multipoint evaluation and interpolation at subspaces of finite fields. In the following, we collect some facts about finite fields that will be useful later. For their proofs, we refer to Lidl & Niederreiter (1983).

- For a prime $p \in \mathbb{N}$, $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \mathbb{Z}/(p)$ is a finite field with p elements.
- Any finite field F contains $\mathbb{F}_p$ as a subfield for some prime $p \in \mathbb{N}$, and $p \ge 1$ is uniquely determined by F by the condition that p be minimal with

$$\underbrace{1 + 1 + \cdots + 1}_{p \text{ times}} = 0.$$

  $\mathbb{F}_p$ is the *prime field* of F, p is the *characteristic* of F, and we write $p = \mathrm{char} F$.
- Any finite field F is a vector space over its prime field $\mathbb{F}_p$, i.e., $F \cong (\mathbb{F}_p)^n$ as vector spaces over $\mathbb{F}_p$ (not as fields!) for some $n \in \mathbb{N}$, called the *degree* of F over $\mathbb{F}_p$. In particular, F is of prime power order $\#F = q = p^n$.
- For an irreducible polynomial $a \in \mathbb{F}_p[x]$ of degree n, the factor ring $F = \mathbb{F}_p[x]/(a)$ is a finite field with p^n elements. The n elements

$$1 \bmod a, x \bmod a, \ldots, x^{n-1} \bmod a \in F$$

  are an $\mathbb{F}_p$-basis of F.
- Finite fields with p^n elements exist for all primes $p \in \mathbb{N}$ and integers $n \in \mathbb{N}$, and any two finite fields with p^n elements are isomorphic (as fields). We use the notation $\mathbb{F}_{p^n}$ for any such finite field.

- The map
$$\varphi\colon \begin{cases} \mathbb{F}_{p^n} & \longrightarrow & \mathbb{F}_{p^n} \\ \alpha & \longmapsto & \alpha^p \end{cases}$$
is an automorphism of the finite field $\mathbb{F}_{p^n}$, called the *Frobenius automorphism.* The following holds for all $\alpha, \beta \in \mathbb{F}_{p^n}$:
$$\begin{array}{rcl} (\alpha+\beta)^p & = & \alpha^p + \beta^p, \\ (\alpha\beta)^p & = & \alpha^p\beta^p, \\ \alpha^p = \alpha & \Longleftrightarrow & \alpha \in \mathbb{F}_p. \end{array} \tag{6.1}$$
The last property, which in the language of Galois theory says that $\mathbb{F}_p$ is the fixed field of φ, is often referred to as *Fermat's Little Theorem* (not to be confused with Fermat's Last Theorem, which was open for about 300 years and only very recently proven). It may equivalently be stated in the form
$$x^p - x = \prod_{u\in\mathbb{F}_p} (x-u). \tag{6.2}$$
Even over an extension field of $\mathbb{F}_p$, the roots of $x^p - x$ are exactly the elements of $\mathbb{F}_p$, which is just another way to state that $\mathbb{F}_p$ is the fixed field of φ.
- All of the above can be generalized to field extensions of an arbitrary (not necessarily prime) finite field $\mathbb{F}_q$, where q is a prime power. If $\mathbb{F}_{q^n}$ is an extension field of $\mathbb{F}_q$ of degree n, the Frobenius automorphism $\varphi\colon \mathbb{F}_{q^n} \longrightarrow \mathbb{F}_{q^n}$ is given by $\varphi(a) = a^q$ for $a \in \mathbb{F}_{q^n}$, and (6.1) and (6.2) still hold with q instead of p.

Now we are ready to discuss the multipoint evaluation algorithm. Let $m \in \mathbb{N}$, $q \in \mathbb{N}$ a prime power, and $\mathbb{F}_q \subseteq \mathbb{F}_{q^m}$ finite fields. Later, we will mainly use $q = 2$, but the proofs are not harder in the general case. The idea of the algorithm is due to Cantor (1989), who only considered the case where $q = p$ is a prime and m is a power of p, and was generalized to arbitrary q and m in von zur Gathen & Gerhard (1995).

Suppose that $\beta_1, \ldots, \beta_m$ is some fixed basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Then we define $\mathbb{F}_q$-linear subspaces of $\mathbb{F}_{q^m}$ by
$$W_i = \langle \beta_1, \ldots, \beta_i \rangle = \{ \sum_{1\le j\le i} c_j\beta_j \colon c_1, \ldots, c_i \in \mathbb{F}_q \}$$
for $0 \le i \le m$. Obviously
$$\{0\} = W_0 \subsetneq W_1 \subsetneq \cdots \subsetneq W_{m-1} \subsetneq W_m = \mathbb{F}_{q^m},$$
$\dim_{\mathbb{F}_q} W_i = i$ and $\# W_i = q^i$ for $0 \le i \le m$.

From the definition of W_i, we derive the recursive decomposition
$$W_i = \bigcup_{u\in\mathbb{F}_q} (u\beta_i + W_{i-1}) \text{ for } 1 \le i \le m, \tag{6.3}$$

where $u\beta_i + W_{i-1} = \{u\beta_i + w : w \in W_{i-1}\}$ is a coset of W_{i-1} in the group theoretic sense. Since $\beta_i \notin W_{i-1}$, the union in (6.3) is disjoint, and the cosets $(u\beta_i + W_{i-1})_{u\in\mathbb{F}_q}$ are a partition of W_i.

We want to evaluate a polynomial $f \in \mathbb{F}_{q^m}[x]$ of degree less than q^k at all the points of W_k for some $k \in \{0, \ldots, m\}$. In analogy to the general multipoint evaluation algorithm, we divide the original problem into q (not 2) subproblems of size q^{k-1} along the above decomposition of the W_k, and then proceed recursively. This corresponds to a recursive descent from the root to the leaves of a complete $q$ary tree, where the vertices at level i are q^{k-i} pairwise disjoint cosets of W_i for $0 \le i \le k$. Before we can do that, we have to precompute the polynomials $M_{i,j}$ of degree q^i vanishing exactly at the points of the jth vertex at level i of the tree.

We define

$$s_i = \prod_{\alpha\in W_i} (x - \alpha) \in \mathbb{F}_{q^m}[x]$$

for $0 \le i \le m$. Then obviously

$$x = s_0 \mid s_1 \mid \cdots \mid s_{m-1} \mid s_m = x^{q^m} - x,$$

s_i is monic of degree q^i, and $s_i(\alpha) = 0$ if and only if $\alpha \in W_i$, for all $\alpha \in \mathbb{F}_{q^m}$ and $0 \le i \le m$.

Example 6.9. *Let $q = 2$ and $\mathbb{F}_q = \mathbb{F}_2 = \{0, 1\}$. Cantor (1989) showed that for any power $m \in \mathbb{N}$ of 2, there is a basis $\beta_1 = 1, \beta_2, \ldots, \beta_m$ of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ such that $s_1 = x^2 + x$ and*

$$s_i = s_1(s_{i-1}) = s_{i-1}^2 + s_{i-1} = \sum_{0\le j\le i} \binom{i}{j} x^{2^j}$$

for $1 \le i \le m$. In this case, the polynomials s_i have coefficients 0 and 1 only. The following table shows the s_i for $0 \le i \le 8$ and $i = 16$.

$$\begin{aligned}
s_0 &= x,\\
s_1 &= x^2 + x,\\
s_2 &= x^4 + x,\\
s_3 &= x^8 + x^4 + x^2 + x,\\
s_4 &= x^{16} + x,\\
s_5 &= x^{32} + x^{16} + x^2 + x,\\
s_6 &= x^{64} + x^{16} + x^4 + x,\\
s_7 &= x^{128} + x^{64} + x^{32} + x^{16} + x^8 + x^4 + x^2 + x,\\
s_8 &= x^{256} + x,\\
&\ \vdots\\
s_{16} &= x^{65536} + x.
\end{aligned}$$

LEMMA 6.10. *Let $0 \le i \le m$.*

(i) s_i *is a* linearized *polynomial, i.e.,*

$$s_i(x+y) = s_i(x) + s_i(y) \text{ and } s_i(cx) = cs_i(x)$$

for indeterminates x, y over $\mathbb{F}_{q^m}$ and all $c \in \mathbb{F}_q$.

(ii) s_i *is a q-*polynomial *, i.e.,*

$$s_i = \sum_{0 \le j \le i} s_{ij} x^{q^j}$$

for some appropriate $s_{i,0}, \ldots, s_{i,i} \in \mathbb{F}_{q^m}$.

(iii) The recursion formula $s_i = s_{i-1}^q - s_{i-1}(\beta_i)^{q-1} s_{i-1}$ holds if $i \ge 1$.

(iv) For $\beta \in \mathbb{F}_{q^m}$, we have

$$\prod_{\alpha \in \beta + W_i} (x - \alpha) = s_i - s_i(\beta) = \prod_{u \in \mathbb{F}_q} \Big(s_{i-1} - s_{i-1}(\beta) - u s_{i-1}(\beta_i) \Big),$$

where of course $i = 0$ has to be excluded for the last equality. (Note that the first equation reduces to the definition of s_i if $\beta = 0$.)

PROOF. We will first prove (i), (iii), and the special case

$$s_i = \prod_{u \in \mathbb{F}_q} \Big(s_{i-1} - u s_{i-1}(\beta_i) \Big) \text{ if } i \ge 1 \tag{6.4}$$

of (iv) by simultaneous induction on i. The case $i = 0$ is immediate, and we assume that $i \ge 1$. We first show that the polynomial $t_i = \prod_{u \in \mathbb{F}_q} \Big(s_{i-1} - u s_{i-1}(\beta_i) \Big)$ also vanishes at all the points of W_i, and then (6.4) follows by the fact that $s_i(\alpha) = 0$ if and only if $\alpha \in W_i$ for all $\alpha \in \mathbb{F}_{q^m}$ and both s_i and t_i are monic polynomials of degree q^i. For $\alpha \in \mathbb{F}_{q^m}$, we have

$$\begin{aligned}
t_i(\alpha) = 0 \quad &\Longleftrightarrow \quad \exists u \in \mathbb{F}_q \quad s_{i-1}(\alpha) - u s_{i-1}(\beta_i) = 0 \\
&\Longleftrightarrow \quad \exists u \in \mathbb{F}_q \quad s_{i-1}(\alpha - u\beta_i) = 0 \\
&\Longleftrightarrow \quad \exists u \in \mathbb{F}_q \quad \alpha - u\beta_i \in W_{i-1} \\
&\Longleftrightarrow \quad \exists u \in \mathbb{F}_q \quad \alpha \in u\beta_i + W_{i-1} \\
&\Longleftrightarrow \quad \alpha \in W_i.
\end{aligned}$$

We used the induction hypothesis (i) in the second line, and (6.3) in the last line.

We write $v = s_{i-1}(\beta_i) \in \mathbb{F}_{q^m}$ for short. Since $\beta_i \notin W_{i-1}$, we have $v \ne 0$. Then (iii) follows from (6.4) and (6.2) when we substitute $v^{-1} s_{i-1}$ for x:

$$\begin{aligned}
\prod_{u \in \mathbb{F}_q} (s_{i-1} - uv) &= v^q \prod_{u \in \mathbb{F}_q} (v^{-1} s_{i-1} - u) = v^q (v^{-q} s_{i-1}^q - v^{-1} s_{i-1}) \\
&= s_{i-1}^q - v^{q-1} s_{i-1}.
\end{aligned}$$

This finishes the induction step in (iii).

Now for the induction step in (i). With v as above, we conclude from what we just proved and the induction hypothesis that

$$\begin{aligned} s_i(x+y) &= s_{i-1}(x+y)^q - v^{q-1}s_{i-1}(x+y) \\ &= \Big(s_{i-1}(x)+s_{i-1}(y)\Big)^q - v^{q-1}\Big(s_{i-1}(x)+s_{i-1}(y)\Big) \\ &= s_{i-1}(x)^q - v^{q-1}s_{i-1}(x) + s_{i-1}(y)^q - v^{q-1}s_{i-1}(y) \\ &= s_i(x)+s_i(y), \end{aligned}$$

where we used the additivity of the Frobenius automorphism in the third equation. The proof of $s_i(cx) = cs_i(x)$ is similar and therefore left as an exercise, and the induction step in (i) is finished.

(iv) now follows by repeated use of the linearity properties (i):

$$\begin{aligned} s_i(x) - s_i(\beta) &= s_i(x-\beta) = \prod_{\alpha\in W_i}(x-\beta-\alpha) = \prod_{\alpha\in\beta+W_i}(x-\alpha), \\ s_i(x) - s_i(\beta) &= s_i(x-\beta) = \prod_{u\in\mathbb{F}_q}\Big(s_{i-1}(x-\beta) - us_{i-1}(\beta_i)\Big) \\ &= \prod_{u\in\mathbb{F}_q}\Big(s_{i-1}(x) - s_{i-1}(\beta) - us_{i-1}(\beta_i)\Big). \end{aligned}$$

Finally, (ii) follows inductively from (iii) since $s_0 = x^{q^0}$ is a q-polynomial and one easily proves that q-polynomials are closed under addition, multiplication by scalars, and composition (Exercise: are they also closed under multiplication?). The details are left as an exercise. □

Statement (iii) of the above Lemma easily allows us, given $\beta_1, \ldots, \beta_m$, to compute the polynomials $s_0, \ldots, s_m$. The last statement says that the polynomial $M_{i,j}$ of degree q^i having a coset $\beta + W_i$ for some $\beta \in \mathbb{F}_{q^m}$ (depending on j) as zero set is exactly the polynomial $s_i - s_i(\beta)$. Finally, statement (ii) implies that the polynomials s_i are extremely sparse: the number of nonzero coefficients of s_i is logarithmic in the degree! This is the very reason that the following multipoint evaluation algorithm is really fast.

Algorithm 6.11. *Multipoint evaluation at all points of W_k.*
Let $0 \le k \le m$. We assume that the polynomials s_i and the values $s_i(\beta_j)$ for $0 \le i < j \le k$, which do not depend on the particular input, have been precomputed and stored.
Input: $i \in \{0, \ldots, k\}$, $\beta = \sum_{i<j\le k} c_j\beta_j \in W_k$ *with* $c_{i+1}, \ldots, c_k \in \mathbb{F}_q$, *and* $f \in \mathbb{F}_{q^m}[x]$ *of degree less than* q^i. *(Initially,* $i = k$ *and* $\beta = 0$.*)*
Output: $f(\alpha)$ *for all* $\alpha \in \beta + W_i$.

1. *If* $i = 0$ *return* f.

2. *Compute* $s_{i-1}(\beta) = \sum_{i \le j < k} c_j s_{i-1}(\beta_j)$.

3. (Division with remainder) *For all* $u \in \mathbb{F}_q$, *compute* $g_u, r_u \in \mathbb{F}_{q^m}[x]$ *with*

$$f = g_u\Big(s_{i-1} - s_{i-1}(\beta) - u s_{i-1}(\beta_i)\Big) + r_u \text{ and } \deg r_u < q^{i-1}.$$

4. (Divide-and-conquer) *For all* $u \in \mathbb{F}_q$, *recursively call the algorithm with input* $(i-1, \beta + u\beta_i, r_u)$, *yielding* $\gamma(\alpha) = r_u(\alpha)$ *for all* $\alpha \in \beta + u\beta_i + W_{i-1}$.

5. *Return* $\gamma(\alpha)$ *for all* $\alpha \in \beta + W_i$.

THEOREM 6.12. *The algorithm works correctly and for* $i = k$ *uses at most*

$$\frac{(q-1)}{2}k^2q^k + \frac{(q-1)}{2}kq^k$$

multiplications and

$$\frac{(q-1)}{2}k^2q^k + \frac{(q-7)}{2}kq^k$$

scalar operations, i.e., additions and multiplications by elements of $\mathbb{F}_q$, *in* $\mathbb{F}_{q^m}$.

PROOF. For the correctness, we first note that $\beta + W_i = \bigcup_{u\in\mathbb{F}_q}(u\beta_i + \beta + W_{i-1})$ is a partition of $\beta + W_i$ into cosets of W_{i-1}, analogous to (6.3). If $\alpha \in \beta + u\beta_i + W_{i-1}$, then

$$\begin{aligned}\gamma(\alpha) &= r_u(\alpha) = f(\alpha) - g_u(\alpha)\Big(s_{i-1}(\alpha) - s_{i-1}(\beta) - u s_{i-1}(\beta_i)\Big)\\ &= f(\alpha) - g_u(\alpha)s_{i-1}(\alpha - \beta - u\beta_i) = f(\alpha),\end{aligned}$$

since $\alpha - \beta - u\beta_i \in W_{i-1}$. This proves partial correctness by induction on i (the case $i = 0$ is clear).

For the cost analysis, we only note that each division with remainder in step (iii) can be performed with $i(q^i - q^{i-1})$ multiplications and additions in $\mathbb{F}_{q^m}$ each, using long division and employing the sparsity of the divisor polynomial. The details can be found in von zur Gathen & Gerhard (1995). □

Note that the factor $\frac{(q-1)}{2}$ is very small in the important case $q = 2$, and the asymptotic running time of the algorithm is then $O(n \log_2^2 n)$ operations in $\mathbb{F}_{q^m}$, when $n = 2^k$ is the degree of the input polynomial.

EXAMPLE 6.13. *Let* $q = 2$ *and* $m = 3$. *The polynomial* $a = y^3 + y + 1 \in \mathbb{F}_2[y]$ *is irreducible (why?), and hence* $\mathbb{F}_8 = \mathbb{F}_2[y]/(a)$ *is a finite field with 8 elements. An element of* $\beta \in \mathbb{F}_8$ *can be uniquely represented by a polynomial expression of the form* $\beta = c_2y^2 + c_1y + c_0 \bmod a$ *with* $c_0, c_1, c_2 \in \mathbb{F}_2$. *The polynomial* $c_2y^2 + c_1y + c_0 \in \mathbb{F}_q[y]$ *is called the* canonical representative *of* β. *Two such elements are added componentwise, and their product is computed by multiplying their canonical representatives and dividing the result by* a *with remainder. For example,*

$$(y^2 + y + 1)(y^2 + y) = y^4 + y = y(y^3 + y + 1) + y^2,$$

so that $(y^2 + y + 1 \bmod a)(y^2 + y \bmod a) = y^2 \bmod a$, *or equivalently,* $(y^2 + y + 1)(y^2 + y) \equiv y^2 \bmod a$. *If we write* $\alpha = y \bmod a$ *for short, then this reads as* $(\alpha^2 + \alpha + 1)(\alpha^2 + \alpha) = \alpha^2$. *This distinction between an element of* $\mathbb{F}_8$ *and its representation in* $\mathbb{F}_2[y]$ *is purely conceptual; in order to compute in* $\mathbb{F}_8$, *one will always use the representation in* $\mathbb{F}_2[y]$.

We take $\beta_1 = 1$, $\beta_2 = \alpha$, *and* $\beta_3 = \alpha^2$ *as basis of* $\mathbb{F}_8$ *over* $\mathbb{F}_2$. *Then the subspaces* W_i *for* $0 \le i \le 3$ *are*

$$\begin{aligned} W_0 &= \{0\}, \\ W_1 &= \{0,1\} = \mathbb{F}_2, \\ W_2 &= \{0,1,\alpha,\alpha+1\}, \\ W_3 &= \{0,1,\alpha,\alpha+1,\alpha^2,\alpha^2+1,\alpha^2+\alpha,\alpha^2+\alpha+1\} = \mathbb{F}_8. \end{aligned} \tag{6.5}$$

Now we compute the polynomials

$$\begin{aligned} s_0 &= x, \\ s_1 &= s_0^2 + s_0(\beta_1)s_0 = x^2 + x, \\ s_2 &= s_1^2 + s_1(\beta_2)s_1 = x^4 + (\alpha^2+\alpha+1)x^2 + (\alpha^2+\alpha)x, \\ s_3 &= s_2^2 + s_2(\beta_3)s_2 = x^8 + x, \end{aligned}$$

and the values of $s_i(\beta_j)$ *for* $0 \le i < j \le 3$:

$s_i(\beta_j)$	$j=1$	$j=2$	$j=3$
$i=0$	1	α	α^2
$i=1$	0	$\alpha^2+\alpha$	α
$i=2$	0	0	$\alpha+1$

We note that the values below the main diagonal are zero since $\beta_j \in W_i$ *if* $j \le i$.

Figure 6.5 shows the computation tree for the recursive multipoint evaluation algorithm. The vertices are the different subsets of $W_3 = \mathbb{F}_8$ *occurring in the course of the algorithm, which starts at the root with the evaluation at all points of* W_3 *and recursively divides the problem into two subproblems of half the size until it arrives at the leaves.*

To illustrate the algorithm, we now evaluate the polynomial $f = x^6 + \alpha x^4 + 1 \in \mathbb{F}_8[x]$ *at the 8 points of* $\mathbb{F}_8$. *According to Algorithm 6.11, we start with* $i = 3$ *and* $\beta = 0$. *In the division step, we divide* f *by* s_2 *and* $s_2 + s_2(\beta_3)$, *respectively, with remainder, and get*

$$\begin{aligned} f &= (x^2+\alpha^2+1)\cdot s_2 + \Big((\alpha^2+\alpha)x^3 + (\alpha^2+\alpha)x^2 + (\alpha+1)x + 1\Big) \\ &= (x^2+\alpha^2+1)(s_2+\alpha+1) + \Big((\alpha^2+\alpha)x^3 + (\alpha^2+1)x^2 + (\alpha+1)x + a^2 + 1\Big), \end{aligned}$$

i.e., $r_0 = (\alpha^2+\alpha)x^2 + (\alpha+1)x + 1$ *and* $r_1 = (\alpha^2+\alpha)x^3 + (\alpha^2+1)x^2 + (\alpha+1)x + a^2 + 1$. *Now, we recursively evaluate* r_0 *at* W_2 *and* r_1 *at* $\beta_3 + W_2$, *recursively calling the algorithm with input* $(2, 0, r_0)$ *and* $(2, \beta_3, r_1)$, *respectively.*

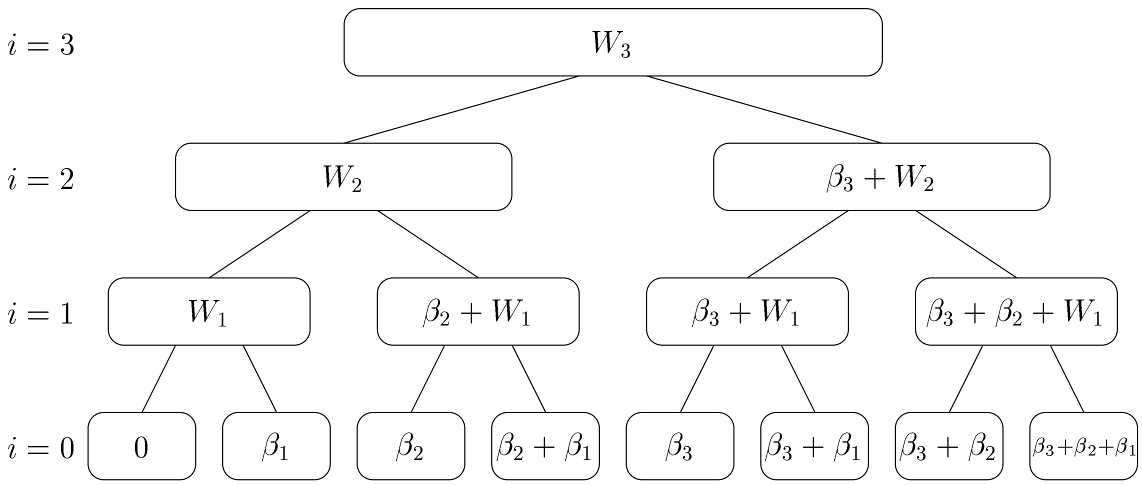


Figure 6.5: Computation tree for the multipoint evaluation over $\mathbb{F}_8$.

The first recursive call now handles the left subtree in Figure 6.5. It divides r_0 by s_1 and $s_1 + s_1(\beta_2)$ with remainder, yielding

$$((\alpha^2 + \alpha)x) \cdot s_1 + ((\alpha + 1)x + 1) = r_0 = ((\alpha^2 + \alpha)x)(s_2 + \alpha^2 + \alpha) + (x + 1).$$

Then again two recursive calls with inputs $(1, 0, (\alpha + 1)x + 1)$ and $(1, \beta_2, x + 1)$ are performed to evaluate the two remainders at W_1 and $\beta_2 + W_1$, respectively. We cut the recursion here and calculate by hand that the results are $(1, \alpha)$ and $(\alpha + 1, \alpha)$, so that the first recursive call returns

$$\Big(r_0(0), r_0(1), r_0(\alpha), r_0(\alpha + 1)\Big) = (1, \alpha, \alpha + 1, \alpha)$$

(simply concatenate the results of the recursive calls).

The second recursive call on the top level starts at the root of the right subtree in Figure 6.5. Division of r_1 with remainder by $s_1 + s_1(\beta_3)$ and $s_1 + s_1(\beta_3) + s_1(\beta_2)$ yields

$$\begin{aligned} r_1 &= ((\alpha^2 + \alpha)x + \alpha + 1)(s_1 + \alpha) + ((\alpha^2 + \alpha + 1)x + 1) \\ &= ((\alpha^2 + \alpha)x + \alpha + 1)(s_1 + \alpha^2) + ((\alpha^2 + 1)x + \alpha). \end{aligned}$$

Two further recursive calls with inputs $(1, \beta_3, (\alpha^2+\alpha+1)x+1)$ and $(1, \beta_3+\beta_2, (\alpha^2 + 1)x + \alpha)$ for the evaluation of the two remainders at $\beta_3 + W_1$ and $\beta_3 + \beta_2 + W_1$, respectively, return $(\alpha, \alpha^2 + 1)$ and $(1, \alpha^2)$, and the second recursive call returns

$$\Big(r_1(\alpha^2), r_1(\alpha^2 + 1), r_1(\alpha^2 + \alpha), r_1(\alpha^2 + \alpha + 1\Big) = (\alpha, \alpha^2 + 1, 1, \alpha^2).$$

Combining, i.e., concatenating the results of the two recursive calls on top level, the values of f at the points of $\mathbb{F}_8$ are

$$(1, \alpha, \alpha + 1, \alpha, \alpha, \alpha^2 + 1, 1, \alpha^2),$$

where the order is according to the natural order of the elements of W_3 as in (6.5). In general, if $\beta_1, \ldots, \beta_m$ is an arbitrary $\mathbb{F}_2$-basis of $\mathbb{F}_{2^m}$, the concatenation scheme works if one identifies an element $\sum_{1\leq j\leq m} c_j\beta_j \in \mathbb{F}_{2^m}$ with its coefficient vector $(c_1, \ldots, c_m) \in (\mathbb{F}_2)^m$ and orders the vectors by ascending value of $\sum_{1\leq j\leq m} c_j 2^{j-1}$.

This example was generated with the friendly help of the computer algebra system MuPAD.

To arrive at an algorithm for polynomial multiplication over $\mathbb{F}_{q^m}$, it remains to do the interpolation fast. The interpolation algorithm at all points of W_k for some $k \in \{0, \ldots, m\}$ turns out to be essentialy the "reverse" of the multipoint evaluation algorithm (the multipoint evaluation algorithm proceeds from the root to the leaves along the $q$ary computation tree, and the interpolation algorithm in the opposite direction). However, the details are a bit more technical and not discussed in this course. Without proof, we quote the following theorem from von zur Gathen & Gerhard (1995).

THEOREM 6.14. *For $0 \leq k \leq m$, interpolation at all points of W_k can be done with at most*

$$\frac{q+1}{2}k^2q^k + \frac{q+5}{2}kq^k$$

multiplications and

$$\frac{q+1}{2}k^2q^k + \frac{3q+5}{2}kq^k$$

scalar operations in $\mathbb{F}_{q^m}$.

COROLLARY 6.15. *Two polynomials $f, g \in \mathbb{F}_{q^m}[x]$ with $\deg(fg) < n$, where $q \leq n \leq q^m$, can be multiplied using less than*

$$\frac{3q^2-q}{2}n\log_q^2 n + \frac{15q^2+7q}{2}n\log_q n$$

multiplications and

$$\frac{3q^2-q}{2}n\log_q^2 n + \frac{19q^2+35q}{2}n\log_q n$$

scalar operations in $\mathbb{F}_{q^m}$.

PROOF. To multiply f and g, we evaluate both polynomials at all points of W_k, where $k \in \{0, \ldots, m\}$ satisfies $q^{k-1} < n \leq q^k$, multiply the values pointwise, and finally interpolate to recover fg. The claim follows by a routine calculation from Theorems 6.12 and 6.14. □

Our goal in this section was to find a fast multiplication algorithm for polynomials f, g over $\mathbb{F}_2$. Using the above techniques, we might choose the minimal $m \in \mathbb{N}$ such that $\deg(fg) \leq 2^m$ and regard f and g as polynomials over the larger field $\mathbb{F}_{2^m}$. (We do not discuss here how to *construct* $\mathbb{F}_{2^m}$, i.e., how to compute an irreducible polynomial of degree m over $\mathbb{F}_2$. This is justified by the fact that in practice, computer algebra systems can only handle polynomials of degree bounded by the hardware memory size, and one can once and for all precompute *one* irreducible polynomial for an m sufficiently large for all such polynomials, say $m = 64$.) Since one arithmetic operation (addition or multiplication) in $\mathbb{F}_{2^m}$ can be performed with $O(m^2)$ arithmetic operations in $\mathbb{F}_2$ using school methods, this will yield a multiplication time of $\mathsf{M}(n) = n \log^4 n$ for $\mathbb{F}_2$. It turns out that one factor $\log n$ can be saved if a more space efficient approach is used. Instead of regarding each coefficient in $\{0, 1\}$ of the input polynomials as an element of $\mathbb{F}_{2^m}$, one packs every $\frac{m}{2}$ consecutive coefficients into one element of $\mathbb{F}_{2^m}$, thereby reducing the degree of the resulting polynomials over $\mathbb{F}_{2^m}$ by a factor of $\frac{m}{2} \in O(\log n)$. Afterwards, it is possible to uniquely recover the coefficients of fg from the product polynomial over $\mathbb{F}_{2^m}$. The details can be found in von zur Gathen & Gerhard (1995), and also the proof of the final theorem.

THEOREM 6.16. *The above algorithm multiplies two polynomials $f, g \in \mathbb{F}_2[x]$ with $\deg(fg) < n$ with $O(n \log_2^3 n)$ operations in $\mathbb{F}_2$.*

7. Division of polynomials and Newton iteration

When F is a field, the polynomials over F form a Euclidean domain when the polynomial degree is used as the Euclidean domain degree. This means that there exists unique $q, r \in F[x]$ s.t. $a = qb + r$ where $r = 0$ or deg$r$ <deg$b$. The division problem is then to find q, r given a, b.

The usual "synthetic division" algorithm learned in school requires $O(n^2)$ field operations. Around 1972, Borodin and Moenck, Strassen, Sieveking, and Kung derived a division algorithm that cost $O(n \log^2 n \log\log n)$ field operations.

7.1. Division using Newton iteration. Let a and b be two polynomials of degree n and m respectively. Assume $m < n$ and $b_m = 1$. We wish to find $q(x)$ and $r(x)$ satisfying $a(x) = q(x)b(x) + r(x)$ with $r = 0$ or $\deg r < \deg b$.

Substituting $\frac{1}{y}$ for the variable x, we obtain

$$y^n a(\frac{1}{y}) = \left(y^{n-m} q(\frac{1}{y})\right)\left(y^m b(\frac{1}{y})\right) + y^{n-m+1}\left(y^{m-1} r(\frac{1}{y})\right).$$

For convenience, we define the *reversal* of a as $\mathrm{rev}_k(a) = y^k a(1/y)$. When a is a degree k polynomial, this is the polynomial with the coefficients of a reversed. For example, if

$$a = a_0 + a_1 x + \cdots + a_n x^n$$

then

$$\mathrm{rev}_n(a) = a_n + a_{n-1} y + \cdots + a_1 y^{n-1} + a_0 y^n.$$

Suppose we could compute $\mathrm{rev}_m(b)^{-1} \in F[[x]]$ exactly, then we could compute $q(x)$ and $r(x)$ as follows. We have

$$\mathrm{rev}_n(a) \equiv \mathrm{rev}_{n-m}(q) \cdot \mathrm{rev}_m(b) + y^{n-m+1}\mathrm{rev}_{m-1}(r).$$

Therefore,

$$\mathrm{rev}_n(a) \equiv \mathrm{rev}_{n-m}(q) \cdot \mathrm{rev}_m(b) \bmod y^{n-m+1},$$

and

$$\mathrm{rev}_n(a) \cdot \mathrm{rev}_m(b)^{-1} \equiv \mathrm{rev}_{n-m}(q) \bmod y^{n-m+1}.$$

We obtain $q = \mathrm{rev}_{n-m}(\mathrm{rev}_{n-m}(q))$ and $r = a - q \cdot b$. We note that the condition that b be monic implies that $\mathrm{rev}_m(b)(0) = 1$.

