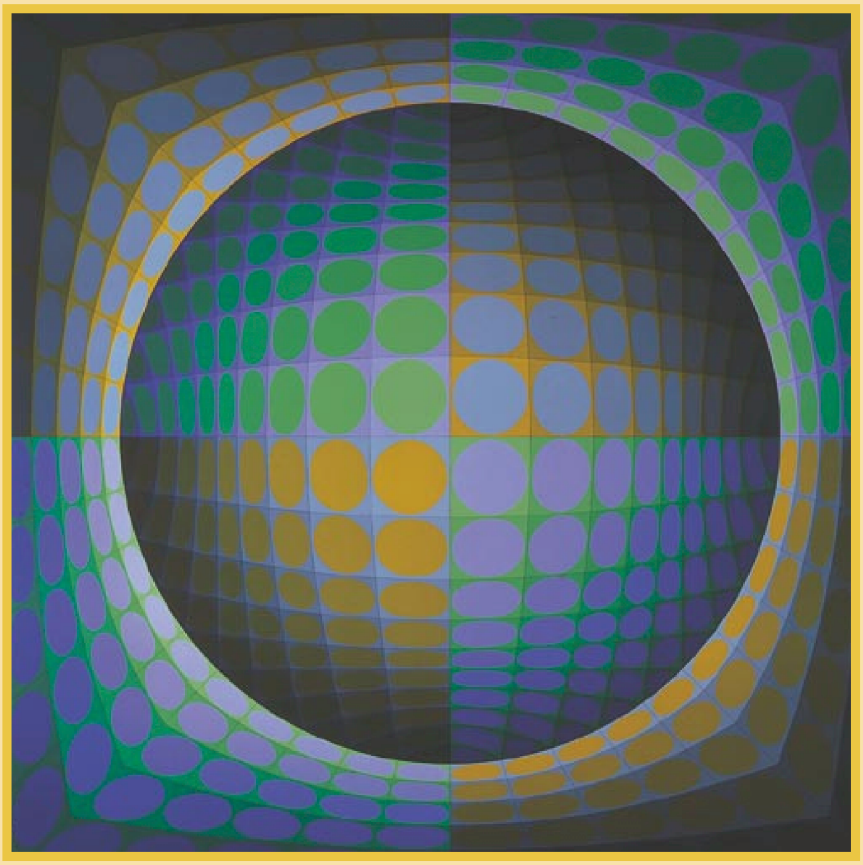
ALGORITHMS of Informatics



volume 1.

ELTE Faculty of Informatics

ALGORITHMS OF INFORMATICS

Volume 1

FOUNDATIONS

ELTE EÖTVÖS KIADÓ
Budapest, 2006

Editor: Antal Iványi

Authors: Gábor Ivanyos and Lajos Rónyai (Chapter 1), Zoltán Kása (2), Jörg Rothe (3 und 4), Recurrences (5), Péter Gács (6)

Validators: Attila Pethő (Chapter 1), Pál Dömösi (2), Lajos Rónyai (3), János Gonda (4), András Recski (5), Anna Gál (6)

© Ingo Althöfer, Ádám Balogh, János Demetrovics, Pál Dömösi, Burkhard Englert, Péter Gács, Aurél Galántai, János Gonda, Tibor Gyires, Csanád Imreh, Antal Iványi, Gábor Ivanyos, András Jeney, Zoltán Kása, Attila Kiss, Dariusz Kowalski, László Lakatos, Claudia Leopold, Kornél Locher, Gregorz Malewicz, János Mayer, András Recski, Lajos Rónyai, Jörg Rothe, Attila Sali, Stefan Schwarz, Alexander Shvartsman, Dezső Sima, Tamás Szántai, Ferenc Szidarovszky, László Szirmay-Kalos, János Sztrik, Ulrich Tamm, László Varga, János Vida, Béla Vizvári, Eberhard Zehendner, 2005

ISBN: 963 463 664 0

Published by ELTE EÖTVÖS KIADÓ
Budapest, Szerb utca 21–23.
Hungary
Telephone/facsimile: 411-6740
Internet: http://www.elte.hu/szervezet/eotvos_kiado.html
E-mail: eotvoskiado@ludens.elte.hu

Responsible publisher: András Pándi
Cover design: Antal Iványi

Printed and bound by ???

Preface

I recommend with a special pleasure into the attention of the Readers the book *Algorithms of Informatics*, carefully edited by Antal Iványi. Computer algorithms form a very important and quickly developing branch of informatics. Design and analysis of big computer nets, large scientific computations and simulations, economic planning, data protection and cryptography and many other applications require effective, carefully planned and precisely analysed algorithms.

Many years ago we wrote a small book with Péter Gács under the title *Algorithms.* The three volumes of the book Algorithms of Computer Science show into how multifold and and branching off area developed the given topic. It gives a special happiness that so many excellent representative of Hungarian informatics have cooperated to create this book. It is obvious for me that this book will be one of the most important source-book of students, researchers and computer users for a long time.

Redmond, April 15, 2005

László Lovász

Introduction

1. Algebra

First, in this chapter, we will discuss some of the basic concepts of algebra, such as fields, vector spaces and polynomials (Section 1.1). Our main focus will be the study of polynomial rings in one variable. These polynomial rings play a very important rôle in *constructive applications*. After this, we will outline the theory of finite fields, putting a strong emphasis on the problem of constructing them (Section 1.2) and on the problem of factoring polynomials over such fields (Section 1.3). Then we will study lattices and discuss the Lenstra-Lenstra-Lovász algorithm which can be used to find short lattice vectors (Section 1.4). We will present a polynomial time algorithm for the factorisation of polynomials with rational coefficients; this was the first notable application of the Lenstra-Lenstra-Lovász algorithm (Section 1.5).

1.1. Fields, vector spaces, and polynomials

In this section we will overview some important concepts related to rings and polynomials.

1.1.1. Ring theoretic concepts

We recall some definitions introduced in Chapters 31-33 of the textbook *Introduction to Algorithms*. In the sequel all cross references to Chapters 31-33 refer to results in that book.

Reference to New Algorithms.

A set S with at least two elements is called a ***ring***, if it has two binary operations, the addition, denoted by the $+$ sign, and the multiplication, denoted by the $\cdot$ sign. The elements of S form an abelian group with respect to the addition, and they form a monoid (that is, a semigroup with an identity), whose identity element is denoted by 1, with respect to the multiplication. We assume that $1 \neq 0$. Further, the distributive properties also hold: for arbitrary elements $a, b, c \in S$ we have

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad \text{and}$$

$$(b + c) \cdot a = b \cdot a + c \cdot a \,.$$

Being an abelian group with respect to the addition means that the operation is associative, commutative, it has an identity element (denoted by 0), and every element has an inverse with respect to this identity. More precisely, these requirements are the following:
associative property: for all triples $a,\ b,\ c \in S$ we have $(a + b) + c = a + (b + c)$;
commutative property: for all pairs $a,\ b \in S$ we have $a + b = b + a$;
existence of the identity element: for the zero element 0 of S and for all elements a of S, we have $a + 0 = 0 + a = a$;
existence of the additive inverse: for all $a \in S$ there exists $b \in S$, such that $a + b = 0$.
It is easy to show that each of the elements a in S has a unique inverse. We usually denote the inverse of an element a by $-a$.

Concerning the multiplication, we require that it must be associative and that the multiplicative identity should exist. The ***identity*** of a ring S is the multiplicative identity of S. The usual name of the additive identity is ***zero***. We usually omit the $\cdot$ sign when writing the multiplication, for example we usually write ab instead of $a \cdot b$.

Example 1.1 *Rings.*
(i) The set $\mathbf{Z}$ of integers with the usual operations + and $\cdot$.
(ii) The set $\mathbf{Z}_m$ of residue classes modulo m with respect to the addition and multiplication modulo m.
(iii) The set $\mathbb{R}^{n\times n}$ of $(n \times n)$-matrices with real entries with respect to the addition and multiplication of matrices.

Let S_1 and S_2 be rings. A map $\phi : S_1 \to S_2$ is said to be a ***homomorphism***, if ϕ preserves the operations, in the sense that $\phi(a \pm b) = \phi(a) \pm \phi(b)$ and $\phi(ab) = \phi(a)\phi(b)$ holds for all pairs $a,\ b \in S_1$. A homomorphism ϕ is called an ***isomorphism***, if ϕ is a one-to-one correspondence, and the inverse is also a homomorphism. We say that the rings S_1 and S_2 are ***isomorphic***, if there is an isomorphism between them. If S_1 and S_2 are isomorphic rings, then we write $S_1 \cong S_2$. From an algebraic point of view, isomorphic rings can be viewed as identical.