EXAMPLE 7.1. *Let $a = x^3 + 2x^2 + x + 2$ and $b = x^2 + x + 2$ be polynomials in $\mathbb{Z}_3[x]$. Then*

$$\mathrm{rev}_3(a) = 2y^3 + y^2 + 2y + 1,$$
$$\mathrm{rev}_2(b) = 2y^2 + y + 1.$$

We claim $\mathrm{rev}_2(b)^{-1} \equiv 2y + 1 \bmod y^2$ *in* $\mathbb{Z}_3[y]$. *Check:*

$$\begin{aligned}(2y+1)(2y^2+y+1) &= 4y^3 + 4y^2 + 3y + 1 \\ &\equiv 1 \bmod y^2 \qquad \text{in } \mathbb{Z}_3[y].\end{aligned}$$

$$\mathrm{rev}_1(q) \equiv (2y^3 + y^2 + 2y + 1)(2y + 1) \equiv y + 1 \bmod y^2.$$

So $q = x + 1$ *and* $r = a - qb = x$.

PROBLEM 7.2. *Given* $g \in F[x]$, *find* $h \in F[x]$ *satisfying* $hg \equiv 1 \bmod x^k$.

From numerical analysis courses, recall that Newton iteration involves computing successive approximations to solutions of $f(t) = 0$. From a suitable initial approximation t_0, subsequent approximations are computed using

$$t_{i+1} = t_i - \frac{f(t_i)}{f'(t_i)}.$$

For our task, we want to find (or approximate) a root of $1/t - g = 0$. The Newton iteration step is

$$t_{i+1} = t_i - \frac{1/t_i - g}{-1/t_i^2} = 2t_i - gt_i^2.$$

The following theorem tells us a good initial approximation and shows us that this method converges "quickly" to a solution.

THEOREM 7.3. *Let* $g, t_0, t_1, \ldots \in F[x]$, *with* $t_0 = 1$ *and*

$$t_{i+1} \equiv 2t_i - gt_i^2 \bmod x^{2^{i+1}},$$

for all i. Assume further that the lowest coefficient of g *satisfies* $g_0 = 1$. *Then*

$$gt_i \equiv 1 \bmod x^{2^i}, \textit{for all } i \geq 0.$$

PROOF. The proof is by induction on i. For $i = 0$ we have

$$gt_0 \equiv g_0 t_0 \equiv 1 \cdot 1 \equiv 1 \bmod x^{2^0}.$$

Assume $gt_i \equiv 1 \bmod x^{2^i}$. Now

$$\begin{aligned} 1 - gt_{i+1} &\equiv 1 - g(2t_i - gt_i^2) \\ &\equiv 1 - 2gt_i + g^2 t_i^2 \\ &\equiv (1 - gt_i)^2 \\ &\equiv 0 \bmod x^{2^{i+1}} \end{aligned}$$

We conclude then that for all $i \in \mathbb{N}$, $gt_i \equiv 1 \bmod x^{2^i}$. $\square$

As a result, we obtain the following algorithm to compute the inverse of $g \bmod x^k$.

ALGORITHM 7.4. *Inversion using Newton iteration.*
Input: $g = g_0 + g_1 x + \cdots + g_n x^n \in F[x]$ *with* $g_0 = 1$.
Output: $u \in F[x]$ *satisfying* $1 - gu \equiv 0 \bmod x^k$.

1. *Set* $t_0 = 1$, $r = \lceil \log_2 k \rceil$.
2. *For* $i = 1, \ldots, r$ *calculate*
$$t_i = (2t_{i-1} - g t_{i-1}^2) \bmod x^{2^i}$$
3. *Return* t_r.

If we want to compute the inverse of a polynomial $g \in F[x]$ with $g(0) \neq 0, 1$, then of course we apply the algorithm to $\frac{g}{g(0)}$ and multiply the result by $g(0)$. If $g(0) = 0$, however, no inverse of g modulo x^k exists. To see why, we assume that there is some $u \in F[x]$ such that $gu \equiv 1 \bmod x^k$, or equivalently, $gu + hx^k = 1$ for some appropriate polynomial $h \in F[x]$. Substituting $x = 0$ in the above equation, we arrive at a contradiction.

Theorem 7.5. *Algorithm 7.4 uses $O(\mathsf{M}(n))$ field operations to correctly compute the reciprocal. Here again $\mathsf{M}(n)$ is the multiplication time for polynomials over F. It is assumed that $\mathsf{M}(n) \geq n$ and $\mathsf{M}(2n) \geq 2\mathsf{M}(n)$.*

Proof. The number of field operations used for computing t_{i+1} from t_i in step 2 is bounded by $2\mathsf{M}(2^i) + 2 \cdot 2^i$, since the arithmetic is performed mod x^{2^i}. The time to compute the reciprocal for $k = n$ satisfies
$$\begin{aligned} T(k) &= \sum_{0<i<r} (2\mathsf{M}(2^i) + 2^{i+1}) \leq 4 \sum_{0<i<r} \mathsf{M}(2^i) \\ &\leq 4\mathsf{M}(2^r) \sum_{0<i<r} 2^{i-r} \leq 4\mathsf{M}(2^r) \in O(\mathsf{M}(n)). \quad \square \end{aligned}$$

We also have the corollary

Corollary 7.6. *For polynomials of degree n in $F[x]$, division with remainder requires $O(\mathsf{M}(n))$ field operations.*

It may seem circular to use an algorithm that uses the mod operation to compute division. However, we are using the mod operation to truncate the polynomial and thus we are using only a very simple form of division. It is similar to finding the quotient and remainder of a large number written in base 10 when divided by 10000. Division in this special case costs no field operations.

7.2. Newton iteration in more general domains. In order to get Newton iteration to work, we need some notion of convergence. In the case of real numbers, numbers are close together if there difference is small. For the polynomial example above, two polynomials were close together if a high power of x divided their difference. We can generalize these notions in the following definition.

Definition 7.7. *A valuation on a ring R is a map*
$$v : R \to \mathbb{R},$$
which satisfies

1. *$v(ab) = v(a)v(b)$, multiplicative*
2. *$v(a + b) \le v(a) + v(b)$, sub-additive*
3. *$v(a) \ge 0$ and $v(a) = 0$ if and only if $a = 0$, positive definite.*

The following are some common valuations on rings. A ring may, of course, have more than one possible valuation.

1. With $R = \mathbb{Z}$, $v(a) = |a|$.
2. With $R = \mathbb{Z}$ and p prime,

$$v_p(a) = \begin{cases} 0, & \text{if } a = 0; \\ p^{-n}, & \text{if } p^n \mid a \text{ but } p^{n+1} \not| \, a. \end{cases}$$

   This is called the *p-adic valuation.*

3. With $R = F[x]$, F a field,

$$v(a) = \begin{cases} 0, & \text{if } a = 0; \\ 2^{-n}, & \text{if } x^n \mid a \text{ but } x^{n+1} \not| \, a. \end{cases}$$

4. With $R = F[x]$, F a field,

$$v(a) = \begin{cases} 0, & \text{if } a = 0; \\ 2^{\deg(a)}, & \text{is } a \neq 0. \end{cases}$$

The notion close together can be expressed in terms of a valuation. In the polynomial case with valuation 3., we use the intuition that a polynomial a is small if $x^n \mid a$ for some large n. Two polynomials are close if their distance is small.

Definition 7.8. *The distance between a,b is defined as $d(a, b) = v(a - b)$.*

Example 7.9.

$$v_3(54) = 3^{-3} = \frac{1}{27}$$
$$v_3(55) = 1$$
$$v_3(54,000,000) = \frac{1}{27}.$$

Definition 7.10. *A Non-archimedean valuation is a valuation with the property of sub-additivity replaced by the stronger condition,*

$$v(a + b) \le \max\{v(a), v(b)\}.$$

The p-adic valuation for integers is non-archimedean, while the absolute value for integers is not non-archimedean (it is called “archimedean”).

The Newton iteration algorithm for computing inverses has an analog that can be used under arbitrary valuations.

LEMMA 7.11. *Let v be a non-archimedean valuation. Suppose $v(g) \le 1$, $v(1-gt) \le \epsilon$, and $v(s-(2t-gt^2)) \le \epsilon^2$. Then $v(1-gs) \le \epsilon^2$.*

PROOF.

$$\begin{aligned} v(1-gs) &= v(1-g(s-(2t-gt^2))-g(2t-gt^2)) \\ &\le \max(v(1-g(2t-gt^2)), v(g(s-(2t-gt^2)))). \end{aligned}$$

Now

$$v(g(s-(2t-gt^2))) = v(g)v(s-(2t-gt^2)) \le v(g)\epsilon^2,$$

and

$$v(1-g(2t-gt^2)) = v((1-gt)^2) = v(1-gt)^2 \le \epsilon^2. \quad \square$$

Iteration of the above lemma leads to the following algorithm. Note the analogy to Algorithm 7.4.

ALGORITHM 7.12. *Inversion modulo prime powers using Newton iteration.*
Let R be an integral domain.
Input: *A prime $p \in R$, $g \in R$ with $p \nmid g$, $u_0 \in R$ with $gu_0 \equiv 1 \bmod p$, and $k \in \mathbb{N}$.*
Output: *$u \in R$ satisfying $1-gu \equiv 0 \bmod p^k$.*

1. *Set $t_0 = u_0$, $r = \lceil \log_2 k \rceil$.*
2. *For $i = 1, \ldots, r$ calculate*

$$t_i = (2t_{i-1} - gt_{i-1}^2) \bmod x^{2^i}$$

3. *Return t_r.*

The correctness of the algorithm follows inductively from the above lemma, when $v = v_p$ is the p-adic valuation.

EXAMPLE 7.13. *Let $R = \mathbb{Z}$ and v_3 be the 3-adic valuation. v_3 is a non-archimedean valuation. Suppose $3^{-k} = v_3(b) \ge v_3(a)$. Then 3^k divides a, b and also $a+b$. Thus, if m is the maximum power of 3 that divides $a+b$ then $m \ge k$. So, $v_3(a+b) = 3^{-m} \le 3^{-k} = v_3(b) \le \max\{v_3(a), v_3(b)\}$. Clearly $v_3(a) \le 1$.*

Suppose we wish to compute the inverse of 5 to an accuracy of 1/27. We begin with $t_0 = -1$. Then $v_3(1-5(-1)) = v_3(6) = 1/3$. Using

$$s = 2t_0 - 5t_0^2,$$

we obtain $s = -7$. If we let $t_1 = 2 \equiv s = -7 \bmod 9$, $v_3(t_1 - s) = 1/9$ and $v_3(1-5\cdot 2) = v_3(-9) = 1/9$. Finally, using

$$s = 2t_1 - 5t_1^2$$

we get $s = -259$. Let $t_1 = s \bmod 27 = -16$. Then

$$v_3(1-5\cdot(-16)) = v(81) = 1/81 < 1/27.$$

Corollary 7.14. *Let R be an integral domain, $p \in R$ a prime, and $k \in \mathbb{N}_{>0}$. An element $g \in R$ is invertible modulo p^k if and only if $p \nmid g$.*

Proof. If $p \nmid g$, the Algorithm 7.12 computes an inverse of g modulo p^k. Conversely, let $u \in R$ with $gu \equiv 1 \bmod p^k$, and assume that $p \mid g$. Then $gu + hp^k = 1$ for some $h \in R$, and p divides the right hand side $gu + hp^k$. But p cannot divide 1, and this contradiction proves $p \nmid g$. $\square$

8. Solving polynomial equations modulo prime powers

Problem 8.1. *For a Euclidean domain R, a prime element $p \in R$, $f \in R[y]$, and $n \in \mathbb{N}$, find $h \in R$ with $f(h) \equiv 0 \bmod p^n$.*

As in the case of inversion modulo p^n, Newton iteration will lead to a fast algorithm. But first we need to adapt some well known tools from numerical analysis to our purely algebraic setting: the derivativeand the Taylor expansion of a polynomial.

8.1. Formal derivatives and Taylor expansion.

Definition 8.2. *Let R be an arbitrary ring (commutative, with 1). For $f = \sum_{0\le i\le d} f_i y^i \in R[y]$ we define the* formal derivative *of f by*

$$f' = \sum_{0\le i\le d} i f_i y^{i-1}.$$

For $R = \mathbb{R}$, this is the familiar notion usually defined by a limit process. But in general, say over a finite field, there is no concept like a "limit".

The formal derivative has some familiar properties.

Lemma 8.3. *(i) $'$ is R-linear,*

(ii) $'$ satisfies the Leibniz (or product) rule $(fg)' = f'g + g'f$,

(iii) $'$ satisfies the chain rule $(f(g))' = f'(g)g'$.

Proof.

(i) Let $f, g \in R[y]$, $f = \sum_{0\le i\le d} f_i y^i$, $g = \sum_{0\le i\le d} g_i y^i$, and $a, b \in R$.

$$\begin{aligned} (af+bg)' &= \Big(\sum_{0\le i\le d}(af_i+bg_i)y^i\Big)' = \sum_{0\le i\le d} i(af_i+bg_i)y^{i-1} \\ &= a\sum_{0\le i\le d} i f_i y^{i-1} + b\sum_{0\le i\le d} i g_i y^{i-1} = af' + bg'. \end{aligned}$$

(ii) Because of linearity, it is enough to show the claim for powers of y. So let $n, m \in \mathbb{N}$.

$$\begin{aligned} (y^n y^m)' &= (y^{n+m})' = (n+m)y^{n+m-1} = ny^{n-1}y^m + my^{m-1}y^n \\ &= (y^n)'y^m + (y^m)'y^n. \end{aligned}$$

(iii) Again, it is sufficient to show the claim for f being a power of y, $f = y^n$ for $n \in \mathbb{N}$ say. But then the claim reduces to $(g^n)' = ng^{n-1}g'$, which is easily proven using the Leibniz rule and induction on n.

Note one difference from the usual derivatives, say over $\mathbb{R}$. Over $\mathbb{F}_p$ (or, more generally, any field of characteristic $p > 0$) any pth derivative is zero. For example, $f'' = 0$ for all $f \in \mathbb{F}_2[y]$.

LEMMA 8.4. *(Formal Taylor expansion) Let $f \in R[y]$ as above and $a \in R$. Then*

$$f(a+y) = f(a) + f'(a)y + cy^2$$

for some $c \in R[y]$.

PROOF. Sorting $g = f(a+y)$ by powers of y, we get $g = \sum_{0 \le i \le n} g_i y^i$ for some $n \in \mathbb{N}$ and all $g_i \in R$. Substituting 0 for y yields $f(a) = g(0) = g_0$. Now we take derivatives:

$$f'(a+y) = (f(a+y))' = g' = \sum_{0 \le i \le n} i g_i y^{i-1}.$$

By again substituting 0 for y, we get $f'(a) = g'(0) = g_1$, and if we now let $c = \sum_{2 \le i \le n} g_i y^{i-2}$, the claim follows. □

This is called the Taylor expansion of f around a. In fact, we can even consider a to be a new indeterminate, and then the lemma is true with $c \in R[y, a]$. It says that $f(a) + f'(a)y$ is an approximation to f near a:

$$f(a+y) \equiv f(a) + f'(a)y \bmod y^2,$$

or equivalently, in terms of the y-adic valuation v_y on $R[y]$:

$$v_y(f(a+y) - f(a) - f'(a)y) \le \frac{1}{4}.$$

8.2. p-adic Newton iteration. Now we are ready to state the main result for the Newton iteration algorithm. R will be a Euclidean domain in the sequel.

LEMMA 8.5. *(Quadratic convergence of Newton iteration) Let $p \in R$ be prime, $f \in R[y]$, $u, w \in R$ with $f(u) \equiv 0 \bmod p^l$ for some $l \in \mathbb{N}_{>0}$ and $f'(u) \not\equiv 0 \bmod p$, and suppose that Newton's formula holds "approximately":*

$$w \equiv u - f(u)f'(u)^{-1} \bmod p^{2l}. \tag{8.1}$$

Then $w \equiv u \bmod p^l$, $f(w) \equiv 0 \bmod p^{2l}$, and $f'(w) \not\equiv 0 \bmod p$. Intuitively, if u is a "good" approximation to a zero of f, then w is a "better" approximation, at least "twice as good".

PROOF. First we have to ensure that w is well defined, i.e., that $f'(u)$ is invertible modulo p^{2l}. The assumption that $f'(u) \not\equiv 0 \bmod p$ means that $p \nmid f'(u)$, and hence $f'(u)$ is invertible modulo any power of p, by Corollary 7.14. Algorithm 7.12 gives a method for computing $f'(u)^{-1} \bmod p^{2l}$.

Since p^l divides p^{2l}, the congruence (8.1) also holds modulo p^l, and

$$w \equiv u - f(u)f'(u)^{-1} \equiv u \bmod p^l$$

because $f(u)$ vanishes modulo p^l. This proves the first assertion.

For the second one, we make use of the Taylor expansion given by Lemma 8.4 of f around u by substituting $w-u$ for y:

$$\begin{aligned} f(w) &= f(u) + f'(u)(w-u) + c(w-u)\cdot(w-u)^2 \\ &\equiv f(u) + f'(u)(w-u) \bmod p^{2l} \\ &\equiv 0 \bmod p^{2l} \end{aligned}$$

Here, we use that $p^{2l} \mid (w-u)^2$ by the first assertion, and $c(w-u)$ is, of course, c with $w-u$ substituted for y.

The last assertion is a simple consequence of the first one: since $w \equiv u \bmod p^l$, we have $w \equiv u \bmod p$, which in turn implies $g(w) \equiv g(u) \bmod p$ for *any* $g \in R[y]$, in particular for $g = f'$. This is just a special case of a general principle: Since the map $x \mapsto x \bmod p$ is a ring homomorphism, it commutes with the ring operations $+$ and $\cdot$, and hence with any function on R defined by a polynomial expression. □

In the language of the p-adic valuation v_p, the lemma says that if $v_p(f(v)) \le \epsilon < 1$ and $v_p(f'(v)) = 1$, then $v_p(f(w)) \le \epsilon^2$. A similar, though slightly weaker, statement is true for the usual Newton iteration over $\mathbb{R}$, with the absolute value instead of v_p.

The p-adic Newton iteration algorithm is based on Lemma 8.5. Note the similarities to the Newton iteration algorithm for inversion.

ALGORITHM 8.6. *p-adic Newton iteration.*
Input: $f \in R[y]$, $n \in \mathbb{N}_{>0}$, $h_0 \in R$ *a starting solution, with* $f(h_0) \equiv 0 \bmod p$ *and* $f'(h_0) \not\equiv 0 \bmod p$.
Output: $h \in R$ *with* $f(h) \equiv 0 \bmod p^n$.

1. *Set* $k = \lceil \log_2 n \rceil$.
2. *For* $1 \le i \le k$ *compute* h_i *with* $h_i \equiv h_{i-1} - f(h_{i-1})f'(h_{i-1})^{-1} \bmod p^{2^i}$, *using the Newton iteration of Algorithm 7.12 for the inversion of* $f'(h_{i-1})$ *modulo* p^{2^i}.
3. *Return* $h = h_k$.

THEOREM 8.7. *Algorithm 8.6 works correctly.*

PROOF. We show the invariants

1. $h_i \equiv h_0 \bmod p$,
2. $f(h_i) \equiv 0 \bmod p^{2^i}$,
3. $f'(h_i) \not\equiv 0 \bmod p$

by induction on i. The case $i = 0$ is clear, the induction step follows by one application of Lemma 8.5 with $v = h_{i-1}$, $w = h_i$, and $l = 2^{i-1}$. Since p^n divides p^{2^k}, we conclude that $f(h) = f(h_k) \equiv 0 \bmod p^n$. □

EXERCISE 8.8. *Under which condition does the above algorithm work for a rational function $f \in R(y)$? Show that the Newton formula for $f = \frac{1}{y} - a \in R(y)$ gives exactly the inversion procedure of section 7. Why does the polynomial $f = ay - 1 \in R[y]$ not work directly?*

EXAMPLE 8.9. *1. $R = \mathbb{Z}$ and $p = 5$. Determine a nontrivial solution (i.e., $\neq \pm 1$) of the equation $y^4 \equiv 1 \bmod 625$, so $f = y^4 - 1$. For a starting solution we can use $h_0 = 2$, since $f(2) \equiv 0 \bmod 5$ by Fermat's Little Theorem and $f'(2) = 4 \cdot 2^3 \not\equiv 0 \bmod 5$.*

$$\begin{aligned}
h_1 &\equiv h_0 - f(h_0)f'(h_0)^{-1} \equiv 2 - 15 \cdot 32^{-1} \\
&\equiv 2 - 15 \cdot 7^{-1} \equiv 2 - 15 \cdot (-7) \equiv 7 \bmod 25, \\
h_2 &\equiv h_1 - f(h_1)f'(h_1)^{-1} \equiv 7 - 2400 \cdot 1372^{-1} \\
&\equiv 7 - 525 \cdot 122^{-1} \equiv 2 - 525 \cdot 333 \equiv 182 \bmod 625,
\end{aligned}$$

and indeed $182^4 = 1 + 1755519 \cdot 625$.

2. $R = \mathbb{F}_3[x]$ and $p = x$. Determine a square root h of the polynomial $a = 1 + x$ modulo x^4 that satisfies $h(0) = -1$. Here $f = y^2 - a \in \mathbb{F}_3[x][y]$ and $h_0 = -1$ can serve as a starting solution, since $h_0(0) = -1$, $f(h_0) = -x \equiv 0 \bmod x$, and $f'(h_0) = 2h_0 = 1 \not\equiv 0 \bmod x$.

$$\begin{aligned}
h_1 &\equiv h_0 - f(h_0)f'(h_0)^{-1} \equiv -1 - (-x)1^{-1} \\
&\equiv -1 + x \bmod x^2, \\
h_2 &\equiv h_1 - f(h_1)f'(h_1)^{-1} \equiv -1 + x - x^2(1 - x)^{-1} \\
&\equiv -1 + x - x^2(1 + x + x^2 + x^3) \equiv -1 + x - x^2 - x^3 \bmod x^4,
\end{aligned}$$

and a calculation shows $(-1 + x - x^2 - x^3)^2 = (1 + x) + x^4(-1 - x + x^2)$.

One question that did not come up with the Newton iteration algorithm for inversion is that of uniqueness of the solution. Inverses modulo p^n are unique, but solutions of an arbitrary polynomial equation $f(y) \equiv 0 \bmod p^n$ generally are not, because there may already be up to $\deg g$ many different solutions modulo p. The following theorem implies that for any $n \in \mathbb{N}_{>0}$, every starting solution gives rise to exactly one solution modulo p^n.

THEOREM 8.10. *(Uniqueness of Newton iteration) Let $f \in R[y]$, $u \in R$ with $f(u) \equiv 0 \bmod p$ and $f'(u) \not\equiv 0 \bmod p$ a starting solution, and $n \in \mathbb{N}_{>0}$. If $w, \tilde{w} \in R$ are solutions modulo p^n with $w \equiv u \equiv \tilde{w} \bmod p$ and $f(w) \equiv 0 \equiv f(\tilde{w}) \bmod p^n$, then $w \equiv \tilde{w} \bmod p^n$.*

PROOF. Again, we make use of the Taylor expansion and get

$$f(\tilde{w}) = f(w) + f'(w)(\tilde{w} - w) + c \cdot (\tilde{w} - w)^2$$

for some $c \in R$, or equivalently

$$f(\tilde{w}) - f(w) = (\tilde{w} - w)(f'(w) + c \cdot (\tilde{w} - w)). \tag{8.2}$$

Now

$$\begin{aligned} f'(w) + c \cdot (\tilde{w} - w) &\equiv f'(w) \bmod p \qquad (\text{since } \tilde{w} - w \equiv 0 \bmod p) \\ &\equiv f'(u) \bmod p \qquad (\text{since } w \equiv u \bmod p) \\ &\not\equiv 0 \bmod p. \end{aligned}$$

But p^n divides the left hand side of (8.2) and hence divides $\tilde{w} - w$, i.e., $\tilde{w} \equiv w \bmod p^n$. □

REMARK 8.11. *The statement of Theorem 8.10 need no longer be true if* u *violates the second condition for a starting solution, i.e., if* $f'(u) \equiv 0 \bmod p$*. For example, the congruence* $y^4 \equiv 0$ *has only one solution* $y \equiv 0$ *modulo* 5*, but five solutions* $y \equiv 0, 5, 10, 15, 20$ *modulo* 25 *that are all congruent to* 0 *modulo* 5*. Here* $f = y^4$ *and* $f'(0) \equiv 0 \bmod 5$*, so* 0 *is not a proper starting solution.*

Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik
Skript Computeralgebra I, Teil 4
Wintersemester 1995/96

9. The Euclidean Algorithm

A global goal of this course is to present fast algorithms for integers and polynomials. We have already seen such algorithms for multiplication via the FFT, for division with remainder using Newton iteration, and for the solution of polynomial equations. Integers and polynomials with coefficients in a field behave similarly in many situations. In general, there is *one* algorithm for both kinds of objects, but the algorithm for polynomials is often easier to understand due to the absence of carries. To cover the structural similarity of gcd calculations for integers and polynomials, there is the notion of a Euclidean domain.

9.1. Euclidean domains.

Definition 9.1. *An integral domain R together with a function $d : R\backslash\{0\} \longrightarrow \mathbb{N}$ is called* Euclidean *if the following holds for all $a, b \in R$ with $b \neq 0$:*

(i) $d(ab) \geq d(a)$,

(ii) $\exists q, r \in R : a = qb + r$ *and* $r = 0$ *or* $d(r) < d(b)$ *(division with remainder).*

We also write $q = a \text{ quo } b$ (for quotient*) and $r = a \text{ rem } b$ (for* remainder*), though q and r need not be unique. Such a d is called a* Euclidean function *on R.*

Example 9.2. *(i) $R = \mathbb{Z}$, $d(a) = |a|$. Here the quotient and the remainder can be made unique by the additional requirement $r \geq 0$.*

(ii) $R = F[x]$, F a field, $d(a) = \deg a$. It is an easy exercise to show uniqueness of the quotient and the remainder in this case.

(iii) $R = \mathbb{Z}[i] = \{a+ib : a, b \in \mathbb{Z}\}$, the ring of Gaussian integers, $d(a+ib) = a^2+b^2$.

(iv) R a field, $d(a) = 1$.

(v) Imaginary quadratic fields $\mathbb{Q}(\sqrt{d})$, where $d \in \mathbb{Z}$ with $d < 0$ is squarefree. Then $R = \mathcal{O}_d$, the ring of algebraic integers in $\mathbb{Q}(\sqrt{d})$*, equals*

$$\mathcal{O}_d = \begin{cases} \mathbb{Z}[\sqrt{d}] & \text{if } d \equiv 2, 3 \bmod 4, \\ \mathbb{Z}[\frac{1+\sqrt{d}}{2}] & \text{if } d \equiv 1 \bmod 4. \end{cases}$$

Example 3 is the special case where $d = -1$. *The* norm $N : R \longrightarrow \mathbb{N}$ *is defined by* $N(a) = a\overline{a} = |a|^2 = b^2 + c^2$, *where* $\overline{a}$ *denotes the complex conjugate of* $a = b + ic$, *with* $b, c \in \mathbb{R}$. *In algebraic number theory it is shown that the norm is a Euclidean function on* R *if and only if* $d \in \{-1, -2, -3, -7, -11\}$ *and that these are the only cases where* R *is Euclidean at all. Furthermore,* R *is a unique factorization domain if and only if* R *is Euclidean or* $d \in \{-19, -43, -67, -163\}$. *See Lemmermeyer (1995) for an exhaustive discussion.*

In fact, the proper analogy between $\mathbb{Z}$ and $F[x]$ is as follows. We can take $d(a) = |a|$ on $\mathbb{Z}$ and $d(a) = 2^{\deg a}$ on $F[x]$, with $d(0) = 0$ in both cases. Or, equivalently, we can take $d(a) = \log_e |a|$ on $\mathbb{Z}$ and $d(a) = \deg a$ on $F[x]$, with $d(0) = -\infty$ in both cases. This makes d everywhere defined on R, only, in the latter case, its values now are in $\mathbb{R}_{\geq 0} \cup \{-\infty\}$, and $\{d(a) \colon d(a) < \lambda, a \in R\}$ is finite in both cases for any $\lambda \in \mathbb{R}$. However, since Examples 9.2 (i) and (ii) are the most natural notions, we modify Definition 9.1 by making d everywhere defined on R, allowing $-\infty$ as a value of $d(a)$, and replacing (ii) by

(ii)′ $\exists q, r \in R : a = qb + r$ and $d(r) < d(b)$.

In particular, we define the degree of the zero polynomial to be $-\infty$.

Lemma 9.3. *Let* d *be a Euclidean function on* R. *Then the following hold for all* $a \in R \setminus \{0\}$.

(i) $d(0) < d(1) \leq d(a)$,

(ii) $d(1) = d(a)$ *if and only if* a *is a unit.*

Proof. For $a \neq 0$, we have $d(a) = d(a \cdot 1) \geq d(1)$ by part (i) of the definition of a Euclidean function. Dividing 1 by itself with remainder yields $q, r \in R$ with $1 = q \cdot 1 + r$ and $d(r) < d(1)$, which implies that $r = 0$, by the above. This proves (i).

If $a \in R \setminus \{0\}$ is a unit, then also $d(1) = d(a \cdot a^{-1}) \geq d(a)$, so that $d(1) = d(a)$. For the reverse direction assume that equality holds. Division of 1 by a with remainder yields $1 = q \cdot a + r$ for $q, r \in R$ with $d(r) < d(a)$. But $d(r) < d(a) = d(1)$ implies $r = 0$, by (i), and a is a unit. □

Definition 9.4. *Let* R *be a ring and* $a, b, c \in R$. *Then* c *is a* greatest common divisor of a and b, *written* $c = \gcd(a, b)$, *if*

(i) $c \mid a$ *and* $c \mid b$,

(ii) $d \mid a$ *and* $d \mid b$ *imply* $d \mid c$ *for all* $d \in R$.

Similarly, c *is called* a least common multiple of a and b, *written* $c = \mathrm{lcm}(a, b)$, *if*

(i) $a \mid c$ *and* $b \mid c$,

(ii) $a \mid d$ *and* $b \mid d$ *implies* $c \mid d$ *for all* $d \in R$.

For example, $\gcd(12, 15) = 3$ and $\operatorname{lcm}(12, 15) = 60$ in $\mathbb{Z}$, $\gcd(a, 1) = 1$ and $\gcd(a, 0) = a = \gcd(a, a)$ for any $a \in R$. In general, neither the gcd nor the lcm are unique, but all gcd's of a and b are precisely the associates of one of them, and a similar statement holds for the lcm's. Thus the functional notation $c = \gcd(a, b)$ is somewhat misleading; strictly speaking, we have a ternary relation. However, if $R = \mathbb{Z}$ or $R = F[x]$, we can make the gcd (and the lcm) unique by the additional requirement that $\gcd(a, b) > 0$ for $R = \mathbb{Z}$, and that $\gcd(a, b)$ be monic for $R = F[x]$, respectively, and $\gcd(0, 0) = 0$ in both cases. As an example, for negative $a \in \mathbb{Z}$ we have $\gcd(a, a) = \gcd(a, 0) = -a$.

Greatest common divisors and least common multiples need not exist in an arbitrary ring (we will see an example below). In the following subsection, however, we will prove that a gcd always exists in a Euclidean domain, and as a consequence also an lcm exists.

Exercise 9.5. *Prove the following properties of the* gcd.

(i) $\gcd(a, b) = a \iff a \mid b$,

(ii) $\gcd(a, b) = \gcd(b, a)$ *(commutativity),*

(iii) $\gcd(a, \gcd(b, c)) = \gcd(\gcd(a, b), c))$ *(associativity),*

(iv) $c \cdot \gcd(a, b) = \gcd(c \cdot a, c \cdot b)$ *(distributivity).*

Of course, they are only valid up to associates. Hint: For (iii) and (iv), show that any divisor of the left hand side also divides the right hand side, and vice versa. What are the corresponding statements for the lcm*? Are they also true?*

Because of the associativity, we write $\gcd(a_1, \ldots, a_n)$ or $\gcd\{a_1, \ldots, a_n\}$ for $\gcd(a_1, \gcd(\ldots, a_n) \ldots)$ (likewise for the lcm). The symmetric notation also comes closer to the fact that $\gcd\{a_1, \ldots, a_n\}$ is the greatest common divisor of all elements in the set $\{a_1, \ldots, a_n\}$, regardless of the order, in the sense that it is a common divisor and a multiple of any other common divisor.

9.2. The Extended Euclidean Algorithm.

Algorithm 9.6. *Extended Euclidean Algorithm (EEA)*
Input: $f, g \in R$, *R Euclidean.*
Output: $\gcd(f, g)$ *and* $s, t \in R$ *such that* $sf + tg = \gcd(f, g)$.

1. Set $a_0 = f, \quad s_0 = 1, \quad t_0 = 0,$
$\qquad a_1 = g, \quad s_1 = 0, \quad t_1 = 1.$

2. *For* $1 \le i \le l$*, successively compute* $q_i, a_i, s_i, t_i \in R$ *with* $d(a_{i+1}) < d(a_i)$ *and* $a_l \neq 0$ *such that*

$$\begin{array}{rcl} a_0 & = & q_1 a_1 + a_2, \\ & \vdots & \\ a_{i-1} & = & q_i a_i + a_{i+1}, \qquad s_{i+1} = s_{i-1} - q_i s_i, \qquad t_{i+1} = t_{i-1} - q_i t_i, \\ & \vdots & \\ a_{l-1} & = & q_l a_l. \end{array}$$

   The number l *of division steps is determined by the condition that the next remainder be zero.*

3. *Return* a_l, s_l, t_l*.*

Note that l with the stated properties must exist because the $d(a_i)$ are strictly decreasing; l is called the *Euclidean length* of the pair (f, g).