For example the map $\phi : \mathbf{Z} \to \mathbf{Z}_6$ which maps an integer to its residue modulo 6 is a homomorphism: $\phi(13) = 1,\ \phi(5) = 5,\ \phi(22) = 4$, etc.

A useful and important ring theoretic construction is the ***direct sum.*** The direct sum of the rings S_1 and S_2 is denoted by $S_1 \oplus S_2$. The underlying set of the direct sum is $S_1 \times S_2$, that is, the set of ordered pairs (s_1, s_2) where $s_i \in S_i$. The operations are defined componentwise: for $s_i,\ t_i \in S_i$ we let

$$(s_1, s_2) + (t_1, t_2) := (s_1 + t_1, s_2 + t_2) \quad \text{and}$$

$$(s_1, s_2) \cdot (t_1, t_2) := (s_1 \cdot t_1, s_2 \cdot t_2)\ .$$

Easy calculation shows that $S_1 \oplus S_2$ is a ring with respect to the operations above. This construction can easily be generalised to more than two rings. In this case, the elements of the direct sum are the k-tuples, where k is the number of rings in the direct sum, and the operations are defined componentwise.

Fields

A ring $\mathbb{F}$ is said to be a ***field***, if its non-zero elements form an abelian group with respect to the multiplication. The multiplicative inverse of a non-zero element a is usually denoted a^{-1}.

The best-known examples of fields are the the sets of rational numbers, real numbers,

and complex numbers with respect to the usual operations. We usually denote these fields by $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, respectively.

Another important class of fields consists of the fields $\mathbb{F}_p$ of p-elements where p is a prime number. The elements of $\mathbb{F}_p$ are the residue classes modulo p, and the operations are the addition and the multiplication defined on the residue classes. The distributive property can easily be derived from the distributivity of the integer operations. By Theorem 33.12, $\mathbb{F}_p$ is a group with respect to the addition, and, by Theorem 33.13, the set $\mathbb{F}_p^*$ of non-zero elements of $\mathbb{F}_p$ is a group with respect to the multiplication. In order to prove this latter claim, we need to use that p is a prime number.

Reference to NA!

Characteristic, prime field

In an arbitrary field, we may consider the set of elements of the form $m \cdot 1$, that is, the set of elements that can be written as the sum $1 + \cdots + 1$ of m copies of the multiplicative identity where m is a positive integer. Clearly, one of the two possibilities must hold:
(a) none of the elements $m \cdot 1$ is zero;
(b) $m \cdot 1$ is zero for some $m \geq 1$.

In case (a) we say that $\mathbb{F}$ is a field with ***characteristic zero***. In case (b) the ***characteristic*** of $\mathbb{F}$ is the smallest $m \geq 1$ such that $m \cdot 1 = 0$. In this case, the number m must be a prime, for, if $m = rs$, then $0 = m \cdot 1 = rs \cdot 1 = (r \cdot 1)(s \cdot 1)$, and so either $r \cdot 1 = 0$ or $s \cdot 1 = 0$.

Suppose that P denotes the smallest subfield of $\mathbb{F}$ that contains 1. Then P is said to be the ***prime field*** of $\mathbb{F}$. In case (a) the subfield P consists of the elements $(m \cdot 1)(s \cdot 1)^{-1}$ where m is an integer and s is a positive integer. In this case, P is isomorphic to the field $\mathbb{Q}$ of rational numbers. The identification is obvious: $(m \cdot 1)(s \cdot 1)^{-1} \leftrightarrow m/s$.

In case (b) the characteristic is a prime number, and P is the set of elements $m \cdot 1$ where $0 \leq m < p$. In this case, P is isomorphic to the field $\mathbb{F}_p$ of residue classes modulo p.

Vector spaces

Let $\mathbb{F}$ be a field. An additively written abelian group V is said to be a ***vector space*** over $\mathbb{F}$, or simply an $\mathbb{F}$-vector space, if for all elements $a \in \mathbb{F}$ and $v \in V$, an element $av \in V$ is defined (in other words, $\mathbb{F}$ acts on V) and the following hold:

$$a(u + v) = au + av, \quad (a + b)u = au + bu \ ,$$

$$a(bu) = (ab)u, \quad 1u = u \ .$$

Here $a,\ b$ are arbitrary elements of $\mathbb{F}$, the elements $u,\ v$ are arbitrary in V, and the element 1 is the multiplicative identity of $\mathbb{F}$.

The space of $(m \times n)$-matrices over $\mathbb{F}$ is an important example of vector spaces. Their properties are studied in Chapter 31.

reference to NA

A vector space V over a field $\mathbb{F}$ is said to be ***finite-dimensional*** if there is a collection $\{v_1, \ldots, v_n\}$ of finitely many elements in V such that each of the elements $v \in V$ can be written as a ***linear combination*** $v = a_1 v_1 + \cdots + a_n v_n$ for some $a_1, \ldots, a_n \in \mathbb{F}$. Such a set $\{v_i\}$ is called a ***generating set*** of V. The cardinality of the smallest generating set of V is referred to as the ***dimension*** of V over $\mathbb{F}$, denoted $\dim_{\mathbb{F}} V$. In a finite-dimensional vector space, a generating system containing $\dim_{\mathbb{F}} V$ elements is said to be a ***basis***.

A set $\{v_1, \ldots, v_k\}$ of elements of a vector space V is said to be ***linearly independent***, if, for $a_1, \ldots, a_k \in \mathbb{F}$, the equation $0 = a_1 v_1 + \cdots + a_k v_k$ implies $a_1 = \cdots = a_k = 0$.

It is easy to show that a basis in V is a linearly independent set. An important property of linearly independent sets is that such a set can be extended to a basis of the vector space. The dimension of a vector space coincides with the cardinality of its largest linearly independent set.

A non-empty subset U of a vector space V is said to be a ***subspace*** of V, if it is an (additive) subgroup of V, and $au \in U$ holds for all $a \in \mathbb{F}$ and $u \in U$. It is obvious that a subspace can be viewed as a vector space.

The concept of homomorphisms can be defined for vector spaces, but in this context we usually refer to them as ***linear maps***. Let V_1 and V_2 be vector spaces over a common field $\mathbb{F}$. A map $\phi : V_1 \to V_2$ is said to be linear, if, for all $a,\ b \in \mathbb{F}$ and $u,\ v \in V_1$, we have

$$\phi(au + bv) = a\phi(u) + b\phi(v) \ .$$

The linear mapping ϕ is an ***isomorphism*** if ϕ is a one-to-one correspondence and its inverse is also a homomorphism. Two vector spaces are said to be isomorphic if there is an isomorphism between them.

Lemma 1.1 *Suppose that $\phi : V_1 \to V_2$ is a linear mapping. Then $U = \phi(V_1)$ is a subspace in V_2. If ϕ is one-to-one, then $\dim_{\mathbb{F}} U = \dim_{\mathbb{F}} V_1$. If, in this case, $\dim_{\mathbb{F}} V_1 = \dim_{\mathbb{F}} V_2 < \infty$, then $U = V_2$ and the mapping ϕ is an isomorphism.*

Proof. As

$$\phi(u) \pm \phi(v) = \phi(u \pm v) \quad \text{and} \quad a\phi(u) = \phi(au),$$

we obtain that U is a subspace. Further, it is clear that the images of the elements of a generating set of V_1 form a generating set for U. Let us now suppose that ϕ is one-to-one. In this case, the image of a linearly independent subset of V_1 is linearly independent in V_2. It easily follows from these observations that the image of a basis of V_1 is a basis of U, and so $\dim_{\mathbb{F}} U = \dim_{\mathbb{F}} V_1$. If we assume, in addition, that $\dim_{\mathbb{F}} V_2 = \dim_{\mathbb{F}} V_1$, then a basis of U is also a basis of V_2, as it is a linearly independent set, and so it can be extended to a basis of V_2. Thus $U = V_2$ and the mapping ϕ must be a one-to-one correspondence. It is easy to see, and is left to the reader, that ϕ^{-1} is a linear mapping. ■