EXERCISE 9.7. *The* Euclidean representation *of a pair* $(f, g) \in R^2$ *is defined as the list* $(q_1, \ldots, q_l, a_l)$ *of the quotients of the Euclidean Scheme together with the* gcd*. Show that for* $R = F[x]$ *the map*

$$\begin{array}{rcl} R^2 & \longrightarrow & \bigcup_{i \ge 1} R^i \\ (f, g) & \mapsto & (q_1, \ldots, q_l, a_l) \end{array}$$

is a bijection.

LEMMA 9.8. *For* $0 \le i \le l$*, we have*

(i) $\gcd(f, g) = \gcd(a_i, a_{i+1})$,

(ii) $s_i f + t_i g = a_i$,

with the convention that $a_{l+1} = 0$*.*

PROOF. For (i) we proceed by induction on i. The case $i = 0$ is clear, so we may assume $i \ge 1$. It is sufficient to show $\gcd(a_{i-1}, a_i) = \gcd(a_i, a_{i+1})$. This follows from $a_{i-1} = a_i q_i + a_{i+1}$ because then for all $d \in R$ we have that $d \mid a_i$ implies $d \mid a_{i-1} \Leftrightarrow d \mid a_{i+1}$.

The second statement is also proven using induction on i. The initial cases i equal to 0 or 1 are a consequence of the definitions. For $i \ge 2$, we calculate

$$\begin{array}{rcl} s_{i+1} f + t_{i+1} g & = & (s_{i-1} - q_i s_i) f + (t_{i-1} - q_i t_i) g \\ & = & (s_{i-1} f + t_{i-1} g) - q_i (s_i f + t_i g) \\ & = & a_{i-1} - q_i a_i = a_{i+1}, \end{array}$$

where the induction hypothesis was used for the first equality in the last line. □

Recall that a nonzero nonunit p in an integral domain R is called *reducible* if there are nonunits $a, b \in R$ such that $p = ab$, otherwise p is called *irreducible*. Units are neither reducible nor irreducible. R is a *unique factorization domain* (UFD) (sometimes also called a *factorial* ring) if every nonzero nonunit $a \in R$ can be written as a product of irreducible elements in a unique way up to reordering and multiplication by units. For example, $R = \mathbb{Z}$ is a UFD and $12 = 2{\cdot}2{\cdot}3 = (-3){\cdot}2{\cdot}(-2)$ are both decompositions of 12 into irreducibles. A nonzero nonunit $p \in R$ is called *prime* if $p \mid (ab)$ implies $p \mid a$ or $p \mid b$ for all $a, b \in R$. In arbitrary integral domains, "prime" and "irreducible" are not necessarily equivalent, but the following lemma says that this is true for Euclidean domains.

Lemma 9.9. *Let R be an integral domain, and $p \in R$.*

(i) If p is prime, then p is irreducible.

(ii) If any two nonzero elements of R have a gcd and p is irreducible, then p is prime.

Proof.

(i) Let $p \in R$ be prime and $p = ab$ be a decomposition of p into two factors. We have to show that one of them is a unit. Certainly $p \mid ab$, and thus $p \mid a$ or $p \mid b$. We may assume that $p \mid a$ without loss of generality, say $pu = a$ with $u \in R$. Then $pub = ab = p$, and $ub = 1$, since R is integral. Hence b is a unit.

(ii) We assume that $p \in R$ is irreducible, $p \nmid a$ and $p \nmid b$. We have to show that $p \nmid ab$. The gcd of p and a is a divisor of p and hence associate to p or a unit by the irreducibility of p. But $\gcd(p, a) = p$ would imply $p \mid a$, and so $\gcd(p, a) = 1$. Similarly, $\gcd(p, b) = 1$. Now the properties of the gcd imply

$$\begin{aligned} \gcd(p, ab) &= \gcd(\gcd(p, pb), ab) = \gcd(p, \gcd(pb, ab)) = \gcd(p, \gcd(p, a)b) \\ &= \gcd(p, b) = 1, \end{aligned}$$

and the claim follows. □

Theorem 9.10. *Any Euclidean domain is a UFD.*

Proof. We show the existence of a decomposition of a nonunit $a \in R\backslash\{0\}$ into irreducibles by induction on $d(a)$. This is clear if a is irreducible itself, so that for the induction step, it is sufficient to prove the claim that if $a = bc$ with nonunits $b, c \in R \setminus \{0\}$, then $d(b) < d(a)$.

Division of b by a with remainder yields $b = qa + r$ for $q, r \in R$ with $d(r) < d(a)$. If $r = 0$, then $b = qa = qcb$ and hence $1 = qc$ since R is integral. But this is a contradiction, since we assumed c to be a nonuit. So r is nonzero, and

$$d(a) > d(r) = d(b - qa) = d(b - qcb) = d(b(1 - qc)) \geq d(b),$$

which proves the above claim.

For the uniqueness, let $r, s \in \mathbb{N}$ and $a = p_1 \cdots p_r = q_1 \cdots q_s$ be two decompositions of a into irreducibles. We prove by induction on r that $r = s$ and the q_i can be reordered in such a way that $p_i = u_i q_i$ for some unit $u_i \in R$ (p_i and q_i are associate) for $1 \leq i \leq r$. If a is irreducible, then this is obviously true, so we assume that a is reducible.

The irreducible p_r divides the product $q_1 \cdots q_s$ and hence $p_r \mid q_j$ for some j by repeated application of Lemma 9.9. By reordering if necessary, we may assume that $j = s$. So $q_s = u_r p_r$ for some $u_r \in R$. But q_s is also irreducible, and hence u_r is a unit. After cancelling $u_r p_r$ from both decompositions, we see that $p_1' p_2 \cdots p_{r-1} = q_1 \cdots q_{s-1}$ are two decompositions of $\frac{a}{u_r p_r} \in R$ into irreducibles, where $p_1' = p_1 u_r^{-1}$. The length of the first decomposition is one less than r, and the induction hypothesis applies: $r-1 = s-1$, and after reordering, we have $p_1' = u_1' q_1$ and $p_i = u_i q_i$ for $2 \leq i \leq r-1$, where $u_1', u_2, \ldots, u_{r-1}$ are units. If we now let $u_1 = u_1' u_r$, we finally have $p_i = u_i q_i$ for $1 \leq i \leq r$. This proves the stated uniqueness. □

EXERCISE 9.11. *Show that each two nonzero elements a, b of a factorial ring R have a* gcd *as well as a* lcm. *Hint: First look at the special case $R = \mathbb{Z}$, use the factorizations of a and b into irreducibles and remember that multiplication by units does not essentially change the situation. Prove that* $\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b$, *and conclude that* $\operatorname{lcm}(a_1, \ldots, a_n) = a_1 \cdots a_n$ *for any n nonzero relatively prime elements $a_1, \ldots, a_n \in R$ (you might need the proof of Lemma 9.9). Is* $\gcd(a_1, \ldots, a_n) \cdot \operatorname{lcm}(a_1, \ldots, a_n) = a_1 \cdots a_n$ *valid for arbitrary $n \in \mathbb{N}$?*

The reverse of the above theorem is false in general. For example, the polynomial ring $F[x_1, \ldots, x_n]$ over a field F is a UFD, and it is Euclidean if and only if $n = 1$ (Exercise: Show that $R = F[x, y]$ is not Euclidean).

There are also examples of integral domains that are not even factorial. Consider $R = \mathcal{O}_{-5} = \mathbb{Z} + \mathbb{Z}\sqrt{-5}$, the ring of algebraic integers of $\mathbb{Q}(\sqrt{-5})$. Then

$$(1 + \sqrt{-5}) \cdot (1 - \sqrt{-5}) = 6 = 2 \cdot 3$$

are two essentially different decompositions of 6 into irreducibles. To see this, we first prove that $2, 3$, and $1 \pm \sqrt{-5}$ are all irreducible. We use the norm $N(a) = a\overline{a} = |a|^2$, where $\overline{a}$ is the complex conjugate of a.

Let us assume $1 + \sqrt{-5} = bc$ for some $b, c \in R$. By the multiplicativity of the norm, we have $6 = N(1 + \sqrt{-5}) = N(b)N(c)$. Is is easily shown that the units of R are exactly the elements of norm 1. But $N(\alpha + \beta\sqrt{-5}) = \alpha^2 + 5\beta^2$ is congruent to $0, 1$, or 4 modulo 5 for any $\alpha + \beta\sqrt{-5} \in R$, so $N(b) \in \{2, 3\}$ is impossible, and either b or c is a unit, i.e., $1 + \sqrt{-5}$ is irreducible. The proof for $1 - \sqrt{-5}, 2$ and 3 goes analogously.

It remains to show that none of $1 \pm \sqrt{-5}$ is associate to 2 or 3. But $N(2) = 4$ and $N(3) = 9$ are both different from $N(1 \pm \sqrt{-5}) = 6$, and associate elements have the same norm. This also shows that irreducibles are not necessarily prime in arbitrary integral domains: 2 is irreducible and divides $(1 + \sqrt{-5})(1 - \sqrt{-5})$ in $R = \mathcal{O}_{-5}$, but divides none of the factors; hence 2 is not prime.

EXERCISE 9.12. *Show that* 6 *and* $2 + 2\sqrt{-5}$ *have no* gcd *in* $\mathcal{O}_{-5}$.

The following theorem, which we will not prove here, summarizes some properties of integral domains that are equivalent to being factorial.

THEOREM 9.13. *For an integral domain R, the following are equivalent.*

(i) R is a UFD.

(ii) Any nonzero nonunit in R can be written as a product of primes.

(iii) Any nonzero nonunit in R can be written as a product of irreducibles, and any irreducible in R is prime.

(iv) Any nonzero nonunit in R can be written as a product of irreducibles, and any two nonzero elements of R have a gcd *in R.*

The following theorem gives an important application of the Extended Euclidean Algorithm.

THEOREM 9.14. *Let R be a Euclidean domain, $a, m \in R$, and $S = R/mR$. Then* $a \bmod m \in S$ *is a unit if and only if* $\gcd(a, m) = 1$. *In this case, the modular inverse can be computed by means of the Extended Euclidean Algorithm.*

PROOF.

$$\begin{aligned} a \text{ is invertible modulo } m &\iff \exists s \in R \quad sa \equiv 1 \bmod m \\ &\iff \exists s, t \in R \quad sa + tm = 1 \implies \gcd(a, m) = 1. \end{aligned}$$

If on the other hand $\gcd(a, m) = 1$, the Extended Euclidean Algorithm will give such $s, t \in R$. □

EXAMPLE 9.15. $R = \mathbb{Z}$, $m = 29$, $a = 12$. *Obviously* $\gcd(a, m) = 1$, *and the EEA computes* $5 \cdot 29 + (-12) \cdot 12 = 1$, *i.e.* $(-12) \cdot 12 \equiv 17 \cdot 12 \equiv 1 \bmod 29$, *and* 17 *is the inverse of* 12 *modulo* 29.

An important application of this is the computational handling of arithmetic in $R = \mathbb{Z}/n\mathbb{Z}$, in particular for n prime, and in $R = F[x]/(f)$, where F is a field and $f \in F[x]$, in particular for f irreducible. In the prime respectively irreducible case, any $a \in R \setminus \{0\}$ is a unit, i.e., R is a field. Modular addition, subtraction, and multiplication are quite straightforward, and the Extended Euclidean Algorithm enables us to compute modular divisions.

9.3. Cost analysis. Consider the (Extended) Euclidean Scheme for $f, g \in R$ as in Algorithm 9.6 with $n = d(f) \geq d(g) = m \geq 0$. The number l of division steps is obviously bounded by $l \leq d(g) + 1$.

We now investigate the two important cases $R = F[x]$ and $R = \mathbb{Z}$ separately, starting with $R = F[x]$, where F is a field, and $d(a) = \deg a$ as usual.

First, we consider dividing a polynomial $a \in F[x]$ by $b \in F[x]$ with remainder. The following algorithm formalizes the familiar "school method".

ALGORITHM 9.16. *Polynomial division with remainder*
Input: $a = \sum_{0 \leq i \leq k} a_i x^i, b = \sum_{0 \leq i \leq h} b_i x^i \in F[x]$, *with each* $a_i, b_i \in F$, $b_h \in F^\times$, *and* $k \geq h \geq 0$, *where* F *is a ring.*
Output: $q, r \in F[x]$ *with* $a = qb + r$ *and* $\deg r < h$.

1. *Set* $r = a$.
2. *Repeat steps 3 and 4 for* $i = k - h, k - h - 1, \ldots, 0$.
3. *Set* $q_i = r_{h+i}/b_h$, *where* r_{h+i} *is the* $(h+i)$*th coefficient of* r.
4. *Replace* r *by* $r - q_i x^i b$.
5. *Return* $\sum_{0 \leq i \leq k-h} q_i x^i$ *and* r.

The cost for one iteration of the loop in step 2 is one division, h multiplications and h additions in F (note that the $(h+i)$th coefficient of r becomes 0 in step 4 and hence need not be computed). Together, we get a cost of

$$(k - h + 1)(2h + 1) \leq 2(k - h + 1)(h + 1) = 2(\deg q + 1)(\deg b + 1) \in O(k^2)$$

operations in F.

We have already seen a faster $O(k \log k \operatorname{loglog} k)$ division algorithm in section 7; but this algorithm is only useful when $\deg q = k - h$ is reasonably large in comparison to $\deg b = h$, while in the Euclidean Algorithm, most of the quotients have degree 1 for a random polynomial. There is, however, a faster gcd algorithm which uses fast division, but we will not discuss it in this course.

Let $n_i = \deg a_i$ for $1 \leq i \leq l + 1$, where we let $a_{l+1} = 0$. Then $n_0 = n$, $n_1 = m$, the n_i are strictly decreasing for $i \geq 1$, and $\deg q_i = n_{i-1} - n_i$ for $1 \leq i \leq l$. First we calculate the total cost for the Euclidean Algorithm, i.e., for only computing the a_i and q_i including the gcd of f and g, as follows.

$$\begin{aligned}\sum_{1 \leq i \leq l} 2(n_{i-1} - n_i + 1)(n_i + 1) &\leq 2 \sum_{1 \leq i \leq l} (n_{i-1} - n_i)(m + 1) + 2 \sum_{1 \leq i \leq l} (m + 1) \\ &= 2(n_0 - n_l + l)(m + 1) \leq 2(n + m + 1)(m + 1) \in O(nm).\end{aligned}$$

It remains to determine the cost for computing s_i, t_i on the way.

LEMMA 9.17.

$$\begin{aligned}\deg s_i &= \sum_{2\le j<i} \deg q_j = n_1 - n_{i-1} \textit{ for } 2 \le i \le l, &(9.1)\\ \deg t_i &= \sum_{1\le j<i} \deg q_j = n_0 - n_{i-1} \textit{ for } 1 \le i \le l.\end{aligned}$$

PROOF. We only prove the first equality; the second is shown in the same way and left as an exercise. We prove (9.1) and

$$\deg s_{i-1} < \deg s_i \text{ for } 2 \le i \le l \tag{9.2}$$

by simultaneous induction on i. For $i = 2$, we have $s_2 = s_0 - q_1 s_1 = 1 - q_1 \cdot 0 = 1$ independently of q_1, and $\deg s_1 = -\infty < 0 = \deg s_2$. Now we assume that $i \ge 2$. Then, by the induction hypothesis (9.2), we have

$$\deg s_{i-1} < \deg s_i < n_{i-1} - n_i + \deg s_i = \deg(q_i s_i),$$

which implies that

$$\deg s_{i+1} = \deg(s_{i-1} - q_i s_i) = \deg q_i + \deg s_i > \deg s_i,$$

and

$$\deg s_{i+1} = \deg q_i + \deg s_i = \sum_{2\le j<i} \deg q_j + \deg q_i = \sum_{2\le j<i+1} \deg q_j,$$

where we used the induction hypothesis (9.1). □

THEOREM 9.18. *The Extended Euclidean Algorithm for polynomials in $F[x]$ of degree $\le n$ can be performed with $O(n^2)$ operations in F.*

PROOF. We only need to show that all the s_i, t_i can be computed within the stated bounds. In each step, the computation of $t_{i+1} = t_{i-1} - q_i t_i$ requires one multiplication of two polynomials of degree $n_{i-1} - n_i$ and $n_0 - n_{i-1}$, respectively, plus one addition of polynomials of degree at most $n_0 - n_i$, by Lemma 9.17. With the naive estimate $2(\deg a + 1)(\deg b + 1)$ for the multiplication cost for polynomials a and b, we get

$$\begin{aligned}&\sum_{1\le i<l} (2(n_{i-1} - n_i + 1)(n_0 - n_{i-1} + 1) + n_0 - n_i + 1)\\ &\quad\le\; 2\sum_{1\le i<l} (n_{i-1} - n_i)(n+1) + 3\sum_{1\le i<l} (n+1)\\ &\quad=\; 2(n_0 - n_{l-1})(n+1) + 3(l-1)(n+1) \le (2n + 3m)(n+1)\\ &\quad\in\; O(n^2).\end{aligned}$$

The same bound works for the s_i's. □

Now we sketch the cost analysis when $R = \mathbb{Z}$ and $d(a) = |a|$. We think of all numbers represented in binary with leading bit equal to 1. Then the *binary length* $\lambda(a)$ of a positive integer a is the number of bits in the binary representation, i.e., $\lambda(a) = \lfloor \log_2 a \rfloor + 1$. Here the bound $l \le d(g)+1 = |g|+1 \le 2^{\log g+1} \le 2 \cdot 2^{\lambda(g)}$ on the length of the Euclidean Scheme for the pair $(f, g) \in \mathbb{N}^2$ is exponential in the input size $\lambda(f) + \lambda(g)$ and hence rather useless. We can in fact prove a polynomial upper bound for l: We may assume that $a_i > 0$ for $0 \le i \le l$ and $q_i > 0$ for $1 \le i \le l$. Then we have

$$a_{i-1} = q_i a_i + a_{i+1} > q_i a_{i+1} + a_{i+1} = (q_i + 1)a_{i+1} \ge 2a_{i+1}$$

for $1 \le i < l$. Then

$$\prod_{1 \le i < l} a_{i-1} > 2^{l-1} \prod_{1 \le i < l} a_{i+1},$$

and $a_0 \ge a_1$ implies

$$2^{l-1} < \frac{a_0 a_1}{a_l a_{l-1}} < a_0^2,$$

or $l \le 1 + 2\log_2 a_0 = 1 + 2\log_2 f \le 1 + 2\lambda(f)$.

This bound can still be improved. For $N \in \mathbb{N}$ and $f, g \in \mathbb{Z}$ with $N \ge f > g > 0$, the largest possible Euclidean length l of (f, g) is the one where all the quotients are equal to 1, i.e., f and g are the two largest successive Fibonacci numbers less or equal to N. As an example, here is the Euclidean Scheme for $(f, g) = (13, 8)$:

$$\begin{aligned} 13 &= 1 \cdot 8 + 5, \\ 8 &= 1 \cdot 5 + 3, \\ 5 &= 1 \cdot 3 + 2, \\ 3 &= 1 \cdot 2 + 1, \\ 2 &= 2 \cdot 1. \end{aligned}$$

Since the nth Fibonacci number F_n (with $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \ge 2$) is approximately $\phi^n/\sqrt{5}$, where $\phi = (1+\sqrt{5})/2 = 1.618...$ is the golden ratio, the following holds for the Euclidean length l of $(f, g) = (F_{n+1}, F_n)$:

$$l = n - 1 \approx \log_\phi \sqrt{5} f - 2 \in 1.441 \log_2 f + O(1)$$

(see Knuth 1981, §4.5.3). Knuth also shows that the *average* length of the Euclidean Scheme is

$$l \approx (12(\log_e 2)^2/\pi^2) \log_2 f \approx 0.584 \log_2 f.$$

Now that we have a good upper bound for the number of steps in the Euclidean Scheme, we look at the cost for each step. The cost measure widely used for algorithms dealing with integers is the number of bit operations which can be rigorously defined as the number of steps of a Turing or register machine (RAM) or the number of gates of a Boolean circuit implementing the algorithm. We give only informal arguments when proving bounds for the required number of bit operations here, since the details of those computational models are rather technical.

First consider the cost for one division step. Let $a > b > 0$ be integers and $a = qb + r$ with $q, r \in \mathbb{N}$ and $0 \le r < b$. Then the number of steps for computing q and r using ordinary long division is one more than the difference of the binary lengths of a and b and hence at most $\log_2 a - \log_2 b + 1$. In each step, a shifted divisor $2^i b$ has to be subtracted from some integer of the same length, and this can be done with at most $c(\log_2 b + 1)$ bit operations for some $c \in \mathbb{N}$. We get a total cost of no more than $c(\log_2 a - \log_2 b + 1)(\log_2 b + 1)$ bit operations. Note the analogy to long division of polynomials; the only difference are the carries during subtraction.

Then with $n = \lambda(f)$ and $m = \lambda(g)$, we can prove that the total cost for performing the Euclidean Algorithm (without computing the s_i and t_i) is $O(nm)$ bit operations. The proof is similar as in the polynomial case and therefore omitted. Furthermore, with analogous bounds for the length of s_i and t_i as in the polynomial case, we have the following theorem.

THEOREM 9.19. *The Extended Euclidean Algorithm for integers of binary length at most n bits can be performed with $O(n^2)$ bit operations.*

9.4. Continued fractions. Let R be a Euclidean domain, $a_0, a_1 \in R$ and $q_i, a_i \in R$ for $1 \le i \le l$ the entries of the Euclidean Scheme for a_0, a_1. Then

$$\begin{aligned}
\frac{a_0}{a_1} &= q_1 + \frac{a_2}{a_1} = q_1 + \cfrac{1}{\cfrac{a_1}{a_2}} \\
&= q_1 + \cfrac{1}{q_2 + \cfrac{a_3}{a_2}} = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\cfrac{a_2}{a_3}}} \\
&= q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{a_4}{a_3}}} \\
&= q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cfrac{1}{\ddots + \cfrac{1}{q_l}}}} .
\end{aligned}$$

This is called the *continued fraction expansion* of $\frac{a_0}{a_1} \in \mathrm{Quot}(R)$, the field of fractions of R. In general, arbitrary elements of R may occur in the relative numerators of a continued fraction, but when all of them are required to be 1 as above, the representation of $\frac{a_0}{a_1}$ by a continued fraction is unique and obviously computed by the Euclidean Algorithm . For abbreviation, we write $[q_1, \ldots, q_l]$ for the continued fraction $q_1 + 1/(q_2 + 1/(\cdots + 1/q_l)\cdots)$.

If $R = \mathbb{Z}$ or $R = F[x]$, then even an element r of an extension field F of $\mathrm{Quot}(R)$ may be represented by an (infinite) continued fraction in the sense that its initial segments converge to r with regard to the absolute value and the x-adic valuation, respectively.

$r \in \mathbb{R}$	continued fraction expansion of r
$\frac{8}{29}$	$[0, 3, 1, 1, 1, 2]$
$\sqrt{\frac{8}{29}}$	$[0, 1, 1, 9, 2, 2, 3, 2, 2, 9, 1, 2, \overline{1, 9, 2, 2, 3, 2, 2, 9, 1, 2}, \ldots]$
$\sqrt{3}$	$[1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, \overline{1, 2}, \ldots]$
$\sqrt[3]{2}$	$[1, 3, 1, 5, 1, 1, 4, 1, 1, 8, 1, 14, 1, 10, 2, 1, 4, 12, 2, 3, 2, \ldots]$
π	$[3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, \ldots]$
$e = \exp(1)$	$[2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, \ldots]$
ϕ	$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, \ldots]$

Figure 9.1: Examples of continued fraction representations of real numbers.

The examples of continued fraction representations of real numbers shown in Figure 9.1 are from Knuth (1981). The overlined parts indicate the period of ultimately periodic sequences. It can be shown that the continued fraction expansion of an irrational number r is ultimately periodic if and only if r is a zero of an irreducible quadratic polynomial with integer coefficients, i.e., r lies in a quadratic number field.

There is a rich and interesting theory of continued fractions; for further reading and references see also Knuth (1981).

i	$[q_1, \ldots, q_i]$	decimal expansion	accuracy
1	3	3.00000000000000000000	1 digit
2	$\frac{22}{7}$	3.14285714285714285714	3 digits
3	$\frac{333}{106}$	3.14150943396226415094	5 digits
4	$\frac{355}{113}$	3.14159292035398230088	7 digits
5	$\frac{103993}{33102}$	3.14159265301190260407	10 digits

Figure 9.2: Rational approximations of π

The continued fraction expansion of an irrational number $r \in \mathbb{R}$ is an excellent tool to approximate r by rational numbers with "small" denominator. This is called *Diophantine approximation*. Figure 9.2 shows the rational approximations of π that result from cutting the continued fraction expansion after the ith component for $i = 1, \ldots, 5$.

Historically, the ancient Babylonians and Egyptians already used $3\frac{1}{8}$ and $3\frac{13}{81}$, respectively, as rational approximation for π. In 1771, Lambert proved that π is irrational (see Beckmann 1977). The first transcendence proof for π is due to Lindemann (1882). There are many open questions about the decimal expansion of π. Its computation is only possible with the help of fast algorithms for high precision integer and floating point arithmetic, based on the FFT, and is a good test for computer hardware, which is routinely performed on supercomputers. Borwein *et al.* (1989) speak from experience:

> *A large-scale computation of π is entirely unforgiving; it soaks into all parts of the machine and a single bit awry leaves detectable consequences.*

The table below shows the historical development in computing the decimal expansion of π.

Archimedes (287-212 B.C.)	$3\frac{10}{71} < \pi < 3\frac{1}{7}$
Ludolph van Ceulen (1540-1610)	34 digits
William Shanks (1853)	526 digits
Metropolis, Reitwieser, von Neumann (1949) on ENIAC	2039 digits
Kanada (1988)	$\approx 2 \cdot 10^{11}$ digits

A funny approximation of π is from Borwein & Borwein (1992). They show that the real number

$$\frac{1}{10^5}\left(\sum_{n\in\mathbb{Z}} e^{-n^2 10^{-10}}\right)^2$$

is different from π but approximates π to $4.2 \cdot 10^{10}$ digits.

Shanks published a book on his computation of 707 digits, but made an error at the 527th digit. With a modern computer algebra system, the first 1000 digits just require a few keystrokes, and bang! there is it on your screen.

10. The Chinese Remainder Algorithm

10.1. The Chinese Remainder Theorem. For this section, R is a Euclidean domain,

$$m_1, \ldots, m_r \in R \text{ are pairwise relatively prime, i.e., } \gcd(m_i, m_j) = 1 \text{ for } 1 \le i < j \le r, \text{ and } m = m_1 \cdots m_r. \tag{10.1}$$

For $1 \le i \le r$, we have the natural ring homomorphism

$$\begin{array}{rcl} \pi_i \colon R & \longrightarrow & R/(m_i), \\ a & \longmapsto & a \bmod m_i. \end{array}$$

Combining these for all i, we get the ring homomorphism

$$\begin{array}{rcl} \chi = \pi_1 \times \cdots \times \pi_r \colon R & \longrightarrow & R/(m_1) \times \ldots \times R/(m_r), \\ a & \longmapsto & (a \bmod m_1, \ldots, a \bmod m_r). \end{array}$$

THEOREM 10.1. *(i)* $\ker \chi = (m)$,

(ii) χ *is surjective.*

PROOF. Let $a \in R$. Then

$$\begin{array}{rcl} a \in \ker \chi & \Longleftrightarrow & \chi(a) = (a \bmod m_1, \ldots, a \bmod m_r) = (0, \ldots, 0) \\ & \Longleftrightarrow & m_i \mid a \text{ for } 1 \le i \le r \\ & \Longleftrightarrow & \operatorname{lcm}(m_1, \ldots, m_r) \mid a \\ & \Longleftrightarrow & m \mid a. \end{array}$$

For the second part, it is sufficient to show that an $l_i \in R$ with $\chi(l_i) = e_i$ exists for $1 \le i \le r$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0) \in R/(m_1) \times \cdots \times R/(m_r)$ denotes the ith unit vector. To see why this is enough, let $b = (b_1 \bmod m_1, \ldots, b_r \bmod m_r) \in R/(m_1) \times \cdots \times R/(m_r)$ be arbitrary, with $b_1, \ldots, b_r \in R$. Then

$$\begin{array}{rcl} \chi\left(\displaystyle\sum_{1 \le i \le r} b_i l_i\right) & = & \displaystyle\sum_{1 \le i \le r} \chi(b_i)\chi(l_i) \\ & = & \displaystyle\sum_{1 \le i \le r} (b_i \bmod m_1, \ldots, b_i \bmod m_r) \cdot e_i \\ & = & \displaystyle\sum_{1 \le i \le r} (0, \ldots, 0, b_i \bmod m_i, 0, \ldots, 0) \\ & = & b. \end{array}$$

We assume $i = 1$ for simplicity. The Extended Euclidean Algorithm, when applied to $m_2 \cdots m_r = \frac{m}{m_1}$ and m_1, computes $s, t \in R$ with $s\frac{m}{m_1} + tm_1 = 1$. If we now let $l_1 = s\frac{m}{m_1}$, then obviously $l_1 \equiv 0 \bmod m_j$ for $2 \le j \le r$, and

$$l_1 = s\frac{m}{m_1} \equiv s\frac{m}{m_1} + tm_1 = 1 \bmod m_1,$$

so that $\chi(l_1) = e_1$ as claimed. □

Using the above theorem and the homomorphism theorem for rings, we have the following corollary.

Corollary 10.2. *(Chinese Remainder Theorem, CRT)*

$$R/(m) \cong R/(m_1) \times \cdots \times R/(m_r).$$

Algorithm 10.3. *Chinese Remainder Algorithm (CRA)*
Input: *$m_1, \ldots, m_r \in R$ pairwise relatively prime, $b_1, \ldots, b_r \in R$, where R is a Euclidean domain.*
Output: *$c \in R$ which solves the system of congruences*

$$c \equiv b_i \bmod m_i \text{ for } 1 \leq i \leq r.$$

1. *Compute $m = m_1 \cdots m_r$ and $\frac{m}{m_i}$ for $1 \leq i \leq r$.*
2. *For $1 \leq i \leq r$ compute $s_i, t_i \in R$ with*

   $$s_i \frac{m}{m_i} + t_i m_i = 1,$$

   using the Extended Euclidean Algorithm , and set $c_i = b_i s_i \operatorname{rem} m_i \in R$, where rem *is the remainder as in Definition 9.1.*
3. *Set $c = \sum_{1 \leq i \leq r} c_i \frac{m}{m_i}$.*

To see that the algorithm works correctly, we observe that $c_i \frac{m}{m_i} \equiv 0 \bmod m_j$ for $j \neq i$ and $c_i \frac{m}{m_i} \equiv b_i s_i \frac{m}{m_i} \equiv b_i \bmod m_i$ by step 2, and hence $c \equiv c_i \frac{m}{m_i} \equiv b_i \bmod m_i$ for $1 \leq i \leq r$ as claimed.

Example 10.4. *1. $R = \mathbb{Z}$, $m_i = p_i^{e_i}$ for $1 \leq i \leq r$, where $p_i \in \mathbb{N}$ are pairwise distinct primes and $e_i \in \mathbb{N}_{>0}$ for $1 \leq i \leq r$. Then*

$$m = \prod_{1 \leq i \leq r} p_i^{e_i}$$

is the prime decomposition of $m \in \mathbb{Z}$. The CRT tells us that

$$\mathbb{Z}/(m) \cong \mathbb{Z}/(p_1^{e_1}) \times \cdots \times \mathbb{Z}/(p_r^{e_r}),$$

and for arbitrary $b_1, \ldots, b_r \in \mathbb{Z}$ the CRA computes a solution $c \in \mathbb{Z}$ of the system of congruences

$$c \equiv b_i \bmod p_i^{e_i} \text{ for } 1 \leq i \leq r.$$

For example, take $r = 2$, $m_1 = 11$, $m_2 = 13$, *and* $m = 11 \cdot 13 = 143$, *and find* $c \in \mathbb{Z}$ *with* $0 \le c < m$ *and*

$$\begin{aligned} c &\equiv 2 \bmod 11, \\ c &\equiv 7 \bmod 13. \end{aligned}$$

There is nothing to do in step 1 of Algorithm 10.3. In step 2, we have to apply the Extended Euclidean Algorithm to 11 *and* 13 *and get* $6 \cdot 13 + (-7) \cdot 11 = 1$, *i.e.,* $s_1 = 6$ *and* $s_2 = -7$. *Then*

$$\begin{aligned} c_1 &= b_1 s_1 \operatorname{rem} m_1 = 2 \cdot 6 \operatorname{rem} 11 = 1, \\ c_2 &= b_2 s_2 \operatorname{rem} m_2 = 7 \cdot (-7) \operatorname{rem} 13 = 3. \end{aligned}$$

Finally, in step 3 we compute

$$c = c_1 \frac{m}{m_1} + c_2 \frac{m}{m_2} = 1 \cdot 13 + 3 \cdot 11 = 46,$$

and indeed $46 = 4 \cdot 11 + 2 = 3 \cdot 13 + 7$.