The ***direct sum*** of vector spaces can be defined similarly to the direct sum of rings. The direct sum of the vector spaces V_1 and V_2 is denoted by $V_1 \oplus V_2$. The underlying set of the direct sum is $V_1 \times V_2$, and the addition and the action of the field $\mathbb{F}$ are defined componentwise. It is easy to see that

$$\dim_{\mathbb{F}} (V_1 \oplus V_2) = \dim_{\mathbb{F}} V_1 + \dim_{\mathbb{F}} V_2 \ .$$

Finite multiplicative subgroups of fields

Let $\mathbb{F}$ be a field and let $G \subseteq \mathbb{F}$ be a finite multiplicative subgroup of $\mathbb{F}$. That is, the set G contains finitely many elements of $\mathbb{F}$, each of which is non-zero, G is closed under multiplication, and the multiplicative inverse of an element of G also lies in G. We aim to show that the group G is cyclic, that is, G can be generated by a single element. The main concepts related to cyclic groups can be found in Section 33.3.4. Recall that the order $\mathrm{ord}(a)$ of an element $a \in G$ is the smallest positive integer k such that $a^k = 1$.

Reference to NA

The cyclic group generated by an element a is denoted by $\langle a \rangle$. Clearly, $|\langle a \rangle| = \mathrm{ord}(a)$, and an element a^i generates the group $\langle a \rangle$ if and only if i and n are relatively prime. Hence

the group $\langle a \rangle$ has exactly $\phi(n)$ generators where ϕ is Euler's totient function (see 33.3.2).

Reference to NA!

The following identity is valid for an arbitrary integer n:

$$\sum_{d|n} \phi(d) = n.$$

Here the summation index d runs through all positive divisors of n. In order to verify this identity, consider all the rational numbers i/n with $1 \leq i \leq n$. The number of these is exactly n. After simplifying these fractions, they will be of the form j/d where d is a positive divisor of n. A fixed denominator d will occur exactly $\phi(d)$ times.

Theorem 1.2 *Suppose that $\mathbb{F}$ is a field and let G be a finite multiplicative subgroup of $\mathbb{F}$. Then there exists an element $a \in G$ such that $G = \langle a \rangle$.*

Reference to NA!

Proof. Suppose that $|G| = n$. Lagrange's theorem (Theorem 33.15) implies that the order of an element $b \in G$ is a divisor of n. We claim, for an arbitrary d, that there are at most $\phi(d)$ elements in $\mathbb{F}$ with order d. The elements with order d are roots of the polynomial $x^d - 1$. If $\mathbb{F}$ has an element b with order d, then, by Lemma 1.5, $x^d - 1 = (x - b)(x - b^2) \cdots (x - b^d)$ (the lemma will be verified later). Therefore all the elements of $\mathbb{F}$ with order d are contained in the group $\langle b \rangle$, which, in turn, contains exactly $\phi(d)$ elements of order d.

If G had no element of order n, then the order of each of the elements of G would be a proper divisor of n. In this case, however, using the identity above and the fact that $\phi(n) > 0$, we obtain

$$n = |G| \leq \sum_{d|n,\ d<n} \phi(d) < n\ ,$$

which is a contradiction. ∎

1.1.2. Polynomials

Suppose that $\mathbb{F}$ is a field and that $a_0, \ldots, a_n$ are elements of $\mathbb{F}$. Recall that an expression of the form

$$f = f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

Reference to NA!

where x is an indeterminate, is said to be a ***polynomial*** over $\mathbb{F}$ (see Chapter 32). The scalars a_i are the ***coefficients*** of the polynomial f. The degree of the zero polynomial is zero, while the ***degree*** of a non-zero polynomial f is the largest index j such that $a_j \neq 0$. The degree of f is denoted by $\deg f$.

The set of all polynomials over $\mathbb{F}$ in the indeterminate x is denoted by $\mathbb{F}[x]$. If

$$f = f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

and

$$g = g(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_n x^n$$

are polynomials with degree not larger than n, then their sum is defined as the polynomial

$$h = h(x) = f + g = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$

whose coefficients are $c_i = a_i + b_i$.

The product fg of the polynomials f and g is defined as the polynomial

$$fg = d_0 + d_1x + d_2x^2 + \cdots + d_{2n}x^{2n}$$

with degree at most $2n$ whose coefficients are given by the equations $d_j = \sum_{k=0}^{j} a_k b_{j-k}$. On the right-hand side of these equations, the coefficients with index greater than n are considered zero. Easy computation shows that $\mathbb{F}[x]$ is a commutative ring with respect to these operations. It is also straightforward to show that $\mathbb{F}[x]$ has no ***zero divisors***, that is, whenever $fg = 0$, then either $f = 0$ or $g = 0$.

Division with remainder and divisibility

The ring $\mathbb{F}[x]$ of polynomials over $\mathbb{F}$ is quite similar, in many ways, to the ring $\mathbf{Z}$ of integers. One of their similar features is that the procedure of division with remainder can be performed in both rings.

Lemma 1.3 *Let $f(x),\ g(x) \in \mathbb{F}[x]$ be polynomials such that $g(x) \neq 0$. Then there there exist polynomials $q(x)$ and $r(x)$ such that*

$$f(x) = q(x)g(x) + r(x)\ ,$$

and either $r(x) = 0$ or $\deg\ r(x) < \deg\ g(x)$. Moreover, the polynomials q and r are uniquely determined by these conditions.

Proof. We verify the claim about the existence of the polynomials q and r by induction on the degree of f. If $f = 0$ or $\deg\ f < \deg\ g$, then the assertion clearly holds. Let us suppose, therefore, that $\deg\ f \geq \deg\ g$. Then subtracting a suitable multiple $q^*(x)g(x)$ of g from f, we obtain that the degree of $f_1(x) = f(x) - q^*(x)g(x)$ is smaller than $\deg\ f(x)$. Then, by the induction hypothesis, there exist polynomials q_1 and r_1 such that

$$f_1(x) = q_1(x)g(x) + r_1(x)$$

and either $r_1 = 0$ or $\deg\ r_1 < \deg\ g$. It is easy to see that, in this case, the polynomials $q(x) = q_1(x) + q^*(x)$ and $r(x) = r_1(x)$ are as required.

It remains to show that the polynomials q and r are unique. Let Q and R be polynomials, possibly different from q and r, satisfying the assertions of the lemma. That is, $f(x) = Q(x)g(x) + R(x)$, and so $(q(x) - Q(x))g(x) = R(x) - r(x)$. If the polynomial on the left-hand side is non-zero, then its degree is at least $\deg\ g$, while the degree of the polynomial on the right-hand side is smaller than $\deg\ g$. This, however, is not possible. ∎

Let R be a commutative ring with a multiplicative identity and without zero divisors, and set $R^* := R \setminus \{0\}$. The ring R is said to be a ***Euclidean ring*** if there is a function $\phi : R^* \to \mathbb{N}$ such that $\phi(ab) \geq \phi(a)\phi(b)$, for all $a,\ b \in R^*$; and, further, if $a \in R$, $b \in R^*$, then there are elements $q,\ r \in R$ such that $a = qb + r$, and if $r \neq 0$, then $\phi(r) < \phi(b)$. The previous lemma shows that $\mathbb{F}[x]$ is a Euclidean ring where the rôle of the function ϕ is played by the degree function.

The concept of ***divisibility*** in $\mathbb{F}[x]$ can be defined similarly to the definition of the corresponding concept in the ring of integers. A polynomial $g(x)$ is said to be a ***divisor*** of a polynomial $f(x)$ (the notation is $g \mid f$), if there is a polynomial $q(x) \in \mathbb{F}[x]$ such that $f(x) = q(x)g(x)$. The non-zero elements of $\mathbb{F}$, which are clearly divisors of each of the polynomials, are called the ***trivial divisors*** or ***units***. A non-zero polynomial $f(x) \in \mathbb{F}[x]$ is said

to be ***irreducible***, if whenever $f(x) = q(x)g(x)$ with $q(x),\ g(x) \in \mathbb{F}[x]$, then either q or g is a unit.

Two polynomials $f,\ g \in \mathbb{F}[x]$ are called ***associates***, if there is some $u \in \mathbb{F}^*$ such that $f(x) = ug(x)$.

Using Lemma 1.3, one can easily prove the unique factorisation theorem in the ring of polynomials following the argument of the proof of the corresponding theorem in the ring of integers (see Section 33.1). The rôle of the absolute value of integers is played by the degree of polynomials.

Reference to NA!