2. $R = F[x]$, *where* F *is a field,* $m_i = x - u_i$ *for* $1 \le i \le r$, *where* $u_1, \ldots, u_r \in F$ *are pairwise distinct. First we note that that* $f \equiv f(u_i) \bmod (x - u_i)$ *for* $1 \le i \le r$ *and arbitrary* $f \in F[x]$. *Hence the ring homomorphism*

$$\begin{aligned} \chi\colon F[x] &\longrightarrow F[x]/(x-u_1) \times \cdots \times F[x]/(x-u_r) \cong F^r \\ f &\longmapsto (f(u_1), \ldots, f(u_r)) \end{aligned}$$

from Theorem 10.1 is just the evaluation homomorphism at the r *points* $u_1, \ldots, u_r$. *Moreover, the* l_i *from the proof of Theorem 10.1 satisfying*

$$\begin{aligned} l_i &\equiv l_i(u_i) = 1 \bmod (x - u_i), \\ l_i &\equiv l_i(u_j) = 0 \bmod (x - u_j) \text{ for } j \ne i \end{aligned}$$

and $\deg l_i < r$ *are the Lagrange interpolants*

$$l_i = \prod_{\substack{1 \le j \le r \\ j \ne i}} \frac{x - u_j}{u_i - u_j}.$$

If $b_1, \ldots, b_r \in F$ *are scalars, then* $c = \sum_{1 \le i \le r} b_i l_i$ *is nothing but the familiar Lagrange interpolation polynomial satisfying*

$$c(u_i) = b_i \text{ for } 1 \le i \le r. \tag{10.2}$$

So Chinese remaindering for r *distinct monic linear polynomials is the same as interpolation at* r *points, and the CRT tells us once more what we already know: the interpolation polynomial is unique modulo* $m = \prod_{1 \le i \le r}(x - u_i)$, *i.e., there is exactly one polynomial* $c \in F[x]$ *of degree strictly less than* r *which solves the interpolation problem (10.2). In fact, it is useful to think of the CRT as a generalization of interpolation.*

THEOREM 10.5. *Let $R = F[x]$, $m_1, \ldots, m_r, m \in R$ as in (10.1), $d_i = \deg m_i \geq 1$ for $1 \leq i \leq r$, $n = \deg m = \sum_{1 \leq i \leq r} d_i$, and $b_i \in F[x]$ of degree $\deg b_i < d_i$. Then the unique solution $c \in F[x]$ with $\deg c < n$ of the Chinese Remainder Problem*

$$c \equiv b_i \bmod m_i \text{ for } 1 \leq i \leq r$$

for polynomials can be computed using $O(n^2)$ operations in F.

PROOF. In step 1 of Algorithm 10.3, we first successively compute $m_1, m_1 m_2, \ldots, m_1 \cdots m_r$ with at most

$$\begin{aligned}
2 \sum_{2 \leq i \leq r} (d_1 + \ldots + d_{i-1} + 1)(d_i + 1) &= 2 \sum_{1 \leq j < i \leq r} d_j(d_i + 1) + 2 \sum_{2 \leq i \leq r} (d_i + 1) \\
&< 2 \sum_{1 \leq i,j \leq r} d_j(d_i + 1) + 2 \sum_{1 \leq i \leq r} (d_i + 1) \\
&= 2\Big(\sum_{1 \leq j \leq r} d_j \Big) \Big(\sum_{1 \leq i \leq r} (d_i + 1) \Big) + 2 \sum_{1 \leq i \leq r} (d_i + 1) \\
&= 2(n+1)(n+r) \in O(n^2)
\end{aligned}$$

operations in F using classical polynomial multiplication. (Actually, this can be done with $O(\mathsf{M}(n) \log n)$ operations when we use fast multiplication, as we will see later in this section.)

Next, we compute $\frac{m}{m_i}$ for $1 \leq i \leq r$. Each division takes $\leq 2(d_i + 1)(n - d_i + 1)$ operations, together we have

$$2 \sum_{1 \leq i \leq r} (d_i + 1)(n - d_i + 1) \leq 2n \sum_{1 \leq i \leq r} (d_i + 1) = 2n(n+r) \in O(n^2)$$

operations.

Fix $1 \leq i \leq r$ in step 2. The Extended Euclidean Algorithm with input $\frac{m}{m_i}$ and m_i takes $O(d_i(n - d_i))$ operations. By the degree formula for s_i (Lemma 9.17), we have $\deg s_i < \deg m_i = d_i$, and hence the multiplication of b_i and s_i, together with the subsequent division with remainder by m_i, takes $O(d_i^2)$ operations. So we have $O(d_i n)$ operations for each i, and $O(n^2)$ for step 2.

Finally, in step 3 we need $O(d_i(n - d_i))$ operations for the multiplication of c_i and $\frac{m}{m_i}$ for $1 \leq i \leq r$, and $O(rn)$ for the addition of all the products (their degree is strictly less than n). This gives a cost of $O(n^2)$ for step 3, and also a total cost of $O(n^2)$ for the whole algorithm. □

COROLLARY 10.6. *Let $n \in \mathbb{N}$, $u_0, \ldots, u_{n-1} \in F$ pairwise distinct, and $b_0, \ldots, b_{n-1} \in F$. Then the unique interpolating polynomial $f \in F[x]$ of degree less than n with $f(u_i) = b_i$ for $0 \leq i < n$ can be computed with $O(n^2)$ operations in F.*

We have already seen in section 6 an algorithm for evaluating a polynomial in $F[x]$ of degree less than n at n points in F that uses $O(\mathsf{M}(n) \log n)$ operations in F, and we mentioned that the same time bound can be also achieved for interpolation at n points. A proof for the latter can e.g. be found in Borodin & Munro (1975), §4.5. Similar results hold for the computational cost of both directions of the isomorphism of the Chinese Remainder Theorem in the general polynomial case (where not all modules need to be linear polynomials), and in the case of integers.

For perspective, we note that the classical algorithms, essentially implementing straightforward formulas for the problems of this section, take time n^2, while ours take essentially linear time, up to factors $\log n$. These were landmark achievements in the golden age of algebraic complexity theory, in the early 1970's. Furthermore, it is conjectured that these running times cannot be improved; this has been proven (with highly nontrivial methods from algebraic geometry) in the so-called *nonscalar model*, where additions and multiplications by scalars from F are for free, and thus $\mathsf{M}(n) = n$. All this progress is based on fast multiplication; using classical multiplication, with $\mathsf{M}(n) = n^2$, these "fast" algorithms are actually slower than naive ones.

10.2. Secret sharing. A neat application of interpolation is secret sharing: you want to give to n players a shared secret, so that together they discover it, but no proper subset of the players can. To achieve this, you identify possible secrets with elements of the finite field $\mathbb{F}_p$ for an appropriate p. If your secrets are PIN codes for ec cards, i.e., four-digit decimal numbers, you choose a prime p just bigger than 10000, say $p = 10007$. Then you choose $2n-1$ random elements $f_1, \ldots, f_{n-1}, u_0, \ldots, u_{n-1} \in \mathbb{F}_p$ uniformly and independently with all u_i nonzero, call your secret f_0, set $f = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1} \in \mathbb{F}_p[x]$, and give to player number i the value $f(u_i) \in \mathbb{F}_p$. (If $u_i = u_j$ for some $i \neq j$, you have to make a new random choice; this is unlikely to happen if $n < \sqrt{p}$.) Then together they can determine the (unique) interpolation polynomial f of degree less than n, and thus f_0. But if any smaller number of them, say $n-1$, get together, then the possible interpolation polynomials consistent with this partial knowledge are such that each value in $\mathbb{F}_p$ of f_0 is equally likely: they have no information on f_0.

EXERCISE 10.7. *Determine the set of all interpolation polynomials $g \in \mathbb{F}_p[x]$ of degree less than n with $g(u_i) = f(u_i)$ for $1 \leq i \leq n-1$.*

There is also a way to extend the above scheme to the situation where $k \leq n$ and each subset of k players can recover the secret, but no set of fewer than k players can. This can be achieved by randomly and independently choosing $n + k - 1$ elements $u_0, \ldots, u_{n-1}, f_1, \ldots, f_{k-1} \in \mathbb{F}_p$ and giving $f(u_i)$ to player i, where $f = f_0 + f_1 x + \cdots + f_{k-1}x^{k-1} \in \mathbb{F}_p[x]$ and $f_0 \in \mathbb{F}_p$ is the secret as above. Again, it is requested that $u_i \neq u_j$ if $i \neq j$. Since f is uniquely determined by its values at k points, each subset of k out of the n players can calculate f and thus the secret f_0, but fewer than k players have no information on f_0.

11. Subresultant theory

11.1. Subresultants and the Extended Euclidean Scheme. Let F be a field, and $f, g \in F[x]$ with

$$\deg f = n \geq \deg g = m \geq 0.$$

Recall the *Extended Euclidean Scheme* for f and g:

$$\begin{aligned} a_0 = f,\ a_1 &= g \\ a_0 &= a_1 q_1 + a_2 \\ a_1 &= a_2 q_2 + a_3 \\ &\vdots \\ a_{k-1} &= a_k q_k + a_{k+1} \\ &\vdots \\ a_{\ell-1} &= a_\ell q_\ell \end{aligned} \qquad \begin{aligned} s_0 = 1,\ t_0 &= 0 \\ s_1 = 0,\ t_1 &= 1 \\ s_{k+1} &= s_{k-1} - s_k q_k \text{ for } 1 \leq k < \ell \\ t_{k+1} &= t_{k-1} - t_k q_k \text{ for } 1 \leq k < \ell \end{aligned}$$

with $\deg a_{k+1} < \deg a_k$ for all $k \geq 1$.

We define the *degree sequence* $(n_0, n_1, \ldots, n_\ell)$ by $n_k = \deg a_k$ for all k. We have

$$n = n_0 \geq n_1 > n_2 \cdots > n_\ell \geq 0.$$

It is convenient to set $a_{\ell+1} = 0$ and $n_{\ell+1} = -\infty$.

The number of arithmetic operations in F performed by the Euclidean Algorithm (and the Extended Euclidean Algorithm) for f and g is $O(n^2)$ (see section 9).

Suppose for a moment that $F = \mathbb{Q}$. Then to get a bound on the bit complexity of Euclid's algorithm, we need to get a bound on the size of the numbers involved in the computation. Define the *size* $L(a)$ of a polynomial $a \in \mathbb{Q}[x]$ to be the number of bits required to encode a. More precisely, we use

- $L(a) = \lfloor \log_2 a \rfloor + 2$, when $a \in \mathbb{Z}$ (including a bit for the sign),
- $L(a) = L(b) + L(c)$, when $a = b/c \in \mathbb{Q}$ with $b, c \in \mathbb{Z}$ and $\gcd(b, c) = 1$,
- $L(a) = \sum_i L(a_i)$, when $a = \sum_i a_i x^i \in \mathbb{Q}[x]$ with all $a_i \in \mathbb{Q}$.

Then for $a, b \in \mathbb{Z}[x]$ and $c, d \in \mathbb{Q}$, we have

$$\begin{aligned} L(ab) &\leq L(a) + L(b) + \log_2(1 + \deg ab), \\ L(cd), L(c/d) &\leq L(c) + L(d), \\ L(c + d) &\leq L(c) + L(d) + 1. \end{aligned}$$

Exercise 11.1. *Give an estimate for $L(ab)$ when $a, b \in \mathbb{Q}[x]$.*

We next consider a division with remainder $a = qb + r$, where $a = \sum_{0\le i\le n} a_i x^i$, $b = \sum_{0\le i\le m} \in \mathbb{Q}[x]$, $a_n b_m \ne 0$, in the special case where $m = n-1$. Then

$$\begin{aligned} q &= \frac{a_n}{b_m}x + \frac{a_{n-1}b_m - a_n b_{m-1}}{b_m^2}, \\ L(q) &\le L(b) + \max\{L(a), L(b)\}, \end{aligned}$$

and the last estimate is essentially sharp.

Exercise 11.2. *Give an estimate for $L(q), L(r)$ when $a, b \in \mathbb{Q}[x]$ are arbitrary.*

In a typical execution of the Euclidean algorithm, the degrees of all the quotient polynomials will be 1. From the above worst-case estimate, we then find that $L(a_l) \in O(2^l \cdot \max\{L(a_0), L(a_1)\})$. This looks like bad news: an exponential upper bound on the size of the gcd and the bit cost of the Euclidean Algorithm.

In reality, however, the sizes do not double at every step, and we can prove that the sizes of polynomials in the Euclidean Scheme remain polynomially bounded in the input size. To prove this non-obvious result, we need a "global view" of the Euclidean Algorithm provided by the theory of subresultants. This theory will give us explicit formulas for the coefficients that appear in the polynomials of the Euclidean Scheme; from these formulas, we will easily deduce bounds on their size. As a bonus, this theory will allow us to compute gcd's using a modular approach, yielding a much more practical algorithm.

Now let F be an arbitrary field.

Lemma 11.3. *(i) $\ell \le m + 1$.*

(ii) For $2 \le k \le \ell$, we have

$$\begin{aligned} \deg s_k &= \textstyle\sum_{2\le i<k} \deg q_i = m - n_{k-1} < m - n_k, \\ \deg t_k &= \textstyle\sum_{1\le i<k} \deg q_i = n - n_{k-1} < n - n_k, \\ \deg a_k + \deg t_k &< n. \end{aligned}$$

(iii) For $0 \le k < \ell$, we have

$$s_k t_{k+1} - t_k s_{k+1} = (-1)^k.$$

Proof. (i) and (ii) were already proved in section 9.3. Note that the sum $\sum_{2\le i<k} \deg q_i$ is to be interpreted as zero when $k = 2$.

(iii) can also be proved by induction on k, but it is most clearly seen by noting that

$$\begin{pmatrix} s_k & t_k \\ s_{k+1} & t_{k+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix}.$$

Then taking determinants of both sides, we obtain

$$\begin{aligned} s_k t_{k+1} - t_k s_{k+1} &= \det \begin{pmatrix} s_k & t_k \\ s_{k+1} & t_{k+1} \end{pmatrix} \\ &= \det \begin{pmatrix} 0 & 1 \\ 1 & -q_k \end{pmatrix} \cdots \det \begin{pmatrix} 0 & 1 \\ 1 & -q_1 \end{pmatrix} = (-1)^k. \quad \square \end{aligned}$$

LEMMA 11.4. *Let $s, t \in F[x]$ with $a = sf + tg \neq 0$ and $t \neq 0$, and suppose that*

$$\deg a + \deg t < n.$$

Then there exists a nonzero $c \in F[x]$ and k with $1 \leq k \leq \ell$ such that

$$s = cs_k, \ t = ct_k, \ a = ca_k.$$

PROOF. Define k by

$$n_k \leq \deg a < n_{k-1}. \tag{11.1}$$

First, we claim that $s_k t = st_k$. Suppose that the claim is false, and consider the equation

$$\begin{pmatrix} s_k & t_k \\ s & t \end{pmatrix} \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} a_k \\ a \end{pmatrix}.$$

The coefficient matrix is nonsingular, and we can solve for f in $F(x)$ using Cramer's rule, obtaining

$$f = \frac{\det \begin{pmatrix} a_k & t_k \\ a & t \end{pmatrix}}{\det \begin{pmatrix} s_k & t_k \\ s & t \end{pmatrix}}. \tag{11.2}$$

The degree of the left-hand side of (11.2) is n, whereas

$$\begin{aligned} \deg(a_k t - a t_k) &\leq \max\{\deg a_k + \deg t, \deg a + \deg t_k\} \\ &\leq \max\{\deg a + \deg t, \deg a + n - n_{k-1}\} \\ &< \max\{n, n_{k-1} + n - n_{k-1}\} = n, \end{aligned}$$

by Lemma 11.3(ii) and (11.1), and the degree of the right-hand side is strictly less than n. Thus we have a contradiction, proving the claim.

Now, Lemma 11.3(iii) implies that s_k and t_k are relatively prime, and from the claim we have $t_k \mid s_k t$, so that $t_k \mid t$. We write $t = ct_k$, where $c \in F[x]$ and $c \neq 0$, since $t \neq 0$. Then we have $st_k = s_k t = cs_k t_k$, and cancelling t_k [why is $t_k \neq 0$?], we obtain $s = cs_k$. Furthermore,

$$a = sf + tg = c(s_k f + t_k g) = ca_k. \quad \square$$

THEOREM 11.5. *Let $0 \leq i < m$. Then i does* not *appear in the degree sequence if and only if there exist $s, t \in F[x]$ satisfying*

$$t \neq 0, \ \deg s < m - i, \ \deg t < n - i, \ \deg(sf + tg) < i. \tag{11.3}$$

PROOF. "$\Longrightarrow$": Suppose that i does not appear in the degree sequence. Then there exists a k with $2 \leq k \leq \ell + 1$ such that

$$n_k < i < n_{k-1}.$$

We claim that $s = s_k$ and $t = t_k$ do the job. We have $sf + tg = a_k$, and $\deg a_k = n_k < i$. Furthermore, from Lemma 11.3(ii), we have

$$\begin{aligned} \deg s &= m - n_{k-1} < m - i, \\ 0 \le \deg t &= n - n_{k-1} < n - i. \end{aligned}$$

(The case $k = \ell + 1$ gives $s = g/a_\ell$ and $t = -f/a_\ell$ where $i < n_\ell$ and $a_{\ell+1} = 0$.)

"$\Longleftarrow$": Suppose there exist $s, t \in F[x]$ satisfying (11.3). By Lemma 11.4, there exist $k \in \{1, \ldots, \ell\}$ and $c \in F[x] \setminus \{0\}$ such that $t = ct_k$ and $a = ca_k$. Then from Lemma 11.3 (ii) we find

$$\begin{aligned} n - n_{k-1} &\le \deg c + n - n_{k-1} = \deg(ct_{k-1}) = \deg t < n - i, \\ n_k &\le \deg c + n_k = \deg(ca_k) = \deg a < i. \end{aligned}$$

Together these imply that $n_k < i < n_{k+1}$, so that i is between two consecutive remainder degrees and does not occur in the degree sequence. □

The key to the theory we are developing is to restate Theorem 11.5 in the language of linear algebra. Suppose that

$$f = \sum_{0 \le j \le n} f_j x^j, \quad g = \sum_{0 \le j \le m} g_j x^j,$$

with all $f_j, g_j \in F$. Furthermore, let $0 \le i < m$, and let $s, t \in F[x]$ be arbitrary polynomials of the form

$$s = \sum_{0 \le j < m-i} y_j x^j, \quad t = \sum_{0 \le j < n-i} z_j x^j,$$

with all $y_j, z_j \in F$, and where y_{m-i-1} and z_{n-i-1} may be zero. Finally, set

$$a = sf + tg = \sum_{0 \le j < n+m-i} u_j x^j \in F[x].$$

For $0 \le i \le m$, let P_i be the $(n + m - 2i) \times (n + m - 2i)$ matrix with entries in F defined as

$$P_i = \begin{pmatrix} f_n & & & & g_m & & & \\ f_{n-1} & f_n & & & g_{m-1} & g_m & & \\ f_{n-2} & f_{n-1} & f_n & & g_{m-2} & g_{m-1} & g_m & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

where there are $m - i$ columns of f's and $n - i$ columns of g's.

Then by comparing coefficients in the equation $a = sf + tg$, one sees that

$$\begin{pmatrix} u_{n+m-i-1} \\ \vdots \\ u_i \end{pmatrix} = P_i \cdot \begin{pmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{pmatrix}.$$

From this, we see that

$$\deg a < i \quad\Longleftrightarrow\quad P_i \cdot \begin{pmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix},$$

and

$$\deg a = i \text{ and } a \text{ monic} \quad\Longleftrightarrow\quad P_i \cdot \begin{pmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

The next theorem now follows almost immediately from Theorem 11.5.

THEOREM 11.6. *Let $0 \leq i < m$, and $0 \leq k \leq \ell + 1$.*

(i) i appears in the degree sequence $\Longleftrightarrow$ $\det P_i \neq 0$.

(ii) If $i = n_k$, and $y_0, \ldots, y_{m-i-1}, z_0, \ldots, z_{n-i-1} \in F$ are the (unique) solutions to

$$P_i \cdot \begin{pmatrix} y_{m-i-1} \\ \vdots \\ y_0 \\ z_{n-i-1} \\ \vdots \\ z_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{11.4}$$

then

$$s_k = c_k \sum_{0 \leq j < m-i} y_j x^j, \quad t_k = c_k \sum_{0 \leq j < n-i} z_j x^j,$$

where c_k is the leading coefficient of a_k.

PROOF. Restating Theorem 11.5 in the language of linear algebra, we see that

$$\begin{aligned} & i \text{ does not appear in the degree sequence} \\ \Longleftrightarrow\quad & \text{the equation } P_i v = 0 \text{ has a nonzero solution} \\ \Longleftrightarrow\quad & P_i \text{ is singular} \\ \Longleftrightarrow\quad & \det P_i = 0. \end{aligned}$$

This proves (i).

To prove (ii), note that if $i = n_k$, then s_k and t_k are polynomials of degree less than $m-i$, $n-i$, respectively, such that $s_k f + t_k g = a_k$. Set $s = c_k^{-1} s_k$, and $t = c_k^{-1} t_k$. Then $a = sf + tg$ is a monic polynomial of degree i. Thus, the coefficients of s and t satisfy (11.4). The solution to (11.4) is of course unique, since $\det P_i \neq 0$. □

If R is an integral domain and $f, g \in R[x]$, then $\mathrm{Syl}(f,g) = P_0 \in R^{(m+n)\times(m+n)}$ is their *Sylvester matrix*, and $\mathrm{res}(f,g) = \det P_0 \in R$ is their *resultant*. The elements $\det P_i \in R$ for $0 \le i \le \deg g$ are the *subresultants*.

EXAMPLE 11.7. *Let $f = a_0 = x^4 + x^3 - x^2 + 3x + 1$ and $g = a_1 = x^3 - x^2 + x$ in $\mathbb{Z}[x]$. The entries of the Euclidean Scheme are*

$$\begin{aligned} a_0 &= a_1 q_1 + a_2 = a_1(x+2) + (x+1), \\ a_1 &= a_2 q_2 + a_3 = a_2(x^2 - 2x + 3) - 3, \\ a_2 &= a_3 q_3 = a_3(-\frac{1}{3}x - \frac{1}{3}). \end{aligned}$$

The degree sequence is $4, 3, 1, 0$, and 2 does not appear. The determinants of the matrices P_i are

$$\begin{aligned} \det P_0 &= \det \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 & -1 & 1 & 0 \\ 3 & -1 & 1 & 0 & 1 & -1 & 1 \\ 1 & 3 & -1 & 0 & 0 & 1 & -1 \\ 0 & 1 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = 3, \\ \det P_1 &= \det \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 1 & 0 \\ -1 & 1 & 1 & -1 & 1 \\ 3 & -1 & 0 & 1 & -1 \\ 1 & 3 & 0 & 0 & 1 \end{pmatrix} = 1, \\ \det P_2 &= \det \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix} = 0, \\ \det P_3 &= \det \begin{pmatrix} 1 \end{pmatrix} = 1, \end{aligned}$$

and in fact $\det P_i$ vanishes for exactly those $i \in \{0, 1, 2, 3\}$ that do not appear in the degree sequence, namely $i = 2$.

We highlight the special case of Theorems 11.5 and 11.6 for the gcd.

COROLLARY 11.8. *Let F be a field, and $f, g \in F[x]$. Then the following are equivalent:*

(i) $\deg \gcd(f, g) = d$,

(ii) $\det P_0 = \cdots = \det P_{d-1} = 0, \quad \det P_d \neq 0,$

(iii) there exist $s, t \in F[x]$ *such that*

$$t \neq 0,\ \deg s \leq m - d,\ \deg t \leq n - d,\ sf + tg = 0,$$

and no such s, t *with smaller degree.*

In (iii), we can take $s = g/\gcd(f, g)$ and $t = -f/\gcd(f, g)$. Because it is so important, here is the special case $d = 0$:

COROLLARY 11.9. *Let* F *be a field, and* $f, g \in F[x]$*. Then the following are equivalent:*

(i) $\gcd(f, g) = 1,$

(ii) $\operatorname{res}(f, g) = \det P_0 \neq 0,$

(iii) there do not exist $s, t \in F[x]$ *such that*

$$t \neq 0,\ \deg s < m,\ \deg t < n,\ sf + tg = 0.$$

EXAMPLE 11.10. *If* $F \subseteq K$ *are fields,* $f, g \in F[x]$ *and* $h \in K[x]$ *is nonconstant and divides* f *and* g*, then there is also a nonconstant polynomial* $k \in F[x]$ *dividing* f *and* g*. This is because the resultant* $\operatorname{res}(f, g)$ *is the same whether we consider it over* F *or* K*. By assumption, it is zero in* K*, hence also in* F*. Note the difference with the fact that* f *may very well have a nontrivial factor, with degree between* 1 *and* $\deg f - 1$*, over* K*, but not over* F*.*

The following example illustrates the huge coefficients that actually occur in the Euclidean Algorithm in $\mathbb{Q}[x]$. It is typical in the sense that for most pairs of polynomials, with about as many coefficient digits as the degree, a similar growth of intermediate results occurs.

EXAMPLE 11.11. *The following is generated on most platforms by the* MAPLE *commands in the lines beginning with a* `>` *character.*

```
> a[0] := randpoly(x, coeffs = rand(-999 .. 999), degree = 5);
> a[1] := randpoly(x, coeffs = rand(-999 .. 999), degree = 4);
```

$$a_0 := 824\,x^3 - 65\,x^2 - 814\,x - 741 - 979\,x^5 - 764\,x^4$$

$$a_1 := 216\,x^2 + 663\,x + 880 + 916\,x^4 + 617\,x^3$$

```
> for i from 1 to 5 do
>   q[i] := quo(a[i - 1], a[i], x, 'a[i + 1]');
>   a[i + 1] := a[i + 1];
> od;
```

$$q_1 := -\frac{979}{916}\,x - \frac{95781}{839056}$$

$$a_2 := \frac{944180045}{839056}x^3 + \frac{140176147}{209764}x^2 + \frac{169663539}{839056}x - \frac{33590826}{52441}$$

$$q_2 := \frac{768575296}{944180045}x + \frac{57856003253091792}{891475957376202025}$$

$$\begin{aligned} a_3 := {} & \frac{7159437220924839504}{891475957376202025}x^2 + \frac{10441762166177331954 72}{891475957376202025}x \\ & + \frac{8215582223337388969 12}{891475957376202025} \end{aligned}$$

$$\begin{aligned} q_3 := {} & \frac{841713809551880509893591125}{600716875684031213486822 4}x \\ & - \frac{115913182735422536794733563949711484383925}{5695282368929332126198133985350996224} \end{aligned}$$

$$\begin{aligned} a_4 := {} & \frac{48280541203865249645597762299683876782 5}{2036317850869071477779123438251 2}x \\ & + \frac{127308019850675360362336365366155980725}{678772616956357159259707812750 4} \end{aligned}$$

$$\begin{aligned} q_4 := {} & 1457888981514570687603635731723980453185101443540 48/ \\ & 43040941692356942895843220726973615364758999657\backslash \\ & 9269845625x + 1145301654647800888897186913461 14\backslash \\ & 490894496680600994590857601818600214133 76/23310\backslash \\ & 10658938130129540076477498181947005081674458487\backslash \\ & 091341947533388642552306 25 \end{aligned}$$

$$\begin{aligned} a_5 := {} & 120030364377926256361133140278480945412591234 27\backslash \\ & 040801461552/2614776808786606231660286291119753\backslash \\ & 81317669206915015415511225 \end{aligned}$$

$$\begin{aligned} q_5 := {} & 126242839455533029309330874298262527750676960 72\backslash \\ & 23660855602834770538605446478592260960399214563\backslash \\ & 35625/2444199736290903479601424877771496073422 9\backslash \\ & 32690718636017545835981076141855817646838291786\backslash \\ & 24x + 33288205787809083716735227313062571758896\backslash \\ & 37940136044568859695730328247172040230439213096\backslash \\ & 7121138125/814733245430301159867141625923832024\backslash \\ & 47431089690621200584861199369204728527254894609\backslash \\ & 726208 \end{aligned}$$

$$a_6 := 0$$

```
> with(linalg):
> P[0] := transpose(sylvester(a[0], a[1], x));
> P[1] := submatrix(P[0], 1 .. 7, [1, 2, 3, 5, 6, 7, 8]);
> P[2] := submatrix(P[1], 1 .. 5, [1, 2, 4, 5, 6]);
> P[3] := submatrix(P[2], 1 .. 3, [1, 3, 4]);
> P[4] := submatrix(P[3], 1 .. 1, [1]);
> for i from 0 to 4 do
>   p[i] := det(P[i]);
> od;
```

$$P_0 := \begin{bmatrix} -979 & 0 & 0 & 0 & 916 & 0 & 0 & 0 & 0 \\ -764 & -979 & 0 & 0 & 617 & 916 & 0 & 0 & 0 \\ 824 & -764 & -979 & 0 & 216 & 617 & 916 & 0 & 0 \\ -65 & 824 & -764 & -979 & 663 & 216 & 617 & 916 & 0 \\ -814 & -65 & 824 & -764 & 880 & 663 & 216 & 617 & 916 \\ -741 & -814 & -65 & 824 & 0 & 880 & 663 & 216 & 617 \\ 0 & -741 & -814 & -65 & 0 & 0 & 880 & 663 & 216 \\ 0 & 0 & -741 & -814 & 0 & 0 & 0 & 880 & 663 \\ 0 & 0 & 0 & -741 & 0 & 0 & 0 & 0 & 880 \end{bmatrix}$$

$$P_1 := \begin{bmatrix} -979 & 0 & 0 & 916 & 0 & 0 & 0 \\ -764 & -979 & 0 & 617 & 916 & 0 & 0 \\ 824 & -764 & -979 & 216 & 617 & 916 & 0 \\ -65 & 824 & -764 & 663 & 216 & 617 & 916 \\ -814 & -65 & 824 & 880 & 663 & 216 & 617 \\ -741 & -814 & -65 & 0 & 880 & 663 & 216 \\ 0 & -741 & -814 & 0 & 0 & 880 & 663 \end{bmatrix}$$

$$P_2 := \begin{bmatrix} -979 & 0 & 916 & 0 & 0 \\ -764 & -979 & 617 & 916 & 0 \\ 824 & -764 & 216 & 617 & 916 \\ -65 & 824 & 663 & 216 & 617 \\ -814 & -65 & 880 & 663 & 216 \end{bmatrix}$$

$$P_3 := \begin{bmatrix} -979 & 916 & 0 \\ -764 & 617 & 916 \\ 824 & 216 & 617 \end{bmatrix}$$

$$
\begin{aligned}
P_4 &:= \begin{bmatrix} -979 \end{bmatrix} \\
p_0 &:= 176834423457073273392629476 3 \\
p_1 &:= 1624739539111010690619 \\
p_2 &:= 8532728710509 \\
p_3 &:= 944180045 \\
p_4 &:= -979
\end{aligned}
$$

Theorem 11.6 gives almost explicit formulas for the coefficients of s_k, t_k, and a_k; our next goal is to derive a similarly explicit formula for the leading coefficient c_k. Assuming the notation in Theorem 11.6, let $0 \le i < m$, $i = n_k$,

$$\hat{s}_k = c_k^{-1} s_k = \sum_{0 \le j < m-i} y_j x^j, \quad \hat{t}_k = c_k^{-1} t_k = \sum_{0 \le j < n-i} z_j x^j.$$

For $0 \le j \le m - i - 1$, let $Y_i^{(j)}$ be the matrix obtained by replacing the column in P_i corresponding to the variable y_j in (11.4) by the vector on the right-hand side of (11.4). Then by Cramer's rule, we have

$$y_j = \frac{\det Y_i^{(j)}}{\det P_i}. \tag{11.5}$$

Similarly, for $0 \le j \le n - i - 1$, define $Z_i^{(j)}$ to be the matrix obtained by replacing the column in P_i corresponding to the variable z_j in (11.4) by the vector on the right-hand side of (11.4). Then

$$z_j = \frac{\det Z_i^{(j)}}{\det P_i}. \tag{11.6}$$

Theorem 11.12. *Let $2 \le k \le \ell$, $\gamma_2 = \det Y_{n_2}^{(0)}$, and for $3 \le j \le \ell$,*

$$\gamma_j = \det Y_{n_{j-1}}^{(0)} \cdot \det Z_{n_j}^{(0)} - \det Y_{n_j}^{(0)} \cdot \det Z_{n_{j-1}}^{(0)}.$$

Then

$$c_k = (-1)^{(k+1)(k+2)/2} \cdot \det P_{n_k} \cdot \prod_{2 \le j \le k} \gamma_j^{(-1)^{k+j-1}}.$$

Proof. We have $1 = s_2 = c_2 \hat{s}_2$, so

$$c_2 = (\text{constant-term}(\hat{s}_2))^{-1} = y_0^{-1} = \det P_{n_2} \cdot \gamma_2^{-1}.$$

For $3 \le k \le \ell$, we have from Lemma 11.3 (iii)

$$(-1)^{k-1} = s_{k-1} t_k - t_{k-1} s_k = c_{k-1} c_k (\hat{s}_{k-1} \hat{t}_k - \hat{t}_{k-1} \hat{s}_k).$$

Thus,

$$\begin{aligned} c_k &= (-1)^{k-1}(\text{constant-term}(\hat{s}_{k-1}\hat{t}_k - \hat{t}_{k-1}\hat{s}_k))^{-1}c_{k-1}^{-1} \\ &= (-1)^{k-1}\det P_{n_k} \cdot \det P_{n_{k-1}} \cdot \gamma_k^{-1} c_{k-1}^{-1}. \end{aligned}$$

The claim follows by induction on k. Notice that the product of the determinants $\det P_j$ telescopes. □

We are now in a position to derive explicit bounds on the size of the coefficients in the Euclidean Scheme over $\mathbb{Q}$.

THEOREM 11.13. *Assume the above notation, and that $f, g \in \mathbb{Q}[x]$ have integer coefficients bounded in absolute value by M. Then for $2 \le k \le \ell$, the coefficients of the polynomials of a_k, s_k, and t_k are rational numbers whose numerator and denominator (when expressed in lowest terms) are bounded in absolute value by*

$$2^{n(n+3)+1/2}n^{n(n+2)}(n+1) \cdot M^{2n(n+2)+1}.$$

For the monic associate of a_k (and so in particular for the monic gcd), the bounds are

$$2^n(n+1) \cdot n^n \cdot M^{2n+1}.$$

In particular, the bit complexity of Euclid's algorithm in this situation (assuming rational numbers are kept in lowest terms) is

$$(n + \log M)^{O(1)}.$$

PROOF. Hadamard's inequality states that for a $t \times t$ matrix with real coefficients bounded by M in absolute value, the absolute value of its determinant is bounded by $t^{t/2}M^t$.

We use Hadamard's inequality to bound the determinants of P_i, $Y_i^{(j)}$, and $Z_i^{(j)}$ appearing in (11.5), (11.6), and in Theorem 11.12. The result follows by a straightforward calculation. □

REMARK 11.14. *The second (and better) estimate suggests the following variant of the Euclidean Algorithm: instead of the a_k, use their monic associate $\widehat{a_k} = c_k^{-1}a_k$ at each step. These are the quantities naturally provided by subresultants, correctness follows as before ($\gcd(f,g) = \gcd(\widehat{a_{k-1}}, \widehat{a_k})$), but the coefficients are often considerably smaller. We demonstrate this effect for Example 11.11; almost any (random) input will exhibit the same behaviour:*

$$\widehat{q_1} := -\frac{979}{916}x - \frac{95781}{839056}$$

$$\widehat{a_2} := x^3 + \frac{560704588}{944180045}x^2 + \frac{169663539}{944180045}x - \frac{537453216}{944180045}$$

$$\widehat{q_2} := 916\,x + \frac{68953685157}{944180045}$$

$$\begin{aligned}
\widehat{a_3} &:= x^2 + \frac{414821822229479}{2844242903503}x + \frac{979145876239177}{8532728710509} \\
\widehat{q_3} &:= x - \frac{3900717067342309091747 91}{2685477392620393197635} \\
\widehat{a_4} &:= x + \frac{428417677887923659527}{541579846370336896873} \\
\widehat{q_4} &:= x + \frac{22344061481403647888990941967589608 6}{154038463471907569137435260144611 9} \\
\widehat{a_5} &:= 1 \\
\widehat{q_5} &:= x + \frac{428417677887923659527}{541579846370336896873} \\
\widehat{a_6} &:= 0
\end{aligned}$$

THEOREM 11.15. *Assume the above notation, and that $F = E(y)$, where E is a field, and that $f, g \in F[x]$ have coefficients in $E[y]$ of degree at most δ. Then for $2 \leq k \leq \ell$, the coefficients of the polynomials of a_k, s_k, and t_k are rational functions whose numerator and denominator (when expressed in lowest terms) are bounded in degree by*

$$(2n(n+2)+1)\delta.$$

For the monic associate of a_k (and so in particular for the monic gcd), the bounds are

$$(2n+1)\delta.$$

In particular, Euclid's algorithm in this situation (assuming rational functions are kept in lowest terms) uses

$$(n+\delta)^{O(1)}$$

arithmetic operations in E.

PROOF. The proof is along the same lines as that of Theorem 11.13. In this case, however, we use the upper bound $t\delta$ on the degree of the determinant of a $t \times t$ matrix whose entries are polynomials of degree at most δ. □

11.2. Notes. Modern subresultant theory was developed in the late 1960s by Collins (1967) and Brown & Traub (1971). At that time, the first computer algebra systems were available, and researchers were amazed at the empirically observed coefficient growth, as in Example 11.11 (and almost any random example over $\mathbb{Q}$). Collins realized that the entries of the Extended Euclidean Scheme can be described by subresultants, and thus polynomially bounded. This discovery lead to several new variants of the Euclidean Algorithm: using the monic versions of the remainders (Remark 11.14), and modular algorithms (section 13).

The presentation of subresultants in this course, with Lemma 11.4 and Theorem 11.5 as cornerstones, is based on von zur Gathen (1984), where the goal are parallel algorithms for the Extended Euclidean Scheme—not a topic of this course.

11.3. Intersecting plane curves. The resultant was invented by algebraic geometers (Sylvester) in the 19th century to solve the problem of curve intersection. Suppose we are given $f, g \in F[x, y]$, and want to intersect the two plane curves

$$X = \{(a, b) \in F^2 : f(a, b) = 0\},$$

$$Y = \{(a, b) \in F^2 : g(a, b) = 0\}.$$

Consider the resultant with respect to the variable y, i.e., writing $f = \sum_i f_i y^i$, $g = \sum_i g_i y^i$ with all $f_i, g_i \in F[x]$:

$$r = \mathrm{res}_y(f, g) \in F[x].$$

We assume that F is *algebraically closed*, i.e., that every nonconstant univariate polynomial over F has a root. This is often required to make general statements about geometric objects such as our curves X and Y true. Intuition: $F = \mathbb{C}$. Now we let Z be the projection of $X \cap Y$ onto the first axis. Then for any $a \in F$ we have

$$\begin{aligned} a \in Z &\iff \exists b \in F \qquad f(a, b) = g(a, b) = 0 \\ &\iff \gcd(f(a, y), g(a, y)) \neq 1 \\ &\iff r(a) = \mathrm{res}_y(f, g)(a) = 0. \end{aligned}$$

Thus to determine $X \cap Y$, one first computes the $(n + m) \times (n + m)$-determinant over $F[x]$ giving $r \in F[x]$, then finds all roots of r, and for each such root a finds all the roots b of $\gcd(f(a, y), g(a, y)) \in F[y]$. The fact that $r(a) = 0$ guarantees that such a b exists.

This means that intersecting two plane curvesis reduced to finding roots of univariate polynomials, a much easier task. If $n = \deg f$ and $m = \deg g$, then $X \cap Y$ has “in general” nm points. This is called Bézout’s Theorem after the French 18th century geometer Bézout, and was first proven in the modern sense by van der Waerden in the 1920s. It is valid for arbitrary algebraic varieties, provided one counts points “at infinity” and “multiple points and components” properly.

Die Kapitel 12 "Modular algorithms" und 13 "Modular gcd computation" sind im Wintersemester 1995/96 nicht klausurrelevant.

12. Modular algorithms

12.1. The determinant.

PROBLEM 12.1. *Given* $A = (a_{ij})_{1 \le i,j \le n} \in \mathbb{Z}^{n \times n}$, *compute* $\det A \in \mathbb{Z}$.

We know from linear algebra that this problem can be solved by means of Gaussian elimination over $\mathbb{Q}$, which costs $\le 2n^3$ operations in $\mathbb{Q}$. But how large can the numerators and denominators of the intermediate coefficients grow? Consider the kth stage during the elimination and suppose for simplicity that A is nonsingular and that no row or column permutations are necessary.

$$\boxed{\begin{array}{ccccccc}
* & & & & & & \\
& \ddots & & & \large{*} & & \\
& & * & & & & \\
& & & a_{kk}^{(k)} & \cdots & a_{kj}^{(k)} & \cdots \\
& \large{0} & & \vdots & & \vdots & \\
& & & \vdots & & \vdots & \\
& & & a_{ik}^{(k)} & \cdots & a_{ij}^{(k)} & \cdots \\
& & & \vdots & & \vdots &
\end{array}}$$

The tableau represents the matrix after $k-1$ pivoting stages, a "$*$" denotes any rational number, and the upper diagonal entries are nonzero. The diagonal element $a_{kk}^{(k)} \ne 0$ is the new pivot element, and the entries of the kth column below the pivot element must be made to zero in the kth stage by subtracting an appropriate multiple of the kth row. The entries of the matrix for $k < i \le n$ and $k \le j \le n$ change according to the formula

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}. \tag{12.1}$$

If b_k is an upper bound for the absolute value of the numerators and denominators of all $a_{ij}^{(k)}$ for $1 \le i,j \le n$, so that in particular $|a_{ij}| \le b_0$ for $1 \le i,j \le n$, the formula (12.1) gives

$$b_k \le b_{k-1}^3 \le b_{k-2}^{3^2} \le \cdots \le b_0^{3^k},$$

which is an exponentially large upper bound in the input size. Bareiss (1968) showed, however, that the number of bit operations for Gaussian elimination over $\mathbb{Q}$ is in fact polynomial in the input size, but the proof is nontrivial. We use an alternate approach to reach the same goal, a polynomial time algorithm for the computation of $\det A$.

Algorithm 12.2. *Modular determinant.*
Input: $A = (a_{ij})_{1 \le i,j \le n} \in \mathbb{Z}^{n \times n}$.
Output: $\det A \in \mathbb{Z}$.

1. *Choose $m_1, \ldots, m_r \in \mathbb{N}$ pairwise relatively prime, e.g., the first r prime numbers, such that $m = \prod_{1 \le i \le r} m_i$ is "large enough".*
2. *For $1 \le i \le r$ compute $d_i = \det A \bmod m_i$ using Gaussian elimination over $\mathbb{Z}/(m_i)$.*
3. *Determine $d \in \mathbb{Z}$ of least absolute value with $d \equiv d_i \bmod m_i$ for $1 \le i \le r$ using the Chinese Remainder Algorithm.*
4. *Return d.*

Because $\det A$ is a polynomial expression in the coefficients of A, we have $\det A \equiv d_i \bmod m_i$ for $1 \le i \le r$ and hence $\det A \equiv d \bmod m$ by the relative primality of the moduli. If m is large enough (we will go into that later), then actually $d = \det A$.

Example 12.3. $A = \begin{pmatrix} 4 & 5 \\ 6 & -7 \end{pmatrix}$.
After Gaussian elimination, the matrix has the form

$$\begin{pmatrix} 4 & 5 \\ 0 & -\frac{29}{2} \end{pmatrix},$$

so $\det A = -58$. *We take the first four prime numbers as moduli and get*

i	m_i	d_i
1	2	0
2	3	2
3	5	2
4	7	−2

We have $m = 2 \cdot 3 \cdot 5 \cdot 7 = 210$, and the solutions to the Chinese Remainder System $d \equiv d_i \bmod m_i$ for $1 \le i \le 4$ are $d \in -58 + 210\mathbb{Z} = \{\ldots, -268, -58, 152, 362, \ldots\}$, and the correct solution -58 is the one of least absolute value. If we had taken only the first three primes, we would have incorrectly computed $d = 2$.

The example shows that is sufficient to have $m > 2|\det A|$, since then the only solution d of the Chinese Remainder System satisfying $-\frac{m}{2} < d < \frac{m}{2}$ is $\det A$. It remains to determine a "good" a priori bound on $|\det A|$, i.e., one that is only polynomially large in n and the size of the coefficients of A, and which is easy to find without calculating $\det A$. Let $b \in \mathbb{N}$ be such that $|a_{ij}| \le b$ for $1 \le i, j \le n$, e.g., $b = \max_{1 \le i,j \le n} |a_{ij}|$. Then

$$|\det A| = \Big| \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \cdot a_{1\sigma(1)} \cdots a_{n\sigma(n)} \Big| \le n!\, b^n.$$

In the above example, the bound $|\det A| \leq 2! \cdot 7^2 = 98$ is quite close, but in general it can still be strengthened by using Hadamard's inequality. It makes use of the Euclidean norm $\|v\|_2 = \sqrt{\sum_{1\leq j\leq n} v_j^2}$ of a vector $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$.

FACT 12.4. *(Hadamard's inequality) Let $A = (a_{ij})_{1\leq i,j\leq n} \in \mathbb{R}^{n\times x}$ be a square matrix, $a_i = (a_{i1}, \ldots, a_{in}) \in \mathbb{R}^{1\times n}$ its ith row, and $b \in \mathbb{R}$ such that $|a_{ij}| \leq b$ for all i, j. Then*

(i) $|\det A| \leq \prod_{1\leq i\leq n} \|a_i\|_2$,

(ii) $|\det A| \leq n^{\frac{n}{2}} b^n$.

A proof of (i) can be found in Grötschel *et al.* (1993), e.g.; the geometrical idea is that the volume $|\det A|$ of the polytope spanned by $a_1, \ldots, a_n \in \mathbb{R}^n$ is maximal when these vectors are mutually orthogonal; in this case it is the product on the right hand side of (i). Now (ii) follows easily from (i), using that $\|a_i\|_2 \leq \sqrt{n}b$ for $1 \leq i \leq n$. In Example 12.3,

$$|-58| = |\det A| \leq \sqrt{4^2 + 5^2} \cdot \sqrt{6^2 + 7^2} = \sqrt{3485} < 59.04$$

from (i), and $|-58| \leq 2 \cdot 7^2 = 98$ from (ii), as before.

In general, we have

$$\frac{n!\, b^n}{n^{n/2} b^n} \approx \left(\frac{\sqrt{n}}{e}\right)^n$$

since $n! \approx n^n e^{-n}$ by Stirling's formula. The following table shows the values of the ratio for some small values of n.

n	$n!$	$n^{\frac{n}{2}}$	$\frac{n!}{n^{n/2}}$
2	2	2	1.0
4	24	16	1.5
6	720	216	3.3
8	40320	4096	9.8
10	3628800	100000	36.3

How many prime numbers $m_1 = 2, \cdots, m_r$ are necessary? By the above,

$$m = \prod_{1\leq i\leq r} m_i > 2n^{\frac{n}{2}} b^n \geq 2|\det A|$$

is sufficient, and since $m_i \geq 2$ for $1 \leq i \leq r$,

$$r > \log_2(2n^{\frac{n}{2}} b^n) = n(\frac{1}{2}\log_2 n + \log_2 b) + 1$$

is large enough. This already achieves a polynomial bound. Can we do still better?

In analytic number theory, there is a famous theorem about the number of primes in an initial segment of the natural numbers. It may be stated in three equivalent ways:

FACT 12.5. *(Prime Number Theorem, see Rosser & Schoenfeld 1962) Denote by $\mathbb{P}$ the set of prime numbers, let $x \in \mathbb{R}_{>0}$, $n \in \mathbb{N}$, and define*

$$\begin{aligned} \pi(x) &= \#\{p \in \mathbb{P} : p \leq x\}, \\ \vartheta(x) &= \sum_{\substack{p \in \mathbb{P} \\ p \leq x}} \log p, \\ p_n &= \text{the } n\text{th prime number.} \end{aligned}$$

Then approximately $\pi(x) \approx \frac{x}{\log x}$, $\vartheta(x) \approx x$, $p_n \approx n \log n$, and more precisely

$$\begin{array}{rcccl} \frac{x}{\log x}\left(1 + \frac{1}{2\log x}\right) & < & \pi(x) & < & \frac{x}{\log x}\left(1 + \frac{3}{2\log x}\right) \text{ if } x \geq 59, \\ x\left(1 - \frac{1}{\log x}\right) & < & \vartheta(x) & < & x\left(1 + \frac{1}{\log x}\right) \text{ if } x \geq 41, \\ n \log n & < & p_n & < & n(\log n + \operatorname{loglog} n) \text{ if } n \geq 6. \end{array}$$

All logarithms are in base e.

In our case,

$$\begin{aligned} \log m &= \log \prod_{\substack{p \in \mathbb{P} \\ p \leq m_r}} p = \vartheta(m_r) \\ &\geq \vartheta(r \log r) > r \log r \left(1 - \frac{1}{\log r + \operatorname{loglog} r}\right) \\ &\geq r(\log r - 1) \end{aligned}$$

if $r \geq 16$, so that it is sufficient to choose $r \in \mathbb{N}$ minimal satisfying $r(\log r - 1) > \log(2n^{\frac{n}{2}} b^n)$.

The "naive" and the "sophisticated" lower bounds on r in the above example lead to $r > \log_2(196) \approx 7.61$ or $r \geq 8$, and $r(\log r - 1) > \log_e(196) \approx 5.28$ or $r \geq 7$, respectively, whereas $r = 4$ turned out to be sufficient.

Now we can make concrete the notion "large enough" in the first step of Algorithm 12.2 to mean

$$r(\log r - 1) > n(\frac{1}{2}\log_2 n + \log_2 b) + 1.$$

Call the latter number c for short. Note that c is polynomial in the input size, which is approximately $n^2 \log b$. If $\log b \approx n$, then $r \approx \frac{n^2}{\log n}$.

Finally, we want to analyze the cost of the algorithm, i.e., the number of bit operations. Step 1 may, e.g., be done by means of the Sieve of Eratosthenes, but we do not go into that here. Think of step 1 as a precomputation stage or assume (correctly) that its cost is negligible.

The number of bit operations for an arithmetic operation $(+, -, *, /)$ in $\mathbb{Z}/(m_i)$ is $O(\log^2 m_i)$ for classical arithmetic or $O(\log m_i (\operatorname{loglog} m_i)^2 \operatorname{logloglog} m_i)$ using fast arithmetic, and $\log m_i \le \log m_r \approx \log r \approx \log c$. So we have a cost of $O(rn^3 \log^2 c)$ or $O(n^3 c \log^2 c)$ bit operations for step 2 (r different moduli m_i, n^3 operations in $\mathbb{Z}/(m_i)$ per module m_i, and $\log^2 c$ bit operations per arithmetic operation in $\mathbb{Z}/(m_i)$).

Step 3 can be done with $O(c^2)$ bit operations (and $O(c \log^2 c \operatorname{loglog} c)$ with fast algorithms), hence the total cost for the algorithm is $O(n^3 c \log^2 c + c^2)$ bit operations, and this is polynomial in the input size.

Similarly, a modular algorithm for computing determinants of matrices with entries in $F[x]$, where F is a field, can be designed. This will be discussed in the exercises.

12.2. General scheme for modular algorithms. The modular algorithm of the previous section can be generalized in several respects: it works over any Euclidean domain for any polynomial expression.

Let (R, d) be a Euclidean domain, $R[y_1, \ldots, y_n]$ the polynomial ring in n indeterminates $y_1, \ldots, y_n$ over R, and $f \in R[y_1, \ldots, y_n]$. Suppose that we want to evaluate f at arbitrary points $(a_1, \ldots, a_n) \in R^n$, but that the direct way of "plugging in the values and evaluating" over R is for some reasons uncomfortable (e.g., slow, or hard to analyze, or possibly having large intermediate results). Then the following indirect way may be advantageous.

Choose moduli $m_1, \ldots, m_r \in R$ pairwise relatively prime, evaluate $f(a_1, \ldots, a_n)$ modulo m_i for $1 \le i \le r$, and put the results together using the Chinese Remainder Algorithm.

$$\begin{tikzcd} R^n \arrow[rrr, "\text{reduction mod } m_i \text{ for } 1 \le i \le r"] \arrow[d, "\text{direct evaluation}"] & & & (R/(m_1))^n \times \cdots \times (R/(m_r))^n \arrow[d, "\cdots\ \text{modular evaluation}"] \\ R & & & R/(m_1) \times \cdots \times R/(m_r) \arrow[lll, "\text{CRA}"] \end{tikzcd}$$

Figure 12.1: The scheme for modular algorithms.

A further generalization is to allow several polynomials $f_1, \ldots, f_k \in R[y_1, \ldots, y_n]$ to be simultaneously evaluated at the same point $(a_1, \ldots, a_n) \in R^n$. Let $D \in \mathbb{N}$ be an a priori bound such that $d(f_j(a_1, \ldots, a_n)) \le D$ for $1 \le j \le k$ (in general, D will depend on $a_1, \ldots, a_n$).

In the determinant example, we had $R = \mathbb{Z}$, $k = 1$, and

$$f_1 = f = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) y_{1\sigma(1)} \cdots y_{n\sigma(n)} \in R[y_{11}, \ldots, y_{nn}],$$

the determinant polynomial in n^2 variables. With $d(a_{ij}) = |a_{ij}| \leq b$ for $1 \leq i, j \leq n$, we could choose $D = n^{\frac{n}{2}} b^n$.

The following algorithm works for $R = \mathbb{Z}$ and $R = F[x]$, with a parameter λ that equals 1 in the polynomial and 2 in the integer case, respectively.

Algorithm 12.6.
Input: *$f_1, \ldots, f_k \in R[y_1, \ldots, y_n]$, $a = (a_1, \ldots, a_n) \in R^n$, $D \in \mathbb{N}$ as above.*
Output: *$f_1(a_1, \ldots, a_n), \ldots, f_k(a_1, \ldots, a_n) \in R$.*

1. *Choose r moduli $m_1, \ldots, m_r \in R$ pairwise relatively prime such that $d(m) > \lambda D$, where $m = \prod_{1 \leq i \leq r} m_i$.*
2. *(Reduction) For $1 \leq i \leq r$ and $1 \leq j \leq k$ set $a^{(i)} = a \bmod m_i$ and $f_j^{(i)} = f_j \bmod m_i$ (only the coefficients have to be reduced).*
3. *(Modular evaluation) For $1 \leq i \leq r$ and $1 \leq j \leq k$ compute $b_j^{(i)}$ satisfying*
$$b_j^{(i)} \equiv f_j^{(i)}(a_1^{(i)}, \ldots, a_n^{(i)}) \bmod m_i.$$
4. *(CRA) For $1 \leq j \leq k$ compute $b_j \in R$ with $d(b_j)$ miminal satisfying*
$$b_j \equiv b_j^{(i)} \bmod m_i \text{ for } 1 \leq i \leq r.$$
5. *Return $b_1, \ldots, b_k$.*

Theorem 12.7. *If R is $\mathbb{Z}$ or $F[x]$, where F is a field, then Algorithm 12.6 works correctly, i.e., $b_j = f_j(a_1, \ldots, a_n)$ for $1 \leq j \leq k$.*

Proof. Let $c_j = f_j(a_1, \ldots, a_n)$ for short. By construction, we have for any j that

$$b_j \equiv b_j^{(i)} \equiv f_j^{(i)}(a_1^{(i)}, \ldots, a_n^{(i)}) \equiv f_j(a_1, \ldots, a_n) = c_j \bmod m_i$$

for $1 \leq i \leq r$, and hence $b_j \equiv c_j \bmod m$ by the CRT, i.e., $b_j - c_j = mg_j$ for some $g_j \in R$. Assume that $g_j \neq 0$. Then $d(m) \leq d(mg_j) = d(b_j - c_j)$.

1. $R = \mathbb{Z}$. Then
$$d(b_j - c_j) = |b_j - c_j| \leq |b_j| + |c_j| \leq |b_j| + D < |b_j| + \frac{|m|}{2},$$
and furthermore $|b_j| \leq \frac{|m|}{2}$, too, by the minimality of $d(b_j) = |b_j|$. Hence
$$d(m) \leq d(b_j - c_j) < \frac{|m|}{2} + \frac{|m|}{2} = |m| = d(m),$$
which is a contradiction.

2. $R = F[x]$. Then

$$d(b_j - c_j) = \deg(b_j - c_j) \leq \max\{\deg b_j, \deg c_j\}.$$

Now $\deg c_j \leq D < \deg m$ by the choice of D, and $\deg b_j < \deg m$ by the minimality of $\deg b_j = d(b_j)$. Hence $d(m) \leq d(b_j - c_j) < \deg m = d(m)$, which is again a contradiction.

In any case we have derived a contradiction from the assumption $g_j \neq 0$, and so $b_j = c_j$. $\square$

13. Modular gcd computation

We first begin with a definition that will be technically convenient. The gcd of the polynomials f and g in $F[x]$ is only defined up to multiplication by a constant, but can be uniquely defined by specifying the leading coeffcient. Typically, one defines the *monic* gcd to be the one with leading coefficient 1; we shall define the *normalized* gcd of f and g, denoted by $\text{ngcd}(f, g)$, to be the one with leading coefficient equal to the leading coefficient $lc(f)$ of f.

We now apply the results of the previous section to the situation where R is a UFD, f and g are polynomials in $R[x]$, and K is the field of fractions of R. Our two standard examples are: $R = \mathbb{Z}$ and $K = \mathbb{Q}$, and $R = F[y]$ and $K = F(y)$.

For a polynomial $f = \sum_{i=0}^{n} a_i x^i \in R[x]$, we define the *content* of f to be $\text{cont}(f) = \gcd(a_0, a_1, \ldots, a_n)$; this is defined only up to units. A polynomial with content 1 is called *primitive* . The following two results are important but not proven in this course. They should be part of most algebra courses.

THEOREM 13.1. *(Gauss' Lemma) The product of two (or more) primitive polynomials is also primitive.*

THEOREM 13.2. *If a ring R is a UFD, then so is $R[x]$.*

Let h be the gcd of f and g in $R[x]$, and ϕ the monic gcd of f and g in $K[x]$. The following statements are consequences of Gauss' Lemma: $h = a \cdot \phi$ for some $a \in R$, and the leading coefficient $lc(h) = a$ of h divides $lc(f)$. Thus, $\text{ngcd}(f, g) = lc(f) \cdot \phi$ is in $R[x]$. The polynomial $\text{ngcd}(f, g)$ may not be primitive, but if we extract its content and f and g were primitive to start with, then we obtain the gcd of f and g in $R[x]$.

By Theorem 11.13, the Euclidean Algorithm for $\mathbb{Q}[x]$ runs in polynomial time. Together with Gauss' Lemma, this gives a polynomial time algorithm for computing gcd's in $\mathbb{Z}[x]$: To compute $h = \gcd(f, g)$ over $\mathbb{Z}[x]$, first use the Euclidean Algorithm to determine the monic gcd ϕ of f and g over $\mathbb{Q}[x]$, and then take a multiple of ϕ with content equal to the gcd of $\text{cont}(f)$ and $\text{cont}(g)$ over $\mathbb{Z}$. Similar statements are true for $F(y)[x]$ and $F[y][x]$, respectively, where F is a field.

EXAMPLE 13.3. *Let $f = 24x^2 + 4x - 4$ and $g = 12x^2 - 18x - 12$ in $\mathbb{Z}[x]$. The factorizations into irreducibles in $\mathbb{Z}[x]$ are*

$$f = 2 \cdot 2 \cdot (3x - 1) \cdot (2x + 1), \quad g = 2 \cdot 3 \cdot (2x + 1) \cdot (x - 2),$$

and hence $\text{cont}(f) = 4$, $\text{cont}(g) = 6$, $\phi = \gcd(f, g) = x + \frac{1}{2}$ *in* $\mathbb{Q}[x]$, *and*

$$h = \gcd(f, g) = 4x + 2 = \gcd(\text{cont}(f), \text{cont}(g)) \cdot (2x + 1)$$

in $\mathbb{Z}[x]$. *Furthermore* $\text{ngcd}(f, g) = 24x + 12$ *in* $\mathbb{Z}[x]$.

The next theorem is an immediate consequence of Theorem 11.6.

THEOREM 13.4. *Let $\mathcal{M}$ be a maximal ideal in R, and let F be the field $R/\mathcal{M}$. Let $a \mapsto \overline{a}$ be the residue class map from R onto F, and extend this map coefficient-wise to a homomorphism from $R[x]$ onto $F[x]$. Assume that $\overline{lc(f)} \neq 0$ and $\overline{lc(g)} \neq 0$. Let $d = \deg \operatorname{ngcd}(f, g)$.*

If $d = \deg g$, then

$$\overline{\operatorname{ngcd}(f, g)} = \operatorname{ngcd}(\overline{f}, \overline{g}).$$

If $d < \deg g$, then

$$\det \overline{P_d} \neq 0 \implies \overline{\operatorname{ngcd}(f, g)} = \operatorname{ngcd}(\overline{f}, \overline{g}),$$

$$\det \overline{P_d} = 0 \implies d < \deg \operatorname{ngcd}(\overline{f}, \overline{g}).$$

PROOF. Exercise. □

EXAMPLE 13.5. *Let $R = \mathbb{Z}$, $\overline{a} = a \bmod 17$, and $f = 12x^3 + 28x^2 + 20x + 4$ and $g = 12x^2 + 10x + 2$ be polynomials in $\mathbb{Z}[x]$. Then $\operatorname{ngcd}(f, g) = 12x + 4$ and hence $d = 1$. Computing*

$$\det P_d = \det \begin{pmatrix} 12 & 12 & 0 \\ 28 & 10 & 12 \\ 20 & 2 & 10 \end{pmatrix} = 432 = 2^4 3^3 \not\equiv 0 \bmod 17,$$

we see that $\deg \operatorname{ngcd}(\overline{f}, \overline{g}) = 1$, too. In fact, $\operatorname{ngcd}(\overline{f}, \overline{g}) = 12x + 4 \in \mathbb{F}_{17}[x]$, and obviously $\operatorname{ngcd}(\overline{f}, \overline{g}) = \operatorname{ngcd}(f, g) \bmod 17$.

EXAMPLE 13.6. *Let $R = \mathbb{Z}$, $\overline{a} = a \bmod 3$, and $f = x^4 + x^3 - x^2 + 3x + 1$ and $g = x^3 - x^2 + x$ (cf. Example 11.7). Then $\operatorname{ngcd}(f, g) = 1$ and $d = 0$. But here*

$$\det P_d = \det \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 & 1 & 0 & 0 \\ -1 & 1 & 1 & 1 & -1 & 1 & 0 \\ 3 & -1 & 1 & 0 & 1 & -1 & 1 \\ 1 & 3 & -1 & 0 & 0 & 1 & -1 \\ 0 & 1 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} = 3,$$

and hence $\deg \operatorname{ngcd}(\overline{f}, \overline{g}) > 0$. A calculation modulo 3 shows that $\operatorname{ngcd}(\overline{f}, \overline{g}) = x+1$.

EXAMPLE 13.7. *Let $R = \mathbb{Z}_5[y]$,*

$$f = (y^3 + 3y^2 + 2y)x^3 + (y^2 + 3y + 2)x^2 + (y^3 + 3y^2 + 2y)x + (y^2 + 3y + 2)$$

and

$$g = (2y^3 + 3y^2 + y)x^2 + (3y^2 + 4y + 1)x + (y + 1)$$

be polynomials in $R[x]$*. Then* $\operatorname{ngcd}(f,g) = (y^3+3y^2+2y)x + (y^2+3y+2)$ *and* $d=1$*. Furthermore,*

$$\begin{aligned}\det P_1 &= \det\begin{pmatrix} y^3+3\,y^2+2\,y & 2\,y^3+3\,y^2+y & 0 \\ y^2+3\,y+2 & 3\,y^2+4\,y+1 & 2\,y^3+3\,y^2+y \\ y^3+3\,y^2+2\,y & y+1 & 3\,y^2+4\,y+1 \end{pmatrix} \\ &= 4(y-4)^3(y-3)^2(y-1)y^3.\end{aligned}$$

If we now let $\overline{a} = a(2)$ *for* $a \in R$*, then* $\overline{\det P_1} = (\det P_1)(2) \neq 0$*, and hence* $\deg \operatorname{ngcd}(\overline{f},\overline{g}) = 1$*. Actually* $\operatorname{ngcd}(\overline{f},\overline{g}) = 4x+2 = \overline{\operatorname{ngcd}(f,g)}$*. On the other hand, if* $\overline{a} = a(1)$ *for* $a \in R$*, then* $\overline{\det P_1} = 0$ *and* $\deg \operatorname{ngcd}(\overline{f},\overline{g}) > 1$*. It turns out that* $\operatorname{ngcd}(\overline{f},\overline{g}) = x^2+3x+2$.

Theorem 13.4 and the bounds in Theorems 11.13 and 11.15 allow a modular computation of gcd's.

13.1. A modular gcd algorithm for $\mathbb{Z}[x]$. Suppose that $R = \mathbb{Z}$, and the coefficients of f and g are integers bounded by M in absolute value. Then by Theorem 11.13, the coefficients of $\operatorname{ngcd}(f,g)$ are integers bounded in absolute value by

$$H_1 = 2^n(n+1)\cdot n^n \cdot M^{2n+2},$$

and by Hadamard's inequality, $\det P_d$ is an integer bounded in absolute value by

$$H_2 = (2n)^n M^{2n}.$$

These estimates and Theorem 13.4 justify the following algorithm for computing $\operatorname{ngcd}(f,g)$:

1. Let $S \subseteq \mathbb{N}$ be a set of primes such that none of the primes in S divide $lc(f)$ or $lc(g)$, and the product of the primes in S exceeds $2H_1H_2$.

2. For each $p \in S$, compute $h_p = \operatorname{ngcd}(\overline{f},\overline{g}) \in \mathbb{Z}_p[x]$, where the bar indicates reduction of each coefficient mod p. This is done using Euclid's algorithm for $\mathbb{Z}_p[x]$.