Theorem 1.4 *An arbitrary polynomial* $0 \neq f \in \mathbb{F}[x]$ *can be written in the form*

$$f(x) = uq_1(x)^{e_1} \cdots q_r(x)^{e_r},$$

where $u \in \mathbb{F}^*$ *is a unit, the polynomials* $q_i \in \mathbb{F}[x]$ *are pairwise non-associate and irreducible, and, further, the numbers* e_i *are positive integers. Furthermore, this decomposition is essentially unique in the sense that whenever*

$$f(x) = UQ_1(x)^{d_1} \cdots Q_s(x)^{d_s}$$

is another such decomposition, then $r = s$*, and, after possibly reordering the factors* Q_i*, the polynomials* q_i *and* Q_i *are associates, and moreover* $d_i = e_i$ *for all* $1 \leq i \leq r$*.*

Two polynomials are said to be ***relatively prime,*** if they have no common irreducible divisors.

A scalar $a \in \mathbb{F}$ is a ***root*** of a polynomial $f \in \mathbb{F}[x]$, if $f(a) = 0$. Here the value $f(a)$ is obtained by substituting a into the place of x in $f(x)$.

Lemma 1.5 *Suppose that* $a \in \mathbb{F}$ *is a root of a polynomial* $f(x) \in \mathbb{F}[x]$*. Then there exists a polynomial* $g(x) \in \mathbb{F}[x]$ *such that* $f(x) = (x - a)g(x)$*. Hence the polynomial* f *may have at most* deg f *roots.*

Proof. By Lemma 1.3, there exists $g(x) \in \mathbb{F}[x]$ and $r \in \mathbb{F}$ such that $f(x) = (x - a)g(x) + r$. Substituting a for x, we find that $r = 0$. The second assertion now follows by induction on deg f from the fact that the roots of g are also roots of f. ∎

The cost of the operations with polynomials

Suppose that $f(x),\ g(x) \in \mathbb{F}[x]$ are polynomials of degree at most n. Then the polynomials $f(x) \pm g(x)$ can obviously be computed using $O(n)$ field operations. The product $f(x)g(x)$ can be obtained, using its definition, by $O(n^2)$ field operations. If the Fast Fourier Transform can be performed over $\mathbb{F}$, then the multiplication can be computed using only $O(n \lg n)$ field operations (see Theorem 32.2). For general fields, the cost of the fastest known multiplication algorithms for polynomials (for instance the Schönhage-Strassen-method) is $O(n \lg n \lg \lg n)$, that is, $\widetilde{O}(n)$ field operations.

Reference to NA!

The division with remainder, that is, determining the polynomials $q(x)$ and $r(x)$ for which $f(x) = q(x)g(x) + r(x)$ and either $r(x) = 0$ or deg $r(x) <$ deg $g(x)$, can be performed using $O(n^2)$ field operations following the straightforward method outlined in the proof of Lemma 1.3. There is, however, an algorithm (the Sieveking-Kung algorithm) for the same problem using only $\widetilde{O}(n)$ steps. The details of this algorithm are, however, not discussed here.

Congruence, residue class ring

Let $f(x) \in \mathbb{F}[x]$ with $\deg f = n > 0$, and let $g,\ h \in \mathbb{F}[x]$. We say that g is ***congruent*** to h modulo f, or simply $g \equiv h \pmod{f}$, if f divides the polynomial $g - h$. This concept of congruence is similar to the corresponding concept introduced in the ring of integers (see 33.3.2). It is easy to see from the definition that the relation $\equiv$ is an equivalence relation on the set $\mathbb{F}[x]$. Let $[g]_f$ (or simply $[g]$ if f is clear from the context) denote the equivalence class containing g. From Lemma 1.3 we obtain immediately, for each g, that there is a unique $r \in \mathbb{F}[x]$ such that $[g] = [r]$, and either $r = 0$ (if f divides g) or $\deg r < n$. This polynomial r is called the ***representative*** of the class $[g]$. The set of equivalence classes is traditionally denoted by $\mathbb{F}[x]/(f)$.

Reference to NA!