3. Let $d = \min\{\deg h_p : p \in S\}$, and select a subset $S' \subset S$ such that $\deg h_p = d$ for all primes p in S' and the product of the primes in S' exceeds $2H_1$ (such a set S' must exist, and d as computed will equal $\deg \operatorname{ngcd}(f,g)$) .

4. At this point, for $0 \leq i \leq d$, the coefficient of x^i in $\operatorname{ngcd}(f,g)$ mod p is equal to the coefficient of x^i in h_p for all $p \in S'$, and we recover this coefficient using the Chinese Remainder Algorithm for $\mathbb{Z}$.

13.2. A modular gcd algorithm for $F[x,y]$. Suppose that $R = F[y]$, where F is a field, and f and g are polynomials in $R[x]$ whose coefficients are polynomials in R of degree at most δ. Then by Theorem 11.15, the coefficients of $\mathrm{ngcd}(f,g)$ are polynomials in R of degree at most $(2n+2)\delta$ and $\det P_d$ is a polynomial in R of degree at most $2n\delta$.

These estimates and Theorem 13.4 justify the following algorithm for computing $\mathrm{ngcd}(f,g)$:

1. Let S be a set of $(4n+2)\delta+1$ points in F such that none of the points in S is a zero of $lc(f)$ or $lc(g)$.

2. For each $c \in S$, compute $h_c = \mathrm{ngcd}(\overline{f}, \overline{g}) \in F[x]$, where the bar indicates evaluating each coefficient at $y = c$. This is done using Euclid's algorithm for $F[x]$.

3. Let $d = \min\{\deg h_c : c \in S\}$, and select a subset $S' \subset S$ of size $(2n+2)\delta+1$ such that $\deg h_c = d$ for all points c in S' (such a set S' must exist, and d as computed will equal $\deg \mathrm{ngcd}(f,g)$).

4. At this point, for $0 \le i \le d$, the coefficient of x^i in $\mathrm{ngcd}(f,g)$ evaluated at $y = c$ is equal to the coefficient of x^i in h_c for all $c \in S'$, and we recover this coefficient by using an interpolation algorithm for $F[y]$.

Remark. The input to this algorithm consists of $O(n\delta)$ elements of F, and the output consists of $O(n^2\delta)$ elements in F. The number of arithmetic operations in F performed by the algorithm (assuming classical arithmetic) is $O(n^3\delta^2)$.

14. Primality testing

The general definition of primes says for an integer $N \in \mathbb{Z}$ that it is *prime* if and only if $N \notin \mathbb{Z}^\times$ and for all $a, b \in \mathbb{Z}$, $N = ab$ implies that either $a \in \mathbb{Z}^\times$ or $b \in \mathbb{Z}^\times$, where $\mathbb{Z}^\times = \{-1, +1\}$ is the group of units in $\mathbb{Z}$.

FUNDAMENTAL THEOREM OF ARITHMETIC. *Every $N \in \mathbb{N}$ can be written as a product of positive prime factors. This is unique up to the order of the factors (i.e., $\mathbb{Z}$ is a Unique Factorization Domain).*

This section of the course deals with testing for primality. Define the set PRIMES $= \{N \in \mathbb{N} : N \text{ is prime}\}$. We will show that PRIMES $\in \mathcal{BPP}$. Two useful references are Knuth (1981) and Koblitz (1987).

Recall a few notions from complexity theory. A decision problem $D \subseteq I$ is a subset of the set I of instances. (Example: $D =$ PRIMES $\subseteq I = \mathbb{N}$.) $D \in \mathcal{P}$ if there exists a Turing machine (an idealized computer "hard-wired" to a particular program) which correctly accepts or rejects any $x \in I$ in a polynomial number of steps in the the input size $|x|$ (such a machine is called a polynomial-time Turing machine). (Example: for $x \in \mathbb{N}$, $|x| = 1 + \lfloor \log_2 x \rfloor =$ binary length of x.)

Randomized algorithms lead to the complexity class $\mathcal{BPP}$. It consists of those decision problems D for which there exists a polynomial-time Turing machine which, given an instance $x \in I$ of D and a random string of length polynomial in $|x|$, does the following. If $x \in D$, then with probability at least $2/3$ it accepts x. If $x \notin D$, then with probability at least $2/3$ it rejects x. A standard trick is to run such an algorithm k times and accept if most of the runs accept; then an element in D will be accepted with probability at least $1 - (2/3)^k$. Furthermore, $D \in \mathcal{BPP}$ if and only if its complement $I \setminus D$ is in $\mathcal{BPP}$.

We use the following facts. Let $\mathbb{Z}_N^\times = \{a \bmod N \in \mathbb{Z}_N : \gcd(a, N) = 1\}$ be the multiplicative group of units in $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$. Remember that a unit in a ring is an element that has an inverse in the ring. The elements of $\mathbb{Z}_N^\times$ form a multiplicative group of cardinality $\varphi(N) = \#\mathbb{Z}_N^\times$; φ is *Euler's totient function*. If the prime factorization of N is $N = p_1^{e_1} \cdots p_r^{e_r}$, where $p_1, \ldots, p_r$ are pairwise distinct positive primes, and $e_1, \ldots, e_r$ are positive integers, then the Chinese Remainder Theorem says that $\mathbb{Z}_N \cong \mathbb{Z}_{{p_1}^{e_1}} \times \cdots \times \mathbb{Z}_{{p_r}^{e_r}}$, and that $\mathbb{Z}_N^\times \cong \mathbb{Z}_{p_1^{e_1}}^\times \times \cdots \times \mathbb{Z}_{p_r^{e_r}}^\times$. If N is prime, then $\mathbb{Z}_N$ is a field, and $\mathbb{Z}_N^\times$ is a cyclic group of order $N - 1$. If $N = p^e$ is a prime power, then $\varphi(N) = p^{e-1}(p-1)$. Thus in general, $\varphi(N) = p_1^{e_1-1} \cdot (p_1 - 1) \cdots p_r^{e_r-1} \cdot (p_r - 1)$. A generator g for this group is a called a *primitive element* modulo N. For a prime N, $\{1, \cdots, N-1\} = \{g^0 \mathrm{mod} N, g^1 \mathrm{mod} N, \cdots, g^{N-2} \mathrm{mod} N\}$, and $g^k \equiv 1 \bmod N$ if and only if $N - 1$ divides k.

THEOREM 14.1. *(Fermat's Little Theorem) If N is prime, then for all $a \in \mathbb{Z}_N^\times$, $a^{N-1} = 1$.*

We define the *order* of an element $a \in \mathbb{Z}_N^\times$ as

$$\mathrm{ord}_N(a) = \min\{b \in \mathbb{N} : b \geq 1 \text{ and } a^b = 1\}.$$

Then

$$\mathrm{ord}_N(a) \mid \varphi(N). \tag{14.1}$$

Fermat's Little Theorem is a consequence of this, since $\varphi(N) = N - 1$ when N is prime. (14.1) in turn is a special case of *Lagrange's Theorem* from group theory which says that the cardinality of a subgroup H of a finite group G divides the cardinality of the group itself. Here H is the cyclic subgroup of $G = \mathbb{Z}_N^\times$ generated by a. Lagrange's Theorem is usually proved by showing that the map $h \mapsto hb$ for fixed $b \in G$ is a bijection from H to the right coset Hb of H, so that all right cosets of H have the same cardinality.

EXAMPLE 14.2. *For $N = 17$, we have:*

a	a^2	a^4	a^8	a^{16}
2	4	-1	1	1
3	9	13	-1	1

Thus 2 is not a primitive element modulo 17. You may check "by hand" that 3 is primitive modulo 17 (i.e., by computing all powers $3^k \bmod 17$ for $1 \le k \le 16$); alternatively, the next Theorem says that the information that $3^8 \equiv -1 \bmod 17$ is sufficient.

THEOREM 14.3. *Suppose that N is a prime, and $a \in \mathbb{Z}_N^\times$. Then*

(i) $\mathrm{ord}_N(a) \mid (N-1)$,

(ii) $\mathrm{ord}_N(a) = N - 1$ *if and only if* $a^{(N-1)/p} \neq 1$ *for all primes p dividing $N - 1$,*

(iii) there exists a primitive element in $\mathbb{Z}_N^\times$.

For (iii), see van der Waerden (1966), § 42, with $h = p - 1$.

EXERCISE 14.4. *Convince yourself that 5 is primitive modulo 17, and 4 is not. Draw the directed graph on vertices $1, \ldots, 16$ with an edge from i to j if and only if $i^2 \equiv j \bmod 17$.*

14.1. PRIMES $\in \mathcal{BPP}$. From now on, assume that $N \in \mathbb{N}$ is odd. Fermat's Little Theorem says that if N is prime, then for all $a \in \mathbb{Z}$ with $\gcd(a, N) = 1$ we have $a^{N-1} \equiv 1 \bmod N$. From this theorem, we get the *Fermat test* for primality.

1. Pick $a \in \{1, \cdots, N-1\}$ at random uniformly.
2. If $\gcd(a, N) \neq 1$, then say "composite".
3. Compute $b = a^{N-1} \bmod N$.
4. If $b \neq 1$ then say "composite", else say "prime".

Notice that if the test replies "composite", it is correct. If it replies "prime", it may be right or it may be wrong. If $a^{N-1} \not\equiv 1 \bmod N$ and $\gcd(a, N) = 1$, then a is called a *Fermat witness* to the compositeness of N. If we know any such witness, we are guaranteed that N is composite. The two required properties are easy to check. Here is a famous historical example: $2^1 + 1$, $2^2 + 1$, $2^4 + 1$, $2^8 + 1$, and $2^{16} + 1$ are all prime. Pierre Fermat conjectured that $2^{32} + 1$ is a prime in the year 1640, the same year that he proved his famous "Little" theorem. However, $3^{2^{32}} \not\equiv 1 \bmod 2^{32} + 1$; therefore, by Fermat's Little Theorem, Fermat's conjecture is false. The required computation amounts to 32 multiplications modulo the 10-digit number $2^{32} + 1 = 4294967297$. Had Fermat thought of doing this, he could have closed his conjecture within a day. It turns out that for "most" composite N, most a's are Fermat witnesses.

It would be nice if this test were enough. However, there are composite numbers (the *Carmichael numbers*) without any Fermat witnesses. But a small modification makes the test work in any case: just test whether "$\sqrt{a^{N-1}}$ is $\sqrt{1}$" in the right way.

We first collect some facts that will be used for our primality test, and also later for factoring polynomials.

FACT 14.5. *(i) Let p be a prime. Then $\mathbb{Z}_p$ is a field with p elements.*

(ii) If F is a field, $f \in F[x]$, and $a \in F$, then $f(a) = 0 \iff x - a \mid f$.

(iii) If F is a field and $f \in F[x]$, then f has at most $\deg f$ many roots in F.

(iii) is not true in rings: $f = x^2 \in \mathbb{Z}_{16}[x]$ has the four roots $0 \bmod 16$, $4 \bmod 16$, $8 \bmod 16$, $12 \bmod 16$.

LEMMA 14.6. *Let p be an odd prime, and*

$$S = \{a \in \mathbb{Z}_p^\times : \exists b \in \mathbb{Z}_p^\times \ a = b^2\}$$

be the set of squares in $\mathbb{Z}_p^\times$. Then

(i) $S \subseteq \mathbb{Z}_p^\times$ is a (multiplicative) subgroup of order $(p-1)/2$,

(ii) $S = \{a \in \mathbb{Z}_p^\times : a^{(p-1)/2} = 1\}$,

(iii) $\forall a \in \mathbb{Z}_p^\times \quad a^{(p-1)/2} \in \{1, -1\}$.

PROOF. (i) Since $b_1^2 \cdot b_2^2 = (b_1 b_2)^2$, S is a subgroup of $\mathbb{Z}_p^\times$. We have the squaring homomorphism

$$\sigma : \mathbb{Z}_p^\times \longrightarrow S$$

with $\sigma(b) = b^2$. By definition, σ is a surjective homomorphism of multiplicative groups, and the kernel of σ is

$$\ker \sigma = \{a \in \mathbb{Z}_p^\times : \sigma(a) = 1\} = \{a \in \mathbb{Z}_p^\times : a^2 = 1\} = \{1, -1\}.$$

In the last equality, we have used that $1, -1 \in \ker\sigma$, and that $\#\ker\sigma \leq 2$ by Fact 14.5 (iii). It follows that

$$p-1 = \#\mathbb{Z}_p^\times = \#\ker\sigma \cdot \#\mathrm{im}\sigma = 2 \cdot \#S.$$

(ii) Let

$$T = \{a \in \mathbb{Z}_p^\times : a^{(p-1)/2} = 1\}$$

be the right-hand side in (ii). For any $a \in S$, there exists $b \in \mathbb{Z}_p^\times$ with $a = b^2$, and then

$$a^{(p-1)/2} = b^{p-1} = 1,$$

by Fermat's Little Theorem, and thus $S \subseteq T$. On the other hand, $\#T \leq (p-1)/2$ by Fact 14.5 (iii), so that $S = T$.

(iii) follows from the fact that $(a^{(p-1)/2})^2 = 1$ for all $a \in \mathbb{Z}_p^\times$, and that 1 and -1 are the only square roots of 1, by Fact 14.5. □

Let $N \in \mathbb{N}$ be odd, $n \in \mathbb{N}$, and $\pi_n : \mathbb{Z}_N^\times \to \mathbb{Z}_N^\times$ the powering function, with $\pi_n(a) = a^n$ for $a \in \mathbb{Z}_N^\times$. Furthermore, let $\pm 1 = \{1, -1\} \subseteq \mathbb{Z}_N^\times$.

Lemma 14.7. *If N is not a prime power, then*

$$\mathrm{im}\pi_n \neq \pm 1.$$

Proof. By assumption, there exist $u, v \in \mathbb{N}$ with $u, v \geq 2$, $N = uv$ and $\gcd(u, v) = 1$. Suppose that $-1 \in \mathrm{im}\pi_n$, and $a \in \mathbb{Z}$ with $a^n \equiv -1 \bmod N$. Then $a^n \equiv -1 \bmod u$, and by the Chinese Remainder Theorem, there exists $b \in \mathbb{Z}$ such that $b \equiv a \bmod u$ and $b \equiv 1 \bmod v$. Then $(b^n \bmod N) \notin \pm 1$. □

Let $T_N = \mathrm{im}\pi_{(N-1)/2} \subseteq \mathbb{Z}_N^\times$.

Theorem 14.8. *Let $N \in \mathbb{N}$ be odd. Then T_N is a subgroup of $\mathbb{Z}_N^\times$, and $T_N = \pm 1$ if and only if N is prime.*

Proof. As image of $\mathbb{Z}_N^\times$ under a group homomorphism, T_N is a group. Using Lemma 14.6 (iii) and Lemma 14.7, the only remaining case is where $N = p^e$ is a power of a prime p, with $e \in \mathbb{N}$ and $e \geq 2$. Suppose that $T_N = \pm 1$. Then $\mathrm{im}\pi_{N-1} = \{1\}$. Let $a = 1 + p^{e-1} \in \mathbb{Z}_N^\times$. Then $a^n = 1 + np^{e-1}$ for $n \in \mathbb{N}$, and

$$a^n \neq 1 \text{ for } 1 \leq n < p, \quad a^p = 1.$$

Thus the order of a is p. By the above, also $a^{N-1} = 1$, so that p divides $N-1 = p^e - 1$. This contradiction shows that $T_N \neq \pm 1$. □

In fact, $\mathbb{Z}_{p^e}^\times$ is cyclic of order $p^{e-1}(p-1)$, and any $a \in \mathbb{Z}_{p^e}^\times$ with order divisible by p would do in the above proof.

The following algorithm, due to Lehmann (1982), results from the above discussion.

ALGORITHM 14.9. *Lehmann's primality test—uu.*
Input: *An odd integer N, and a confidence parameter $k \geq 2$.*
Output: *"prime" or "composite".*

1. *Choose $a_1, \ldots, a_k \in \{1, \ldots, N-1\}$ at random uniformly.*
2. *Compute $g_i = \gcd(a_i, N)$ for $1 \leq i \leq k$. If $g_i \neq 1$ for some i, then return "composite".*
3. *Compute $b_i = a_i \bmod N \in \mathbb{Z}_N^\times$ and $c_i = b_i^{(N-1)/2} \in \mathbb{Z}_N^\times$ for $1 \leq i \leq k$.*
4. *If $\{c_1, \ldots, c_k\} \neq \{1, -1\}$, then return "composite", else return "prime".*

THEOREM 14.10. *If N is prime, the algorithm returns "prime" with probability $1 - 2^{-k+1}$. If N is composite, the algorithm returns "composite" with probability at least $1 - 2^{-k}$. The algorithm can be performed with $O(k\mathsf{M}(n)n)$ or $O(k \cdot n^2 \log n \operatorname{loglog} n)$ bit operations, where $n = \log_2 N$.*

PROOF. First suppose that N is prime. Then step 2 will never return composite. Let $S \subseteq \mathbb{Z}_N^\times$ be the set of squares, and $b \in \mathbb{Z}_N^\times$. Then

$$b \in S \Longrightarrow b^{(N-1)/2} = 1,$$
$$b \in \mathbb{Z}_N \backslash S \Longrightarrow b^{(N-1)/2} = -1,$$

by Lemma 14.6 for $p = N$. Therefore the algorithm will return "composite" if and only if $b_1, \ldots, b_k \in S$ or $b_1, \ldots, b_k \in \mathbb{Z}_N \backslash S$; each of these happens with probability 2^{-k}, giving a total error probability $2 \cdot 2^{-k}$.

Next suppose that N is composite. If $\pm 1 \not\subseteq T_N$, i.e., $-1 \notin T_N$, then the algorithm returns "composite". So now we may assume that $\pm 1 \subseteq T_N$. Since $T_N \neq \pm 1$ by Theorem 14.8, ± 1 is a proper subgroup of order 2, and $\# T_N \geq 4$. Since $\pi_{(N-1)/2}$ is a group homomorphism, each element of T_N occurs equally often as an image under $\pi_{(N-1)/2}$. Thus in any case

$$\operatorname{prob}(\pi(b) \in \pm 1) \leq 1/2$$

for random $b \in \mathbb{Z}_N^\times$, and "prime" is returned with probability less than 2^{-k}.

Each execution of step 2 takes $O(\mathsf{M}(n) \log n)$ bit operations, and each execution of step 3 $O(n\mathsf{M}(n))$ operations. □

14.2. Finding large primes. In applications such as cryptography, one wants to find large primes, say with n bits, where n is given. We may think of n as somewhere between 100 and 2000. Let $\pi(x)$ be the number of primes at most x. The Prime Number Theorem says that $\pi(x)$ is asymptotically $x/\log x$, and for example

$$\frac{x}{\ln x} < \pi(x) < \frac{x}{\ln x}\left(1 + \frac{3}{2 \ln x}\right)$$

for $x \geq 17$ (Rosser & Schoenfeld 1962). (ln is the natural logarithm.) Thus a random integer near x is prime with probability about $(\ln x)^{-1}$. If we choose random n-bit integers and test them for primality, we will find a prime after an expected number of about $n \cdot \ln 2$ trials.

14.3. Other types of primality tests. Here is an overview of this important problem in *computational number theory.* We do not give proofs. The following are useful survey articles on this area: Bach (1990), Lenstra *et al.* (1990), Lenstra & Lenstra (1993), and Adleman (1994).

Probabilistic primality tests

- Solovay and Strassen (1974, published 1977). This was the first (probabilistic) polynomial time test, and aroused widespread interest in the power of randomized algorithms.
- Miller (1976), Rabin (1980).

These two algorithms actually show that the composite numbers are in the complexity class $\mathcal{RP}$. It consists of those decision problems D for which there exists a polynomial-time Turing machine which, given an instance $x \in I$ of D and a random string of length polynomial in $|x|$, accepts x if $x \in D$ with probability at least $1/2$, while if $x \notin D$ always rejects x. The difference with the definition of $\mathcal{BPP}$ is that our machine is again allowed to make mistakes in rejecting instances in D, but is not allowed to accept instances not in D. A standard trick is to run such an algorithm k times and accept if one of the runs accepts; then an element in D will be accepted with probability at least $1 - 1/2^k$. Furthermore, we define co-$\mathcal{RP}$ to consist of those problems D whose complement $I \setminus D$ is in $\mathcal{RP}$.

For completeness, here is the test by Miller and Rabin.

ALGORITHM 14.11. *Miller-Rabin test.*
Input: N *odd.*
Output: *"prime" or "composite"*

```
Let N − 1 = 2^h · m, m odd.  Choose a ∈ {1, ..., N − 1} at random.
If a^(N−1) ≢ 1 mod N, then
     say composite
else
     Let j be the smallest index such that a^(2^j m) ≡ 1 mod N
     If j = 0 or a^(2^(j−1) m) ≡ −1 mod N then
          say prime
     else
          say composite
```

If N is composite, then a number $a \in \{1, \ldots, N-1\}$ such that this algorithm outputs **composite** is called a *Miller-Rabin witness* for the compositeness of N.

THEOREM 14.12. *If N is prime, the answer of the Miller-Rabin test is correct. If N is composite, the answer is correct with probability $\geq \frac{3}{4}$. The algorithm requires $O(\log^3 N)$ bit operations.*

Both the Solovay-Strassen and the Miller-Rabin tests should properly be called *compositeness tests* because they show PRIMES∈co-$\mathcal{RP}$ and COMPOSITES∈$\mathcal{RP}$, but the wrong terminology has stuck.

Primality tests for special numbers. Fermat (1640) believed that $F_n = 2^{2^n}+1$ were prime for $n \geq 1$. Pepin (1877) obtained the following primality test for Fermat numbers:

$$F_n \text{ prime } \iff 3^{(F_n-1)/2} \equiv -1 \bmod F_n.$$

Another interesting fact is that if p is a prime dividing F_n, then p is of the form $p = a \cdot 2^{n+2}+1$ for some integer a. Trying divisors of this form led to the large factor $5 \cdot 2^{1947}+1$ of F_{1945} (see Sierpinski, *Elementary Theory of Numbers* for the details, and Hardy & Wright (1962), 2.5, for the example F_5.)

The Mersenne numbers are given by the formula $M_n = 2^n - 1$ for $n \geq 1$. A necessary condition for M_n to be prime is that n itself be prime, for if $n = ab$ with $a, b > 1$, then $2^a - 1$ and $2^b - 1$ are non-trivial factors of M_n. There is a special test for these numbers called the Lucas-Lehmer test The test says that $2^n - 1$ is prime if and only if $L_{n-1} \equiv 0 \bmod 2^n - 1$, where L_i is recursively defined by $L_1 = 4, L_i = L_{i-1}^2 - 1, i \geq 1$. Presently it is not known if there are infinitely many Fermat primes or infinitely many Mersenne primes. No new Fermat primes have been discovered since Fermat's time, while modern computers and algorithms have enlarged the list of Mersenne primes to 30, M_{216091} being currently the largest known Mersenne prime. Pomerance and Wagstaff (1983) have conjectured that

$$\#\{p < x \mid M_p \text{ is prime}\} \sim e^{\gamma} \log_2 x,$$

where $\gamma = 0.5772157\ldots$ is Euler's constant.

Randomness vs. ERH. Probabilistic algorithms like the Lehmann or Miller-Rabin test cannot be accurately implemented yet because there is no known source of truly random numbers to be used in today's computers. However, these algorithms work fine in practice with *pseudo-random number generators*; the most popular ones are the linear!congruential generators. Knuth (1981), chapter 3 gives an extensive discussion of this topic. Miller made the observation that we can replace the assumption of randomness by the assumption of the Extended Riemann Hypothesis (ERH).

The Riemann Hypothesis is a well-studied mathematical conjecture (for more than 100 years) but it has not yet been proved. If correct, many statements we can say about numbers get more precise. For us, the following consequence is relevant.

THEOREM 14.13. *(Ankeny 1952) Assuming the ERH, for each composite N there exists a Miller-Rabin witness a with*

$$a = O(\ln^2 N).$$

This result proves the existence of a deterministic algorithm for primality testing but does not enable the construction of such an algorithm. A later result by Bach (1985) showed that, under the ERH, there exists a Miller-Rabin witness a with

$$a \leq 2 \ln^2 N.$$

Thus, the assumption of ERH produces a deterministic algorithm for primality testing which runs in polynomial time.

All reasonably small numbers have small Miller-Rabin witnesses:

THEOREM 14.14. *(Pomerance* et al. *1980) For all composite $N \leq 25 \times 10^9$ (except for $N = 3\,215\,031\,751$), at least one of 2,3,5 and 7 is a Miller-Rabin witness.*

EXERCISE 14.15. *Prove that for the exceptional N, 11 is the smallest Miller-Rabin witness.*

Elliptic curves. (H. W. Lenstra, Goldwasser & Kilian). The tests discussed so far perform some computations in the multiplicative group of units modulo N. The basic idea of this new test is to use different types of groups. *Elliptic curves* are curves in the sense of algebraic geometry, defined by an equation of the form $y^2 = x^3 + ax + b$ for some a, b. They carry a natural group structure, and the same ideas as in the other tests can be used. Lenstra introduced the elliptic curve method for factoring integers; Goldwasser and Kilian combined an approach by Pratt (1975) that showed PRIMES$\in \mathcal{NP}$ for primality proving with this new technique. The advantage is not being tied to a specific group like $\mathbb{Z}_N^\times$, but to have many groups at one's disposal. Goldwasser & Kilian (1986) used this method to produce a primality test that ran in expected polynomial time under a *special* hypothesis.

Let $\pi(x)$ be the number of primes up to x. The prime number theorem states that

$$\pi(x) \approx \frac{x}{\ln x}.$$

The special assumption used by Goldwasser and Kilian was that "short intervals contain at least the expected number of primes", namely

$$\exists c \quad \forall x \quad \pi(x + \sqrt{x}) - \pi(x) \geq \frac{\sqrt{x}}{(\ln x)^c}.$$

This is known as Cramér's Conjecture. If it is true, the number of bit operations of their algorithm is

$$\mathrm{O}\left((\log N)^{9+c}\right).$$

PRIMES $\in \mathcal{RP}$. Adleman & Huang (1987) give a probabilistic primality test that runs in expected polynomial time and which shows that PRIMES $\in \mathcal{RP}$. The previous results by Solovay-Strassen and Miller-Rabin showed that PRIMES $\in$ co-$\mathcal{RP}$. Thus PRIMES $\in \mathcal{RP} \cap$ co-$\mathcal{RP} = \mathcal{ZPP}$, and in fact this is one of only two problems in $\mathcal{RP} \cap$ co-$\mathcal{RP}$ which are not known to be in $\mathcal{P}$. (The other one is the problem of deciding whether a given polynomial over a finite field induces a permutation.)

The test of Adleman and Huang does not seem to be practical. Atkin proposed a modification to this method which uses a structure from elliptic curves called "complex multiplication". This was first implemented by Bosma, 1985,and later by Atkin & Morain (1993). Atkin and Morain can routinely show the primality of 100 digit primes on a SUN3 in five minutes, and of primes with 1500 digits in several months, using a network of a few dozen workstations.

As an example, Morain proved that

$\frac{10^{1031}-1}{9} =$
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111
11111111111

is a prime.

The method introduced by Goldwasser and Kilian produces a certificate of primality which is easy to check. The method used by Morain and a different method, based on Jacobi sums, do not have this property. To verify, the test must be repeated.

For the time bound, the analysis of Goldwasser and Kilian's method requires the use of a hypothesis (which is probably correct). However, the correctness of the certificate produced is independent of the hypothesis.

Deterministic primality tests

- *Jacobi sums* (Cohen & Lenstra 1984). This method is a generalization of the Fermat test. Adleman *et al.* (1983) proved the best unconditional deterministic result known today, a primality test with $(\log N)^{O(\log\log\log N)}$ bit operations.

  Implementations and improvements on this method were done by

- Cohen and A.K. Lenstra (1987) and
- Bosma and van de Hulst (1990)

  The largest prime confirmed using this method is $391581 \cdot 2^{21693} - 1$ which has 65087 digits. This algorithm is good for testing primes of a certain form $ab^c \pm d$, with b very small (say $b \leq 12$), a, d small, so that d is about the input size. If $a = 1$, such a number is called a Cunningham number, after the British Army officer who published in 1925 a list of such factorizations. Today, the *Cunningham project* is a regularly revised list of the Cunningham numbers with the "most wanted factorizations" (see Brillhart *et al.* 1988). For general primes of up to about 1,000 decimal digits, primality can be proven in reasonable time (say, a few months on a network of a few thousand workstations).

14.4. Factoring integers. Once we know that a number N is composite, we may want to find its prime factors. This seems, as far as we know, a much more difficult task than determining compositeness. Let $n = \lfloor \log_2 N \rfloor + 1$ be the input length. Here are some factorization methods. All methods, except the first one, are probabilistic. We ignore factors of $\log n$ in the expected running times, i.e., the number of bit operations. For some of the algorithms, the running times have not been rigorously proven, but depend on some unproven but plausible number-theoretic assumptions.

- Trial division. Time $2^{n/2}$.
- Pollard's Rho-method. Time $2^{n/4}$.
- Continued fraction method, quadratic sieve. Time $2^{O((n \log n)^{1/2})}$.
- Elliptic curves (H.W. Lenstra). Same time as for quadratic sieve, except that $n = \log p$ for the smallest prime factor p of N.
- Number field sieve (A.K. Lenstra, H.W. Lenstra, Manasse, Pollard 1990).

$$2^{O((n \log^2 n)^{1/3})}.$$

Note that "polynomial time" would mean $2^{O(\log n)}$ operations.

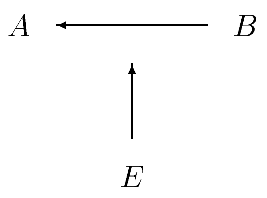
Universität-GH Paderborn
FB 17, AG Algorithmische Mathematik

Skript Computeralgebra I, Teil 5

Wintersemester 1995/96

15. Cryptography

In this section, we discuss the following scenario. *A*lice wants to send a message to *B*ob in such a way that an eavesdropper (*E*ve) listening to the transmission channel cannot understand the message. This is done by enciphering the message so that only Bob, possessing the right *key*, can decipher it, but Eve, having no access to the key, has no chance to recover the message.



The following are some of the ciphers that have been used in history.

- The *Caesar* cipher, which simply permutes the alphabet. The classical Caesar cipher used the cyclic shift by three letters $A \mapsto D$, $B \mapsto E$, $C \mapsto F$, ..., $Y \mapsto B$, $Z \mapsto C$. For example, the word "CAESAR" is then enciphered as "FDHVDU". The key in this simple cryptosystem is the permutation; its inverse is used to decipher an encrypted message. However, if the eavesdropper knows in what language the original message was and thus knows the (approximate) probabilities for individual letters to occur in an average text in that particular language, she may easily recover the message without knowledge of the key by performing a frequency analysis—provided the message is long enough. Thus this cipher is a convenient but highly insecure method.

- The *one-time pad*, which works as follows. To encipher a message of length n over the usual alphabet with 26 elements, a random vector of n letters is chosen as a key and letterwise added modulo 26 to the message. The recipient then deciphers the encrypted message by letterwise subtracting the key. For example, if the message is "CAESAR", the key is "DOHXLG", and we identify the letters A,...,Z with the numbers $0, \ldots, 25$, then the ciphertext is "FOLPLX".

  The one-time pad is the only known provably secure cryptosystem (in an information theoretic sense), but has the disadvantage that the keys are of the same length as the message (keys can not be reused without a loss of security). It is an inconvenient but highly secure method. In a more practical variant, a *pseudo-random number generator* is used to generate keys. Both Alice and Bob then use the same kind of generator parameterized by a *seed*, so that the two generators produce the same sequence when the same seed is used. Popular examples of such generators are the *linear congruential generators*. They are the basis for the `rand` functions in most hardware/software systems. They generate an ultimately periodic sequence of numbers $x_0, x_1, x_2, \ldots \in \mathbb{N}$ related by $x_{i+1} \equiv ax_i + b \bmod m$ and $x_i < m$ for $i \geq 0$, with the four values $x_0, a, b, m \in \mathbb{N}$ as a seed (which should be appropriately chosen so that the period is large enough). Then only the relatively short seed needs to be transmitted over a secure channel, e.g., by means of a courier, and the keys are consecutive parts of the pseudo-random number sequence $(x_i)_{i\geq 0}$ in a suitable encoding as vectors over the message alphabet. For example, if we assume at the moment that $m = 26$ and Alice and Bob want to exchange three messages of length n_1, n_2, and n_3, then the key for the first message is $x_0, \ldots, x_{n_1-1}$, for the second $x_{n_1}, \ldots, x_{n_1+n_2-1}$, and for the third $x_{n_1+n_2}, \ldots, x_{n_1+n_2+n_3-1}$. In this way, the generators at both parties stay synchronized and the same keys are used for both encryption and decryption without a further key exchange between Alice and Bob after the initial agreement on the seed.

It has turned out, however, that linear congruential generators are badly suited for cryptographical purposes since they are amenable to so-called "short vector" attacks (see course notes "Computeralgebra II"). Today there are other types of pseudo-random number generators where no such attacks are known. The variant of the one-time pad using pseudo-random number generators no longer has the property of provable security; its security heavily depends on the hardness of determining the elements of the pseudo-random sequence without knowledge of the seed.

- In World War II, the German Wehrmacht (army) and Marine (navy) used an enciphering machine called Enigma. This was a mechanical device where the keys were on cylinders; the underlying cipher was somewhat similar to the Caesar cipher but vastly more complicated. Its breaking by a British intelligence group at Bletchley Park under the famous computer scientist Alan Turing was apparently decisive for the Allied victory in the North Atlantic submarine war.

The main applications for cryptosystems used to be in military and secret services. Today, they are employed for all kinds of secure electronic data processing like passwords, point-of-sale registers, banking machines, and electronic cash.

All of the classical cryptosystems are *symmetric* in the sense that the same key is used for both enciphering and deciphering, or at least the decryption key is easy (i.e., in polynomial time) to infer from the encryption key. A problem with this approach is that the number of keys grows quadratically with the number of parties if each party needs to communicate with each other party.

In classical cryptography, there never was a clear mathematical understanding of what "difficult to break" means; one could only take it to mean "the tricks that I know are not sufficient to break it". The security of a cryptosystem depends on the eavesdropper's cryptanalytic skills and her knowledge about the system. In the age of Caesar, it may have been reasonable to assume that the cryptanalyst has only a very limited amount of computing power and that the encryption method may be kept secret from her. In the 20th century however, designers of cryptosystems have to take into account that potential eavesdroppers have high mathematical intelligence, access to supercomputers, complete knowledge about the encryption method except the keys, or can even channel in arbitrary parts of plaintext so that they have many plaintext/ciphertext pairs encrypted with the current key.

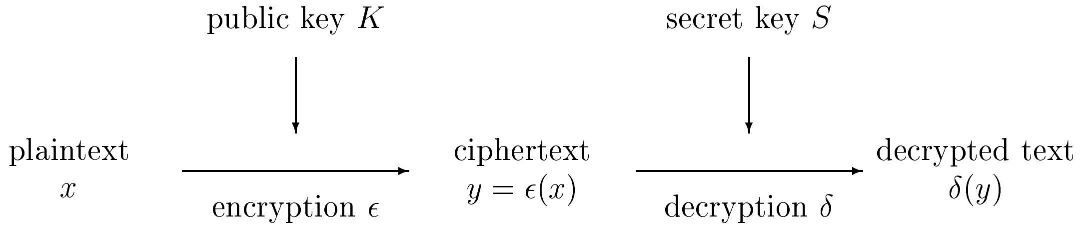


Figure 15.1: A public key cryptosystem

Diffie & Hellman (1976) made a revolutionary proposal which has since then been known as *public key cryptography*. The idea is to have two different keys K and S for encryption and decryption, respectively, such that both encryption and decryption are "easy" but decryption without knowledge of S is "hard". Here "easy" means polynomial time, preferably almost linear or quadratic time in the message length. Figure 15.1 illustrates the situation. The name "public key cryptography" comes from the fact that the encryption key may be available publicly without effecting the security of the cryptosystem. A function that is "easy" but its inverse is "hard" to compute without additional knowledge, like the encryption function in a public key cryptosystem, is also called a *trapdoor function*. The keys K and S are also called *public key* and *private key*, respectively. With such an *asymmetric* cryptosystem, n public-private key pairs are sufficient to permit secure communications among any two of n parties. An excellent introduction to this topic is Koblitz (1987).

There are several possibilities to make precise what "hard" means. Here is a list of some, ordered in increasing desirability.

- The inventor of the cryptosystem does not know of any polynomial time algorithm.
- Nobody knows of a polynomial time algorithm.

- ○ Whoever breaks the system will probably in turn have solved a well-studied "hard" problem.
- ○ Whoever breaks the system has in turn solved a well-studied "hard" problem.
- ○ Whoever breaks the system has in turn solved an $\mathcal{NP}$-complete problem.
- ○ There is probably no polynomial time algorithm.

At present, nobody knows of a cryptosystem fulfilling any of the last three requirements. However, it was a major conceptual breakthrough of the Diffie & Hellman proposal that the hitherto elusive notion of a "hard-to-break cipher" should be studied within the well-established framework of computational complexity.

Some of the modern proposals for cryptosystems have already been broken. In their original paper, Diffie & Hellman suggested a cryptosystem based on a subproblem of the $\mathcal{NP}$-complete *knapsack problem*. This was broken by Odlyzko, Adleman, and Shamir using short vector methods; it turned out that the subproblem was *not* $\mathcal{NP}$-complete. Another cipher by Cade was based on the assumed hardness of the functional decomposition problem for polynomials: Given a polynomial f over a field F of degree n, compute nonconstant polynomials $g, h \in F[x]$ (if they exist) such that $f = g \circ h = g(h)$. The system was broken by Kozen & Landau (1989), who gave an algorithm for the problem with running time $O(n^3)$. Later, von zur Gathen (1990) found an algorithm for the same problem using time $O^{\sim}(n)$.

We now present some modern public key cryptosystems.

15.1. The RSA cryptosystem.. This system is due to Rivest, Shamir, and Adleman (1978) and is based on the assumed hardness of factoring integers. The idea is that Alice randomly chooses two large (say 100-digit) primes $p \neq q$, and sets $N = pq$. Messages are encoded as sequences of elements of $\mathbb{Z}_N = \{0, \ldots, N-1\}$. If, e.g., we use the standard alphabet $\Sigma = \{A, \ldots, Z\}$ of cardinality $\#\Sigma = 26$, then messages of length up to $141 = \lfloor \log_{26} 10^{200} \rfloor$ can be uniquely represented by using the 26-adic representation. For example, the message "CAESAR" is encoded as

$$2 \cdot 26^0 + 0 \cdot 26^1 + 4 \cdot 26^2 + 18 \cdot 26^3 + 0 \cdot 26^4 + 17 \cdot 26^5 = 202302466 \in \mathbb{Z}_N.$$

If Alice wants to receive messages from Bob, she randomly chooses $e \in \{2, \ldots, \varphi(N)-2\}$ with $\gcd(e, \varphi(N)) = 1$, where φ is Euler's totient function and $\varphi(N) = \#\mathbb{Z}_N = (p-1)(q-1)$. The she computes $d \in \{2, \ldots, \varphi(N)-2\}$ with $de \equiv 1 \bmod \varphi(N)$, using the Extended Euclidean Algorithm, publishes the pair $K = (N, e)$ as her public key, and keeps her private key $S = (N, d)$ as well as p, q secret (the latter may even be discarded). The encryption and decryption functions $\epsilon, \delta : \mathbb{Z}_N^\times \longrightarrow \mathbb{Z}_N^\times$ are defined by $\epsilon(x) = x^e$ and $\delta(y) = y^d$. To send a message $x \in \mathbb{Z}_N^\times$ to Alice, Bob looks up her public key, computes $y = \epsilon(x)$, and sends this to Alice, who computes $\delta(y)$, using her private key. Then, with $u \in \mathbb{Z}$ such that $ed - 1 = u \cdot \varphi(N)$, we have

$$(\delta \circ \epsilon)(x) = \delta(x^e) = x^{ed} = x^{1+u\cdot\varphi(N)} = x(x^{\varphi(N)})^u = x,$$

since $x^{\varphi(N)} = 1$ by Lagrange's Theorem. Although the latter is only valid if $\gcd(x, N) = 1$, it can be shown that $(\delta \circ \epsilon)(x) = x$ is true for all x. However, values of x that are not relatively prime to N lead to the factorization of N and thus to a break of the cryptosystem. Fortunately, if p and q are large and we assume that all messages x are equally likely, this will practically never occur.

THEOREM 15.1. *(i) Factoring N is polynomial-time reducible to the computation of $\varphi(N)$.*

(ii) Factoring N is polynomial-time reducible to the computation of $d \in \mathbb{N}$ with $de \equiv 1 \bmod \varphi(N)$ from $K = (N, e)$.

Since the computation of $\varphi(N)$ and d from (N, e) are both trivially reducible to the factorization of N, all three problems are polynomial-time equivalent. Unfortunately, the theorem does *not* say that breaking the system means that one can factor integers, since there might be a successful attack that does not compute the secret key at all.

The RSA scheme can also be used for *authentication*, where the sender of a message has to prove that he actually is the originator. This is also called a *digital signature*. If Bob wants to send a signed message x to Alice, he computes $y = \delta(x)$ using his secret key, and sends this to Alice, who looks up Bob's public key and recovers $x = \epsilon(y)$. Since only Bob is assumed to know his secret key, no forger would have been able to produce y, and Alice is convinced that the message stems from Bob.

The authentication scheme may even be used in conjunction with the encryption scheme ensuring privacy. If ϵ_A, δ_A and ϵ_B, δ_B are Alice's and Bob's encryption and decryption functions, respectively, and Bob wants to send a signed message x to Alice that no one else can decipher, he computes $y = \epsilon_A(\delta_B(x))$, and sends this to Alice, who first decrypts $\delta_A(y) = \delta_B(x)$ and then assures herself that the message originates from Bob by applying ϵ_B.

15.2. The Diffie-Hellman key exchange protocol.. This is from Diffie & Hellman (1976) and is merely a scheme for two parties to agree upon a common key for future communication with a symmetric cryptosystem rather than a public key cryptosystem . An example might be the seed of a pseudo-random generator to be used for a one-time pad. Let $q \in \mathbb{N}$ be a "large" prime power. Then $\mathbb{F}_q^\times$, the multiplicative group of the finite field $\mathbb{F}_q$, is (non-canonically) isomorphic to the cyclic (additive) group $\mathbb{Z}_{q-1}$. If g is any generator of $\mathbb{F}_q^\times$, then $g^i \leftrightarrow i$ is an isomorphism. The protocol works as follows.

(i) Alice secretly chooses $a \in \mathbb{Z}_{q-1}$ at random, computes $u = g^a \in \mathbb{F}_q^\times$, and sends u to Bob.

(ii) Bob secretly chooses $b \in \mathbb{Z}_{q-1}$ at random, computes $v = g^b \in \mathbb{F}_q^\times$, and sends v to Alice.

(iii) Alice computes $g^{ab} = v^a$.

(iv) Bob computes $g^{ab} = u^b$.

Now both parties may use g^{ab} as a common key for further communication with a symmetric cryptosystem. In this context, the following two problems play a central role.

Problem 15.2. *(Diffie-Hellman problem, DH) Given $g^a, g^b \in \mathbb{F}_q^\times$, compute g^{ab}.*

Problem 15.3. *(Discrete logarithm problem, DL) Given $g^a \in \mathbb{F}_q^\times$, compute $a \in \mathbb{Z}_{q-1}$.*

It is conjectured that DL is a "hard" problem, although it is not known whether DL is $\mathcal{NP}$-complete. A potential eavesdropper knowing q, g and the transmitted u, v but not a or b has to solve DH to find out g^{ab}, which in turn is polynomial-time reducible to DL. It is, however, not clear whether DL is polynomial-time reducible to DH.

The currently best known algorithm for the computation of discrete logarithms in $\mathbb{F}_q^\times$ uses $2^{O(\sqrt{n \log n})}$ bit operations, where $n = \log_2 q$ is the length of a description for $\mathbb{F}_q$. For $q = 2^{539}$, e.g., we have $2^{\sqrt{593 \log_2 593}} > 10^{70}$, and this is much bigger than the age of the universe in nanoseconds, which amounts to approximately 10^{10} years, or $3 \cdot 10^{20}$ nanoseconds.

15.3. The El Gamal cryptosystem.. As before, $\mathbb{F}_q^\times$ is a "large" finite field with q elements, and g is a generator of $\mathbb{F}_q^\times$. To receive messages from Bob, Alice randomly chooses $S = b \in \mathbb{Z}_{q-1}$ as her secret key and publishes $K = (q, g, g^b)$. If Bob wants to send a message x to Alice, he looks up her public key, randomly chooses $k \in \mathbb{Z}_{q-1}$, computes g^k and xg^{kb}, and sends $y = (u, v) = (g^k, xg^{kb})$ to Alice, who computes the original message as $x = v/u^b$. Breaking the cryptosystem is obviously polynomial-time equivalent to the Diffie-Hellman problem.

A practical problem in implementing the Diffie-Hellman scheme or the El Gamal system is that exponentiation in $\mathbb{F}_q^\times$ is theoretically easy ($O(n^3)$ bit operations for $q = 2^n$ using classical arithmetic), but not fast enough to achieve high transmission rates. One can, however, achieve time $O^\sim(n^2)$ by using Gauß periods, normal bases, and fast multiplication in $\mathbb{F}_{2^n}$.

15.4. Rabin's cryptosystem.. This is based on the hardness of computing square roots modulo $N = pq$, where p, q are two "large" primes as in the RSA scheme. The factorization of N can be reduced to the computation of square roots, as follows. Choose some $x \in \mathbb{Z}_N$ and compute $y = \sqrt{x^2}$. Then $x^2 \equiv y^2 \bmod N$, or equivalently, $pq = N \mid (x + y)(x - y)$. If $x \not\equiv \pm y \bmod N$, then this gives us the factorization of N.

16. Factoring polynomials

The *Fundamental Theorem of Number Theory* states that every integer can be (essentially uniquely) factored as a product of primes. Similarly, for any field F the polynomials in $F[x_1, \cdots, x_n]$ can be (essentially uniquely) factored into a product of irreducible polynomials. In other words, $\mathbb{Z}$ and $F[x_1, \cdots, x_n]$ are *Unique Factorization Domains* (see section 9).

The "essentially uniquely" says that such a factorization is unique up to the order of the factors and multiplication by units, i.e., by ± 1 in $\mathbb{Z}$ and by nonzero constants from F in $F[x_1, \cdots, x_n]$. As an example, $(x^2-1) = (x-1)(x+1) = (-x+1)(-x-1)$. Recall that a polynomial $f \in F[x_1, \cdots, x_n]$ is *irreducible* if and only if $f \notin F$ and for any $g, h \in F[x_1, \cdots, x_n]$ with $f = gh$ we have $g \in F$ or $h \in F$.

PROBLEM 16.1. *(Univariate polynomial factorization) For a monic polynomial $f \in F[x]$, where F is a field, determine pairwise distinct monic irreducible polynomials $f_1, \ldots, f_r \in F[x]$ and positive integers $e_1, \ldots, e_r \in \mathbb{N}$ such that $f = f_1^{e_1} \cdots f_r^{e_r}$.*

It seems computationally difficult to factor large integers. However, a computer algebra system like Maple can routinely factor reasonably large polynomials; this is a task where the usual computational analogies between integers and polynomials do not work.

We will describe in detail (probabilistic) algorithms that factor a univariate polynomial of degree n over a finite field in polynomial time. More general questions include factoring polynomials in

- $\mathbb{Z}[x]$ and $\mathbb{Q}[x]$,
- $F[x]$, where F is an algebraic number field (i.e. a finite algebraic extension of $\mathbb{Q}$),
- $\mathbb{R}[x]$ and $\mathbb{C}[x]$,
- $F[x]$, where F is an arbitrary field,
- multivariate polynomials.

The dependencies between them are shown in Figure 16.1. It turns out that factoring univariate polynomials over finite fields is a basic task used in many other factoring algorithms.

The algorithm for finite fields proceeds in three stages:

1. squarefree factorization,
2. distinct-degree factorization,
3. equal-degree factorization.

In Figure 16.2, we see how the three stages work. The width of a box represents the degree of the corresponding polynomial. In the example, the original polynomial consists of four factors of degree 2 (two of them equal), one factor of degree 4, and one of degree 6.

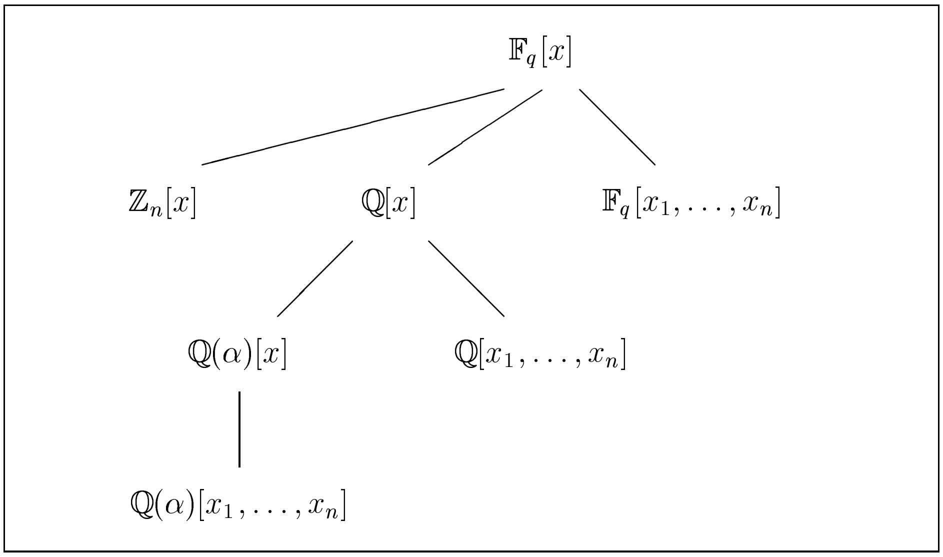


Figure 16.1: Polynomial factorization in various domains

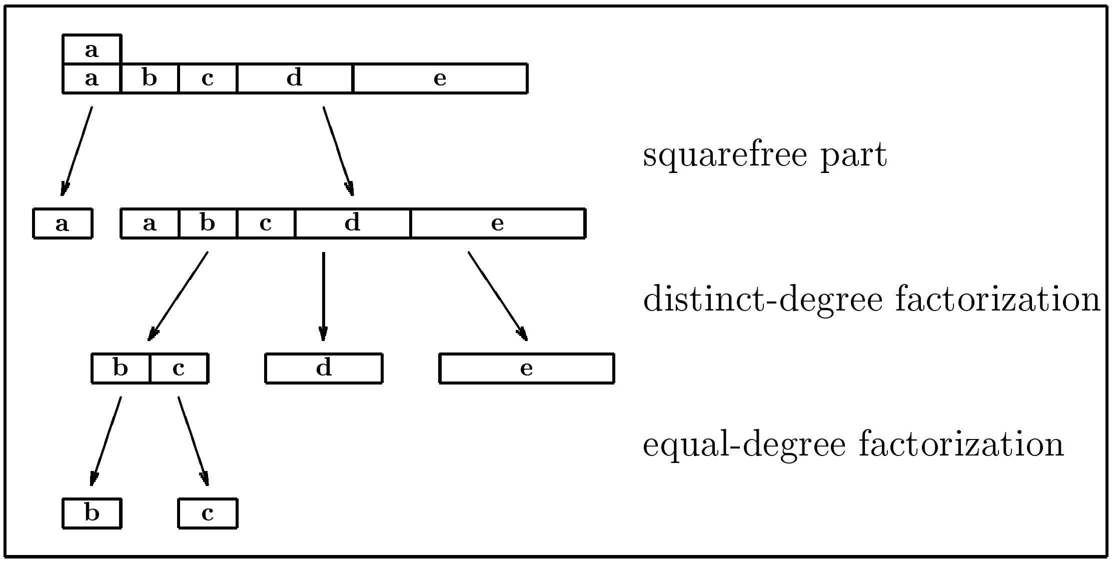


Figure 16.2: The stages of univariate polynomial factorization over finite fields

16.1. Facts about finite fields. Before we proceed, we collect some facts and techniques for finite fields. You find these, and all you ever want to know about finite fields, in the "bible of finite fields" by Lidl and Niederreiter (1983).

We have already dealt with the finite fields $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, where p is a prime. These, together with $\mathbb{Q}$, are the *prime fields*; every field contains exactly one of these. What other finite fields are there? If $f \in \mathbb{F}_p[x]$ is an irreducible polynomial of degree d, then $\mathbb{F}_{p^d} = \mathbb{Z}_p[x]/(f)$ is an algebraic field extension of $\mathbb{Z}_p$ of degree n. It is a field, and a vector space of dimension d over $\mathbb{F}_p$; thus it has p^d elements. It turns out that there are no other finite fields.

FACT 16.2. *For every prime p and $d \geq 1$, there exists a field with p^d elements. All such fields are isomorphic to each other. Every finite field has p^d elements, for some prime p and $d \geq 1$.*

We use $\mathbb{F}_{p^d}$ to denote any field with p^d elements. In particular, we may write $\mathbb{F}_p$ for $\mathbb{Z}_p$.

A ring R containing $\mathbb{F}_p$ is called an $\mathbb{F}_p$-algebra. A fundamental property is that for any commutative $\mathbb{F}_p$-algebra R, elements $a, b \in R$, and $i \in \mathbb{N}$, we have

$$(a+b)^{p^i} = a^{p^i} + b^{p^i}.$$

This is proved by induction on i; for $i = 1$, all binomial coefficients in the expansion of the left-hand power are divisible by p, and hence 0 in R.

Fermat's Little Theorem says that $a^{p-1} = 1$ for all $a \in \mathbb{F}_p^\times$, hence $a^p = a$ for all $a \in \mathbb{F}_p$. The following fact is a generalization to arbitrary finite fields.

THEOREM 16.3. *Let q be a prime power, and $d \geq 1$.*

(i) For all $a \in \mathbb{F}_q$, we have $a^q = a$, and

$$x^q - x = \prod_{a \in \mathbb{F}_q} (x - a).$$

(ii) We can consider $\mathbb{F}_q$ as a subfield of $\mathbb{F}_{q^d}$, and then

$$\mathbb{F}_q = \{a \in \mathbb{F}_{q^d} : a^q = a\}.$$

PROOF. (i) Lagrange's Theorem says that each element g of a group with m elements satisfies $g^m = 1$. The unit group $\mathbb{F}_q^\times = \mathbb{F}_q \setminus \{0\}$ has $q - 1$ elements, so that $a^{q-1} = 1$ for all nonzero $a \in \mathbb{F}_q$, and $a^q = a$ for all $a \in \mathbb{F}_q$. Thus $x - a$ divides $x^q - x$ for all $a \in \mathbb{F}_q$, and since $\gcd(x - a, x - b) = 1$ for $a \neq b$, we have that $\prod_{a \in \mathbb{F}_q} (x - a)$ divides $x^q - x$. Both polynomials are monic and have degree q, and hence they are equal.

(ii) Let $A = \{a \in \mathbb{F}_{q^d} : a^q = a\}$. Then $\mathbb{F}_q \subseteq A$, by (i). Since $\mathbb{F}_{q^d}$ is a field, A has at most q elements by Fact 14.5 (iii). It follows that $\mathbb{F}_q = A$. □

In the following, q always denotes a prime power. The most interesting case is when q is a prime number; you should always think of this case. However, most statements or proofs do not become simpler for this special case, so that we might as well work in full generality.

16.2. Squarefree factorization. For the moment, let F be an arbitrary field. A polynomial $f \in F[x]$ is *squarefree* if and only if f has no proper quadratic divisor, i.e., for any $g \in F[x]$ with g^2 dividing f we have $g \in F$. We will show how to reduce the problem of factoring arbitrary polynomials to that of factoring squarefree polynomials. In section 8, we defined the derivative f' of a polynomial $f = \sum_{0 \le i \le n} f_i x^i \in F[x]$ as $f' = \partial f / \partial x = \sum_{0 \le i \le n} i f_i x^{i-1}$. Note that i plays two different roles here: as a summation index, where it is really just a convenient notation for the vector $(f_0, \cdots, f_n) \in F^{n+1}$ of coefficients, and the field element $i = 1 + \cdots + 1 \in F$. This is a purely formal definition, without any limit processes, and satisfies the usual rules, for all $f, g \in F[x]$ and $a, b \in F$:

Triviality on constants: $a' = 0$,

F-linearity: $(af + bg)' = af' + bg'$,

Leibniz rule: $(fg)' = fg' + f'g$.

Suppose that f is not squarefree, so that $f = g^2 h$ for some $g, h \in F[x]$ and $g \notin F$. Then $f' = g \cdot (2h'h + gh')$, so that g divides $u = \gcd(f, f')$. If $\mathrm{char} F = 0$, then $f' \neq 0$ and $\deg f' < n$, so that u is a proper divisor of f, and it now suffices to factor u and f/u, two polynomials of degree smaller than n.

An interesting possibility may occur in $\mathbb{F}_p[x]$ for a prime p, which does not happen if $\mathrm{char} F = 0$: $f \notin F$ and $f' = 0$. This happens if and only if each i with $f_i \neq 0$ is divisible by p; then the summand $i f_i x^{i-1}$ is zero in $\mathbb{F}_p[x]$. Then we can write

$$f = \sum_{0 \le i \le n/p} f_{ip} x^{ip} = \Big(\sum_{0 \le i \le n/p} f_{ip} x^i \Big)^p, \tag{16.1}$$

since $(g+h)^p = g^p + h^p$ for all $g, h \in \mathbb{F}_p[x]$ and $f_{ip}^p = f_{ip}$ for all $f_{ip} \in \mathbb{F}_p$.

Similarly, if $F = \mathbb{F}_q$ for a prime power $q = p^s$ and $s \ge 1$, Theorem 16.3 says that $a^q = a$ for all $a \in \mathbb{F}_q$, and hence $a^{p^{s-1}} = a^{q/p}$ is a pth root of a. Then for $g = \sum_{0 \le i \le n/p} f_{ip}^{q/p} x^i$, we have $f = g^p$ in analogy to (16.1). Thus in any case we have

$$\forall f \in \mathbb{F}_q[x] \quad f' = 0 \Rightarrow f \text{ is a } p\text{th power.}$$

Now let $f = \prod_{1 \le i \le r} f_i^{e_i}$ be the prime factorization of $f \in \mathbb{F}_q[x]$ as in Problem 16.1, and $n = \deg f$. We define the *squarefree part of* f to be $\prod_{1 \le i \le r} f_i$. Then $f' = \sum_{1 \le i \le r} e_i \frac{f}{f_i} f_i'$. For $1 \le i \le r$, we have that $\gcd(f_i, f_i') = 1$, since $f_i' = 0$ would imply that f_i is a pth power in contradiction to the irreducibility of f_i, and $f_i' \neq 0$ and $\deg f_i' < \deg f_i$ imply that the gcd is not f_i and hence 1 by the irreducibility of f_i. (In the language of field theory, any irreducible polynomial has a nonzero derivative, and thus $\mathbb{F}_q$ is *perfect.* The field $F = \mathbb{F}_q(y)$ of rational functions in y is not perfect, since e.g. $f = x^q - y \in F[x]$ is irreducible with $f' = 0$. The reason for this is that the "constant" $y \in F$ has no qth root.) So

$$u = \gcd(f, f') = \prod_{\substack{1 \le i \le r \\ p \nmid e_i}} f_i^{e_i - 1} \cdot \prod_{\substack{1 \le i \le r \\ p \mid e_i}} f_i^{e_i}.$$

Furthermore,

$$v = \frac{f}{u} = \prod_{\substack{1 \le i \le r \\ p \nmid e_i}} f_i$$

is squarefree,

$$\gcd(u, v^n) = \prod_{\substack{1 \le i \le r \\ p \nmid e_i}} f_i^{e_i - 1},$$

since $e_i \le n$ for $1 \le i \le r$, and finally

$$\frac{u}{\gcd(u, v^n)} = \prod_{\substack{1 \le i \le r \\ p \mid e_i}} f_i^{e_i}.$$

Example 16.4. *Let $\mathbb{F}_4 = \mathbb{F}_2(\alpha)$, where $\alpha^2 + \alpha + 1 = 0$, be the finite field with four elements, and*

$$f = x^9 + x^8 + \alpha\, x^7 + \alpha\, x^6 + (\alpha+1)\, x^5 + (\alpha+1)\, x^4 = x^4(x+1)^3(x+\alpha)^2 \in \mathbb{F}_4[x].$$

Then $n = 9$,

$$\begin{aligned}
f' &= x^8 + \alpha x^6 + (\alpha+1)x^4, \\
u &= \gcd(f, f') = f' = x^4(x+1)^2(x+\alpha)^2, \\
v &= \frac{f}{u} = x + 1, \\
\gcd(u, v^n) &= x^2 + 1 = (x+1)^2, \\
w &= \frac{u}{\gcd(u, v^n)} = x^6 + (\alpha+1)x^4.
\end{aligned}$$

Now $w = (x^3 + \alpha x^2)^2$ is a square, and we recursively compute the squarefree part $x^2 + \alpha x$ of $g = x^3 + \alpha x^2$. The squarefree part of f is then the product of v and the squarefree part of g, i.e.,

$$(x+1)(x^2 + \alpha x) = x^3 + (\alpha+1)x^2 + x = x(x+1)(x+\alpha).$$

We obtain the following algorithm for computing the squarefree part in $\mathbb{F}_q[x]$.

Algorithm 16.5. *Squarefree part.*
Input: *$f \in \mathbb{F}_q[x]$ monic of degree n, and $p = \operatorname{char} \mathbb{F}_q$.*
Output: *The squarefree part of f, i.e., the product of all distinct monic irreducible factors of f.*

(i) $u = \gcd(f, f')$. *If $u = 1$ then return f.*

(ii) $v = \dfrac{f}{u}$.

(iii) $w = \dfrac{u}{\gcd(u, v^n)}$.

(iv) $z = \sqrt[p]{w}$.

(v) *Return* $v \cdot \text{squarefree part}(z)$.

THEOREM 16.6. *The algorithm works correctly as specified. It can be performed with $O(\mathsf{M}(n)\log n + n\log q)$ operations in $\mathbb{F}_q$.*

PROOF. The correctness is clear from the above discussion. Let $S(n)$ denote the cost of the algorithm. Steps 1 and 2 cost $O(\mathsf{M}(n)\log n)$ operations, and the same is true for step 3 if we first compute $\tilde{v} \in \mathbb{F}_q[x]$ with $\deg \tilde{v} < \deg u$ and $\tilde{v} \equiv v^n \bmod u$ using repeated squaring with reduction modulo u after each multiplication, and then compute $\gcd(u, \tilde{v})$. In step 4 we have to compute n/p many (q/p)th powers in $\mathbb{F}_q$ taking $O(\frac{n}{p}\log\frac{q}{p})$ operations. The degree of z in step 4 is at most $\frac{n}{p}$, so that we have the following recursive equation:

$$S(n) = S(\frac{n}{p}) + O(\mathsf{M}(n)\log n + \frac{n}{p}\log\frac{q}{p}). \tag{16.2}$$

If c is the implied constant in (16.2), $d = \frac{pc}{p-1}$, and $T(n) = \mathsf{M}(n)\log n + n\log q$, then we find inductively

$$S(n) \le S(\frac{n}{p}) + cT(n) \le dT(\frac{n}{p}) + cT(n) \le (\frac{d}{p} + c)T(n) = dT(n),$$

where we have used that $p\mathsf{M}(\frac{n}{p}) \le \mathsf{M}(n)$. Thus $S(n) \in O(\mathsf{M}(n)\log n + n\log q)$. □

A more complete factorization of f is given by the *squarefree decomposition* $(g_1, \ldots, g_n)$ of f, where $g_1, \ldots, g_n \in \mathbb{F}_q[x]$ are monic and squarefree, $f = \prod_{1\le i\le n} g_i^i$, and $\gcd(g_i, g_j) = 1$ for $i \ne j$. One can actually compute this at the same cost as in Theorem 16.6 (Yun 1976).

If we want to factor an arbitrary polynomial $f \in \mathbb{F}_q[x]$, we factor its squarefree part and then obtain the factorization of f by checking to what power each irreducible factor occurs in f.

16.3. Distinct-degree factorization. From now on we want to factor a squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree n. We first need the following result.

THEOREM 16.7. *For any $d \ge 1$, $x^{q^d} - x \in \mathbb{F}_q[x]$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[x]$ whose degree divides d.*

PROOF. Since the derivative $(x^{q^d} - x)'$ is -1, the polynomial is squarefree. It is sufficient to show for any monic irreducible polynomial $f \in \mathbb{F}_q[x]$ of degree n that

$$f \text{ divides } x^{q^d} - x \iff n \text{ divides } d.$$

Consider the field extension $\mathbb{F}_q \subseteq \mathbb{F}_{q^d}$. If f divides $x^{q^d} - x$, then from Theorem 16.3 (i), applied to $\mathbb{F}_{q^d}$, we get a set $A \subseteq \mathbb{F}_{q^d}$ with $f = \prod_{a \in A}(x-a)$. Choose $a \in A$, and let $F = \mathbb{F}_q[x]/(f) = \mathbb{F}_q(a) \subseteq \mathbb{F}_{q^d}$. This is a field with q^n elements, and $\mathbb{F}_{q^d}$ is an extension of F, so that $q^d = (q^n)^e$ for some integer $e \geq 1$. Hence n divides d.

Now suppose that n divides d, let $\mathbb{F}_{q^n} = \mathbb{F}_q[x]/(f)$, and $a = (x \bmod f) \in \mathbb{F}_{q^n}$ be a root of f. By Theorem 16.3, we have that $a^{q^n} = a$, and hence

$$(x-a) \mid (x^{q^n} - x) \mid (x^{q^d} - x),$$

so that $(x-a)$ divides $\gcd(f, x^{q^d} - x) \in \mathbb{F}_{q^n}[x]$. But the gcd of two polynomials with coefficients in $\mathbb{F}_q$ also has coefficients in $\mathbb{F}_q$, and since it is nonconstant and f is irreducible, $\gcd(f, x^{q^d} - x) = f$, or, equivalently, f divides $x^{q^d} - x$. $\square$

Remark. Recall from Example 11.10 that the gcd of two polynomials $f, g \in F[x]$ over an extension field $E \supset F$ is the same as the gcd over F.

The *distinct-degree decomposition* of f is the sequence $(g_1, \cdots, g_n)$ of polynomials, where g_i is the product of all monic irreducible polynomials in $\mathbb{F}_q[x]$ of degree i that divide f. As an example, $(x^2 + x, x^4 + x^3 + x + 2, 1, 1, 1, 1)$ is the distinct-degree decomposition of $f = x(x+1)(x^2+1)(x^2+x+2) \in \mathbb{F}_3[x]$; the two quadratic factors are irreducible.

Algorithm 16.8. *District-degree factorization.*
Input: *A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree n.*
Output: *The distinct-degree decomposition $(g_1, \cdots, g_n)$ of f.*

(i) *Set $h_0 = x$ and $f_0 = f$.*

(ii) *For $i = 1, \ldots, n$ do steps 3 and 4.*

(iii) *Compute $h_i \in \mathbb{F}_q[x]$ with $h_i \equiv h_{i-1}^q \bmod f$ and $\deg h_i < n$.*

(iv) *Compute $g_i = \gcd(h_i - x, f_{i-1}) \in \mathbb{F}_q[x]$, and $f_i = f_{i-1}/g_i$.*

(v) *Return $(g_1, \cdots, g_n)$.*

Theorem 16.9. *The algorithm works correctly as specified. It can be performed with $O(n\mathsf{M}(n)\log(nq))$ operations in $\mathbb{F}_q$.*

Proof. Let $(G_1, \cdots, G_n)$ be the distinct-degree decomposition of f. For the correctness, it is sufficient to show by induction on $i \geq 0$ that

$$\begin{aligned} h_i &\equiv x^{q^i} \bmod f \text{ and } f_i = G_{i+1} \cdots G_n, \\ g_i &= G_i \text{ for } i \geq 1. \end{aligned}$$

The first two claims are clear for $i = 0$. For $i \geq 1$, $h_i \equiv h_{i-1}^q \equiv (x^{q^{i-1}})^q = x^{q^i} \bmod f$,

$$\begin{aligned} h_i - x &\equiv x^{q^i} - x \bmod f, \\ g_i &= \gcd(h_i - x, f_{i-1}) = \gcd(x^{q^i} - x, f_{i-1}). \end{aligned}$$

By Theorem 16.7, g_i is the product of all monic irreducible polynomials in $\mathbb{F}_q[x]$ of degree dividing i that divide $f_{i-1} = G_i \cdots G_n$, hence $g_i = G_i$. Furthermore, $f_i = G_i \cdots G_n / g_i = G_{i+1} \cdots G_n$. This finishes the inductive step.

The cost of the ith iteration is $O(\log q)$ multiplications modulo f in step 3, or $O(\mathsf{M}(n) \log q)$ operations in $\mathbb{F}_q$. Each execution of step 4 can be done with $O(\mathsf{M}(n) \log n)$ operations in F. $\square$

16.4. Equal-degree factorization. Our task now is to factor one of the polynomials that are produced by the previous distinct-degree factorization. So we have a prime power q, a monic polynomial $f \in \mathbb{F}_q[x]$ with $\deg f = n$, and also a divisor $d \in \mathbb{N}$ so that each irreducible factor of f has degree d. There are $r = n/d$ such factors, and we can write

$$f = f_1 \cdots f_r$$

with $f_1, \ldots, f_r \in \mathbb{F}_q[x]$ irreducible monic and pairwise different. We may assume that $r \geq 2$; otherwise, we know that f is irreducible. Since $\gcd(f_i, f_j) = 1$ for $i \neq j$, we have the isomorphism of the Chinese Remainder Theorem:

$$\chi : R = \mathbb{F}_q[x]/(f) \longrightarrow \mathbb{F}_q[x]/(f_1) \times \cdots \times \mathbb{F}_q[x]/(f_r) = R_1 \times \cdots \times R_r.$$

Each R_i is a field with q^d elements, and an algebraic extension of $\mathbb{F}_q$ of degree d:

$$R_i = \mathbb{F}_{q^d} = \mathbb{F}_q[x]/(f_i) \supseteq \mathbb{F}_q.$$

We use the notation that for $a \in \mathbb{F}_q[x]$, we have $a \bmod f \in R$ and $\chi(a \bmod f) = (a \bmod f_1, \cdots, a \bmod f_r) = (\chi_1(a), \cdots, \chi_r(a))$, where $\chi_i(a) = a \bmod f_i \in R_i$. For $a \in \mathbb{F}_q[x]$ and $i \leq r$, we have that f_i divides a if and only if $\chi_i(a) = 0$. If we have an $a \in \mathbb{F}_q[x]$ with some $\chi_i(a)$ equal to zero and others nonzero, then $\gcd(a, f)$ is a nontrivial divisor of f. We now describe a probabilistic procedure to find such a *splitting polynomial* a.

We assume now that q is odd, and write $e = (q^d - 1)/2$. For any $\beta \in R_i^\times = \mathbb{F}_{q^d}^\times$, we have $\beta^e \in \{1, -1\}$, and both possibilities occur equally often (see Lemma 14.6 for the case $q = p$ prime and $d = 1$). If we choose $a \in \mathbb{F}_q[x]$ with $\deg a < n$ and $\gcd(a, f) = 1$ at random, then $\chi_1(a), \cdots, \chi_r(a)$ are independently and uniformly distributed elements of $\mathbb{F}_{q^d}^\times$, and $\epsilon_i = \chi_i(a^e) \in R_i$ is 1 or -1, each with probability 1/2. Therefore

$$\chi(a^e - 1) = (\epsilon_1 - 1, \cdots, \epsilon_r - 1),$$

and $a^e - 1$ is a splitting polynomial unless $\epsilon_1 = \cdots = \epsilon_r$. This occurs with probability $2 \cdot (\frac{1}{2})^r = 2^{-r+1} \leq 1/2$.

ALGORITHM 16.10. *Equal-degree factorization.*
Input: *A squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree n, where q is an odd prime power, and a divisor d of n, so that all irreducible factors of f have degree d.*
Output: *A proper factor $g \in \mathbb{F}_q[x]$ of f, or "failure".*

(i) *Choose* $a \in \mathbb{F}_q[x]$ *with* $\deg a < n$ *at random.*

(ii) *If* $g_1 = \gcd(a, f) \neq 1$, *then return* $g = g_1$.

(iii) *Compute* $b \in \mathbb{F}_q[x]$ *with* $\deg b < n$ *and*

$$b \equiv a^{(q^d-1)/2} - 1 \bmod f,$$

by repeated squaring, reducing modulo f *after each multiplication.*

(iv) *If* $g_2 = \gcd(b, f) \neq 1$, *then return* $g = g_2$, *else return "failure".*

THEOREM 16.11. *Algorithm equal-degree factorization works correctly as specified. It returns "failure" with probability less than* $2^{1-r} \leq 1/2$, *where* $r = n/d$. *It can be implemented with* $O((d \log q + \log n)\mathsf{M}(n))$, *or* $O((d \log q + \log n)n \log n \log \log n)$ *operations in* $\mathbb{F}_q$.

PROOF. The failure probability has been given above as 2^{1-r} if $\gcd(a, f) = 1$. For general a, where step 2 might find a factor, the failure probability is less than 2^{1-r}. The cost of steps 2 and 4 is $O(\mathsf{M}(n) \log n)$. Step 3 uses at most $2 \log_2(q^d) = O(d \log q)$ multiplications modulo f. One such multiplication consists of calculating the product of two polynomials in $\mathbb{F}_q[x]$ with degree less than n, and then reducing that product modulo f. Both steps can be done with $O(\mathsf{M}(n))$ operations in $\mathbb{F}_q$. □

The usual trick of running the algorithm k times makes the failure probability less than $2^{(1-r)k} \leq 2^{-k}$.

This algorithm gives a factorization into two factors; however, we will usually want all r factors. The first possibility is to run the algorithm recursively on each factor; this will factor f completely with $O(n\,\mathsf{M}(n) \log n \log q)$ operations in $\mathbb{F}_q$.

A second possibility is the following. We choose only $k = 2\lceil \log_2 n \rceil$ different values of a, and compute the k factorizations into two factors. With probability at least $1/2$, every pair f_i and f_j of factors, with $1 \leq i < j \leq r$, is "separated" by one of these factorizations: f_i divides one factor, and f_j the other. Then there is an easy way of "refining" all these factorizations into one; this will (probably) be the complete factorization into irreducible factors. The following example should make clear how this refinement works.

EXAMPLE 16.12. *Suppose that we want to find all the irreducible factors* f_i *of* $f = f_0 \cdots f_9 \in \mathbb{F}_q[x]$, *where the* f_i *are monic, irreducible, pairwise distinct, and have the same degree* d. *For each iteration of Algorithm 16.10, we get a—hopefully nontrivial—factor* g *of* f *which corresponds to a uniformly random chosen subset* $I \subseteq \{0, \ldots, 9\}$ *in such a way that* $g = \prod_{i \in I} f_i$. *If we now compute* gcd*'s with the partial factorization of* f *that we already have, irreducible factors of* f *that divide* g, *i.e., whose indices lie in* I, *will be separated from those that do not divide* g, *and we get a finer partial factorization of* f *in the sense that some of the composite factors are split and some remain unchanged.*

The following table shows some sets I that were randomly chosen, and the partial factorizations of f in each iteration. For example, the partial factorization denoted by (4)(037)(589)(126) consists of the 4 pairwise relatively prime factors f_4, $f_0f_3f_7$, $f_5f_8f_9$, and $f_1f_2f_6$ whose product is f.

iteration	*I*	*partial factorization*
0		(0123456789)
1	0347	(0347)(125689)
2	4589	(4)(037)(589)(126)
3	1347	(4)(37)(0)(589)(1)(26)
4	02579	(4)(7)(3)(0)(59)(8)(1)(2)(6)
5	234579	(4)(7)(3)(0)(59)(8)(1)(2)(6)
6	02679	(4)(7)(3)(0)(9)(5)(8)(1)(2)(6)

Figure 16.3 shows the refinement process in form of a tree. The leaves correspond to the isolated irreducible factors.

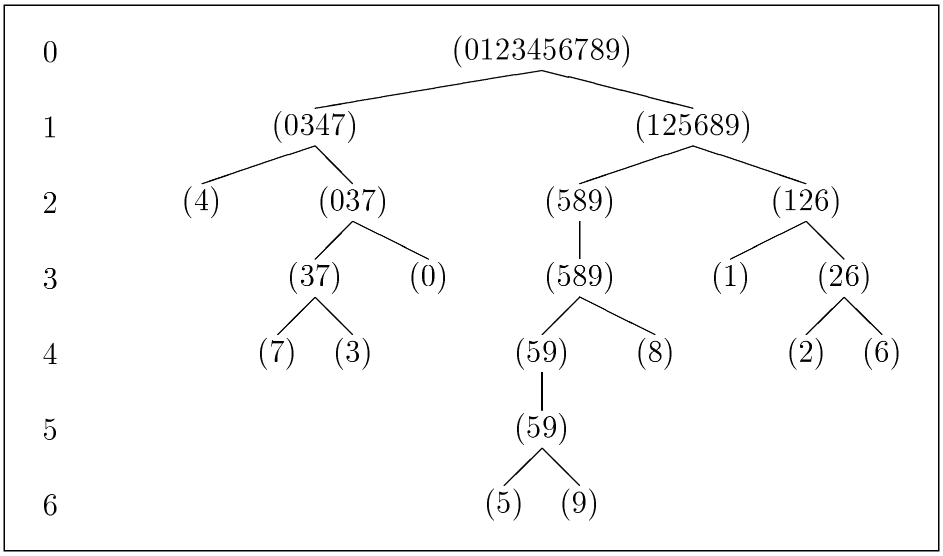


Figure 16.3: Refinement of partial factorizations in Example 16.12

Remark 16.13. *If we are looking for all zeroes of a given polynomial $f \in \mathbb{F}_q[x]$, i.e., we want to determine all the linear factors of f, it is clearly sufficient to first compute $g = \gcd(x^q - x, f)$ and then apply the equal-degree factorization algorithm to g, i.e., we need not compute the* whole *distinct-degree decomposition of f.*

16.5. The iterated Frobenius algorithm. We now sketch the fastest known algorithm for our problem of factoring polynomials. A central role is played by the *Frobenius automorphism*

$$\sigma : \begin{cases} \mathbb{F}_{q^n} & \longrightarrow & \mathbb{F}_{q^n}, \\ a & \longmapsto & a^q, \end{cases}$$

The field extension $\mathbb{F}_{q^n}/\mathbb{F}_q$ is normal, σ generates the Galois group $\mathrm{Gal}(\mathbb{F}_{q^n}/\mathbb{F}_q) = \{\mathrm{id}, \sigma, \ldots, \sigma^{n-1}\}$, and $\sigma^n = \mathrm{id}$, by Fermat's Little Theorem.

For a squarefree monic polynomial $f \in \mathbb{F}_q[x]$ of degree n, we now consider the quotient ring $R = \mathbb{F}_q[x]/(f)$ and the map

$$\sigma : \left\{ \begin{array}{ccc} R & \longrightarrow & R, \\ a & \longmapsto & a^q, \end{array} \right.$$

which is also called the *Frobenius endomorphism* of R (it is in fact an automorphism since f is squarefree). It satisfies the following rules.

- $\forall a, b \in R \quad \sigma(a+b) = \sigma(a) + \sigma(b)$,
- $\forall a, b \in R \quad \sigma(ab) = \sigma(a)\sigma(b)$,
- $\forall c \in \mathbb{F}_q \quad \sigma(c) = c$.

The last one is exactly Fermat's Little Theorem, where we identify $\mathbb{F}_q$ with the subfield of R consisting of the residue classes of the constant polynomials modulo f. These rules imply in particular that $g(a^q) = g(a)^q$ for any $a \in R$ and any polynomial $g \in \mathbb{F}_q[x]$. (Exercise: Prove this!)

If f is irreducible, then $R = \mathbb{F}_{q^n}$, and the two notions of the Frobenius map coincide. But in our situation, f will be a composite polynomial that we want to factor.

Let $\xi = x \bmod f \in R$. Then the powers $\{1, \xi, \cdots, \xi^{n-1}\}$ of ξ form an $\mathbb{F}_q$-basis of R, and any element $\alpha \in R$ can be written as

$$\alpha = a_0 + a_1\xi + \cdots + a_{n-1}\xi^{n-1} = a(\xi) = (a \bmod f),$$

with unique $a_0, \ldots, a_{n-1} \in \mathbb{F}_q$, where $a = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} \in \mathbb{F}_q[x]$ is the canonical representative in $\mathbb{F}_q[x]$ of degree less than n for the residue class α. We will write $\breve{\alpha}$ for a in the sequel, so that

$$\breve{\alpha}(\xi) = \breve{\alpha} \bmod f = \alpha. \tag{16.3}$$

In software, there is no need to distinguish between α and $\breve{\alpha}$, or, equivalently, between $(a \bmod f)$ and a. Both will be represented by an array $[a_0, \ldots, a_{n-1}]$ of coefficients. But conceptually, one has to make the distinction; e.g., it does not make sense to "evaluate α at ξ^q", but we can (and will) evaluate a at ξ^q.

If $\alpha \in R$ is arbitrary and $a = \breve{\alpha} \in \mathbb{F}_q[x]$ as above, we can compute the image of α under the Frobenius map as

$$\alpha^q = a(\xi)^q = a(\xi^q) = \breve{\alpha}(\xi^q). \tag{16.4}$$

This is called the *polynomial representation* of the Frobenius map, and it will be used in the following algorithm.

Algorithm 16.14. *Iterated Frobenius.*
Input: $f \in \mathbb{F}_q[x]$ *squarefree of degree* n, $\xi^q \in R = \mathbb{F}_q[x]/(f)$, *where* $\xi = x \bmod f \in R$, $d \in \mathbb{N}$ *with* $d \le n$, *and* $\alpha \in R$.
Output: $\alpha, \alpha^q, \ldots, \alpha^{q^d} \in R$.

(i) Set $\gamma_0 = \xi$, $\gamma_1 = \xi^q$, *and* $l = \lceil \log_2 d \rceil$.

(ii) For $1 \le i \le l$ *compute*

$$\gamma_{2^{i-1}+j} = \check{\gamma}_{2^{i-1}}(\gamma_j) \text{ for } 1 \le j \le 2^{i-1},$$

using fast multipoint evaluation.

(iii) Compute

$$\delta_k = \check{\alpha}(\gamma_k) \text{ for } 0 \le k \le d,$$

using fast multipoint evaluation.

(iv) Return $\delta_0, \ldots, \delta_d$.

Theorem 16.15. *Algorithm 16.14 works correctly as specified and uses* $O(\mathsf{M}(n)^2 \log n \log d)$ *operations in* $\mathbb{F}_q$.

Proof. For the correctness, we prove the invariant

$$\gamma_k = \xi^{q^k} \text{ for } 0 \le k \le 2^i$$

by induction on i. The case $i = 0$ is clear from step 1. For the inductive step, it is sufficient to prove the claim for $k > 2^{i-1}$. For $1 \le j \le 2^{i-1}$, we have that

$$\gamma_{2^{i-1}+j} = \check{\gamma}_{2^{i-1}}(\gamma_j) = \check{\gamma}_{2^{i-1}}(\xi^{q^j}) = (\check{\gamma}_{2^{i-1}}(\xi))^{q^j} = \gamma_{2^{i-1}}^{q^j} = (\xi^{q^{2^{i-1}}})^{q^j} = \xi^{q^{2^{i-1}+j}},$$

by step 2, (16.3), (16.4), and the induction hypothesis.

Finally, in step 3 we correctly compute

$$\delta_k = \check{\alpha}(\gamma_k) = \check{\alpha}(\xi^{q^k}) = \check{\alpha}(\xi)^{q^k} = \alpha^{q^k}$$

for $0 \le k \le d$.

In section 6, we showed that the problem of evaluating a polynomial over a field of degree at most n at no more than n field elements can be solved using $O(\mathsf{M}(n)\log n)$ field operations. It is not hard to see that the only divisions occurring in that algorithm are by 1, and so the statement is still valid over arbitrary rings. In steps 2 and 3 of Algorithm 16.14, we have to solve $l+1 = O(\log d)$ such multipoint evaluation problems over the ring R, and hence get a total cost of $O(\mathsf{M}(n)\log n \log d)$ operations, i.e., multiplications and additions, in R, or $O(\mathsf{M}(n)^2 \log n \log d)$ operations in $\mathbb{F}_q$. □

In fact, it can be shown that $O((n/d)\mathsf{M}(nd)\log d)$ operations in $\mathbb{F}_q$ are sufficient, but since this saves only factors $\log n$, we omit the proof.

The workings of the iterated Frobenius algorithm can be illustrated as follows:

$$\xi^{q^1} \quad \underbrace{\xi^{q^2}}_{i=1} \quad \underbrace{\xi^{q^3} \quad \xi^{q^4}}_{i=2} \quad \underbrace{\xi^{q^5} \quad \xi^{q^6} \quad \xi^{q^7} \quad \xi^{q^8}}_{i=3} \quad \underbrace{\xi^{q^9} \quad \xi^{q^{10}} \quad \xi^{q^{11}} \quad \xi^{q^{12}} \quad \xi^{q^{13}} \quad \xi^{q^{14}} \quad \xi^{q^{15}} \quad \xi^{q^{16}}}_{i=4} \quad \cdots$$

The ith brace encloses exactly those powers of ξ that are newly computed in the ith iteration of step 2. The progress of the iterated Frobenius algorithm over the naïve successive computation of the ξ^{q^k} is similar to the progress of repeated squaring for the computation of one power a^n over repeated multiplication.

Algorithm 16.14 can be used for distinct-degree factorization as well as for equal-degree factorization. Remember that in Algorithm 16.8, we had to compute

$$x^{q^i} - x \bmod f = \xi^{q^i} - \xi = \gamma_i - \xi$$

for $1 \le i \le n$. This can be done by first computing ξ^q using repeated squaring and then applying steps 1 and 2 of the iterated Frobenius algorithm with $d = n$. The cost for the other steps in the distinct-degree factorization algorithm is dominated by the cost for the iterated Frobenius, and so we have the following corollary.

Corollary 16.16. *The distinct-degree decomposition of a squarefree polynomial $f \in \mathbb{F}_q[x]$ of degree n can be computed using $O(\mathsf{M}(n^2)\log n + \mathsf{M}(n)\log q)$ operations in $\mathbb{F}_q$.*

For the equal-degree factorization, we compute $\alpha^{(q^d-1)/2}$ for a uniformly randomly chosen $\alpha \in R = \mathbb{F}_q[x]/(f)$, where d is the degree of any of the irreducible factors of f. The exponent $\frac{q^d-1}{2}$ can be written as

$$\frac{q^d-1}{2} = (1 + q + \cdots + q^{d-1})\frac{q-1}{2},$$

and hence

$$\alpha^{(q^d-1)/2} = (\alpha\alpha^q \cdots \alpha^{q^{d-1}})^{(q-1)/2} = (\delta_0\delta_1 \cdots \delta_{d-1})^{(q-1)/2},$$

which can be computed using the iterated Frobenius algorithm, and repeated squaring for the computation of the initial power ξ^q and the final $\frac{q-1}{2}$th power.

Corollary 16.17. *The equal-degree factorization of a squarefree polynomial $f \in \mathbb{F}_q[x]$ of degree n with r irreducible factors of degree d can be computed using an expected number of $O(\mathsf{M}(nd)r\log d + \mathsf{M}(n)\log r \log q)$ operations in $\mathbb{F}_q$.*

Theorem 16.18. *A polynomial $f \in \mathbb{F}_q[x]$ of degree n can be completely factored with an expected number of $O((n^2 + n\log q)\log^2 n \operatorname{loglog} n)$ operations in $\mathbb{F}_q$.*

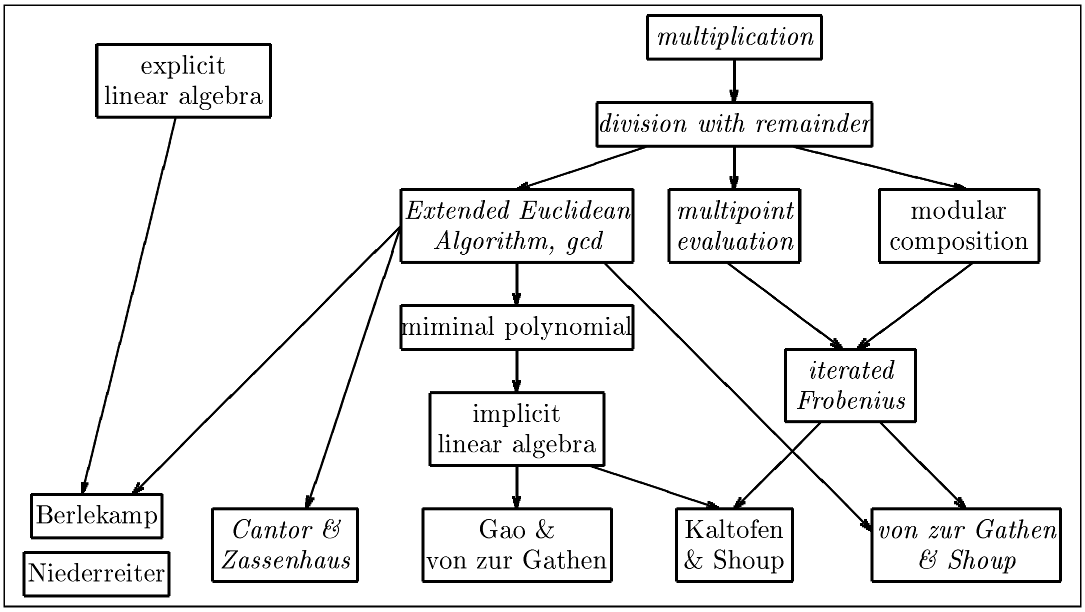


Figure 16.4: Algorithms in computer algebra used for polynomial factorization

16.6. Notes. The algorithms discussed in subsections 16.3 and 16.4 are based on the work by Cantor & Zassenhaus (1981), those from subsection 16.5 are from von zur Gathen & Shoup (1992). The first pioneering random polynomial-time algorithms, with a different approach based on linear algebra, are due to Berlekamp (1967, 1970).

Figure 16.4 gives an overview of algorithms for polynomial arithmetic in computer algebra, in particular those that are used for factoring univariate polynomials over finite fields. The arrows indicate dependencies. The names written in *italics* are algorithms that we have presented in this course.

References

L. M. Adleman, Algorithmic number theory – the complexity contribution. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, ed. S. Goldwasser, Santa Fe, New Mexico, 1994, IEEE Computer Society Press, 88–113.

L. Adleman and M. Huang, Recognizing primes in random polynomial time. In *Proc. 19th Ann. ACM Symp. Theory of Computing*, 1987, 462–469.

L. M. Adleman, C. Pomerance, and R. S. Rumely, On distinguishing prime numbers from composite numbers. *Ann. Math.* **117** (1983), 173–206.

N. C. Ankeny, The least quadratic non residue. *Ann. Math.* **55** (1952), 65–72.

A. O. L. Atkin and F. Morain, Elliptic curves and primality proving. *Math. of Comp.* **61**(203) (1993), 29–68.

E. Bach, *Analytic Methods in the Analysis and Design of Number Theoretic Algorithms.* MIT press, 1985.

E. Bach, Number-theoretic algorithms. *Ann. Rev. Comput. Sci.* **4** (1990), 119–172.

E. H. Bareiss, Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comp.* **22** (1968), 565–578.

P. Beckmann, *A History of Pi.* Golem Press, Boulder, CO, 4th edition, 1977.

E. R. Berlekamp, Factoring polynomials over finite fields. *Bell System Tech. J.* **46** (1967), 1853–1859.

E. R. Berlekamp, Factoring polynomials over large finite fields. *Math. Comp.* **24** (1970), 713–735.

A. Borodin and R. Moenck, Fast modular transforms. *J. Comput. System Sci.* **8** (1974), 366–86.

A. Borodin and I. Munro, *The Computational Complexity of algebraic and numeric Problems.* American Elsevier, New York NY, 1975.

J. M. Borwein and P. B. Borwein, Strange series and high precision fraud. *American Mathematical Monthly* **99**(7) (1992), 622–640.

J. M. Borwein, P. B. Borwein, and D. H. Bailey, Ramanujan, modulular equations, and approximations to pi or how to compute one billion digits of pi. *American Mathematical Monthly* **93**(3) (1989), 201–219.

G. Brassard and P. Bratley, *Algorithmics - Theory & Practice.* Prentice Hall, 1988.

J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, and S. S. Wagstaff, Jr., *Factorizations of* $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$, *up to high powers*, vol. 22 of *Contemporary Mathematics.* American Mathematical Society, Providence RI, 2nd edition, 1988.

W. S. Brown and J. F. Traub, On Euclid's algorithm and the theory of subresultants. *J. Assoc. Comput. Mach.* **18** (1971), 505–514.

D. G. Cantor, On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A* **50** (1989), 285–300.

D. G. Cantor and E. Kaltofen, On fast multiplication of polynomials over arbitrary algebras. *Acta. Inform.* **28** (1991), 693–701.

D. G. Cantor and H. Zassenhaus, A new algorithm for factoring polynomials over finite fields. *Math. Comp.* **36** (1981), 587–592.

H. Cohen and H. W. Lenstra, Jr., Primality testing and Jacobi sums. *Math. Comp.* **42** (1984), 297–330.

G. E. Collins, Subresultants and reduced polynomial remainder sequences. *J. Assoc. Comput. Mach.* **14** (1967), 128–142.

J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* **19** (1965), 297–301.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms.* MIT Press, 1989.

J. C. Cunningham and H. J. Woodall, *Factorization of* $y^n \pm 1$, $y = 2$, 3, 5, 6, 7, 10, 11, 12 *up to high powers* (n). Hodgson & Sons, London, 1925.

W. Diffie and M. Hellman, New directions in cryptography. *IEEE Trans. Inform. Theory* **22** (1976), 644–654.

J. von zur Gathen, Parallel algorithms for algebraic problems. *SIAM J. Comput.* **13** (1984), 802–824.

J. von zur Gathen, Functional decomposition of polynomials: the tame case. *J. Symb. Comp.* **9** (1990), 281–299.

J. von zur Gathen and J. Gerhard, Fast multipoint evaluation and interpolation at arbitrary linear subspaces of finite fields. Preprint, 1995.

J. von zur Gathen and V. Shoup, Computing Frobenius maps and factoring polynomials. *Comput complexity* **2** (1992), 187–224.

K. O. Geddes, S. R. Czapor, and G. Labahn, *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.

S. Goldwasser and J. Kilian, Almost all primes can be quickly certified. In *Proc. 18th Ann. ACM Symp. Theory of Computing*, Berkeley, CA, 1986, 316–329. See also: J. Kilian, *Uses of randomness in algorithms and protocols*, ACM Distinguished Doctoral Dissertation Series, MIT Press, Cambridge MA, 1990.

R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics.* Addison Wesley, Reading MA, 1989.

M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Algorithms and Combinatorics **2**. Springer, Berlin, Heidelberg, 2nd edition, 1993.

G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*. Clarendon Press, Oxford, 1962.

W. G. Horner, A new method of solving numerical equations of all orders by continuous approximation. *Philosophical Transactions of the Royal Society, London* **109** (1819), 308–335.

A. Karatsuba and Y. Ofman, Умножение многозначных чисел на автоматах. *Dokl. Akad. Nauk USSR* **145** (1962), 293–294. Multiplication of multidigit numbers on automata, Soviet Physics–Doklady **7** (1963), 595-596.

D. E. Knuth, The analysis of algorithms. In *Proc. Int. Congr. Math.*, vol. 3, Nice, 1970, 269–274.

D. E. Knuth, *The Art of Computer Programming, Vol.2, Seminumerical Algorithms*. Addison-Wesley, Reading MA, 2 edition, 1981.

N. Koblitz, *A course in number theory and cryptography*. Springer-Verlag, New York NY, 1987.

D. Kozen and S. Landau, Polynomial decomposition algorithms. *J. Symb. Comp.* **7** (1989), 445–456.

H. T. Kung, On computing reciprocals of power series. *Numer. Math.* **22** (1974), 341–348.

D. J. Lehmann, On primality tests. *SIAM J. Comput.* **11** (1982), 374–375.

F. Lemmermeyer, The Euclidean Algorithm in algebraic number fields. Preprint, 1995.

A. K. Lenstra and H. W. Lenstra, Jr., ed., *The development of the number field sieve*, Lecture Notes in Mathematics **1554**. Springer, 1993.

A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, The number field sieve. In *Proc. 22nd ACM Symp. Theory Comput.*, 1990, 564–572.

R. Lidl and H. Niederreiter, *Finite Fields*, vol. 20 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, Reading MA, 1983.

F. Lindemann, Über die Zahl π. *Mathematische Annalen* **20** (1882), 213–225.

J. D. Lipson, *Elements of Algebra and Algebraic Computing*. Addison-Wesley, 1981.

G. L. Miller, Riemann's hypothesis and tests for primality. *J. Comput. System Sci.* **13** (1976), 300–317.

I. Newton, De analysi per aequationes infinitas. In *The mathematical papers of Isaac Newton*, ed. D. T. Whiteside, vol. II, 206–247. University Press, Cambridge, 1969.

V. Ya. Pan, Methods of computing values of polynomials. *Russ. Math. Surv.* **21** (1966), 105–136.

M. S. Paterson and L. Stockmeyer, On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.* **2** (1973), 60–66.

C. Pomerance, J. Selfridge, and S. S. Wagstaff, Jr., The pseudoprimes to $25 \cdot 10^9$. *Math. Comp.* **35** (1980), 1003–1025.

V. Pratt, Every prime has a succinct certificate. *SIAM J. of Comput.* **4** (1975), 214–220.

M. O. Rabin, Probabilistic algorithms for testing primality. *J. of Number Theory* **12** (1980), 128–138.

D. Reischert, Schnelle Multiplikation von Polynomen über GF(2) und Anwendungen. Diplomarbeit, University of Bonn, Germany, 1995.

R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* **21** (1978), 120–126.

J. B. Rosser and L. Schoenfeld, Approximate formulas for some functions of prime numbers. *Ill. J. Math.* **6** (1962), 64–94.

A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2. *Acta Inf.* **7** (1977), 395–398.

A. Schönhage, *Fast Algorithms – A Multitape Turing Machine Implementation.* BI Wissenschaftsverlag, 1994.

A. Schönhage and V. Strassen, Schnelle Multiplikation großer Zahlen. *Computing* **7** (1971), 281–292.

M. Sieveking, An algorithm for division of powerseries. *Computing* **10** (1972), 153–156.

R. Solovay and V. Strassen, A fast Monte-Carlo test for primality. *SIAM J. Comput.* **6** (1977), 84–85. Erratum, in **7** (1978), 118.

V. Strassen, Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten. *Numer. Math.* **20** (1973), 238–251.

B. L. van der Waerden, *Algebra, Erster Teil.* Springer-Verlag, Berlin, 7th edition, 1966.

D. Y. Y. Yun, On square-free decomposition algorithms. In *Proc. ACM Symp. Symbolic and Algebraic Computation*, ed. R. D. Jenks, 1976, 26–35.

Index

The index entries can be understood as follows:

The reference "10" belongs to a main entry in the text on page 10, e.g. to a definition. The reference "*10*" belongs to an image or a table, while the reference "10" belongs to a simple mention on page 10.