Lemma 1.6 *Let $f,\ f_1,\ f_2,\ g_1,\ g_2 \in \mathbb{F}[x]$ and let $a \in \mathbb{F}$. Suppose that $f_1 \equiv f_2 \pmod{f}$ and $g_1 \equiv g_2 \pmod{f}$. Then*

$$f_1 + g_1 \equiv f_2 + g_2 \pmod{f},$$

$$f_1 g_1 \equiv f_2 g_2 \pmod{f},$$

and

$$af_1 \equiv af_2 \pmod{f}.$$

Proof. The first congruence is valid, as

$$(f_1 + g_1) - (f_2 + g_2) = (f_1 - f_2) + (g_1 - g_2),$$

and the right-hand side of this is clearly divisible by f. The second and the third congruences follow similarly from the identities

$$f_1 g_1 - f_2 g_2 = (f_1 - f_2)g_1 + (g_1 - g_2)f_2$$

and

$$af_1 - af_2 = a(f_1 - f_2),$$

respectively. ∎

The previous lemma makes it possible to define the sum and the product of two congruence classes $[g]_f$ and $[h]_f$ as $[g]_f + [h]_f := [g + h]_f$ and $[g]_f[h]_f := [gh]_f$, respectively. The lemma claims that the sum and the product are independent of the choice of the congruence class representatives. The same way, we may define the action of $\mathbb{F}$ on the set of congruence classes: we set $a[g]_f := [ag]_f$.

Theorem 1.7 *Suppose that $f(x) \in \mathbb{F}[x]$ and that* $\deg f = n > 0$.
(i) The set of residue classes $\mathbb{F}[x]/(f)$ is a commutative ring with an identity under the operations $+$ and $\cdot$ defined above.
(ii) The ring $\mathbb{F}[x]/(f)$ contains the field $\mathbb{F}$ as a subring, and it is an n-dimensional vector space over $\mathbb{F}$. Further, the residue classes $[1],\ [x], \ldots, [x^{n-1}]$ form a basis of $\mathbb{F}[x]/(f)$.
(iii) If f is an irreducible polynomial in $\mathbb{F}[x]$, then $\mathbb{F}[x]/(f)$ is a field.

Proof. (i) The fact that $\mathbb{F}[x]/(f)$ is a ring follows easily from the fact that $\mathbb{F}[x]$ is a ring. Let us, for instance, verify the distributive property:

$$[g]([h_1]+[h_2]) = [g][h_1+h_2] = [g(h_1+h_2)] = [gh_1+gh_2] = [gh_1]+[gh_2] = [g][h_1]+[g][h_2]\ .$$

The zero element of $\mathbb{F}[x]/(f)$ is the class $[0]$, the additive inverse of the class $[g]$ is the class $[-g]$, while the multiplicative identity element is the class $[1]$. The details are left to the reader.

(ii) The set $\{[a] \mid a \in \mathbb{F}\}$ is a subring isomorphic to $\mathbb{F}$. The correspondence is obvious: $a \leftrightarrow [a]$. By part (i), $\mathbb{F}[x]/(f)$ is an additive Abelian group, and the action of $\mathbb{F}$ satisfies the vector space axioms. This follows from the fact that the polynomial ring is itself a vector space over $\mathbb{F}$. Let us, for example, verify the distributive property:

$$a([h_1] + [h_2]) = a[h_1 + h_2] = [a(h_1 + h_2)] = [ah_1 + ah_2] = [ah_1] + [ah_2] = a[h_1] + a[h_2] \ .$$

The other properties are left to the reader.

We claim that the classes $[1],\ [x], \dots, [x^{n-1}]$ are linearly independent. For, if

$$[0] = a_0[1] + a_1[x] + \cdots + a_{n-1}[x^{n-1}] = [a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}] \ ,$$

then $a_0 = \cdots = a_{n-1} = 0$, as the zero polynomial is the unique polynomial with degree less than n that is divisible by f. On the other hand, for a polynomial g, the degree of the class representative of $[g]$ is less than n. Thus the class $[g]$ can be expressed as a linear combination of the classes $[1], [x], \dots, [x^{n-1}]$. Hence the classes $[1], [x], \dots, [x^{n-1}]$ form a basis of $\mathbb{F}[x]/(f)$, and so $\dim_{\mathbb{F}} \mathbb{F}[x]/(f) = n$.

(iii) Suppose that f is irreducible. First we show that $\mathbb{F}[x]/(f)$ has no zero divisors. If $[0] = [g][h] = [gh]$, then f divides gh, and so f divides either g or h. That is, either $[g] = 0$ or $[h] = 0$. Suppose now that $g \in \mathbb{F}[x]$ with $[g] \neq [0]$. We claim that the classes $[g][1], [g][x], \dots, [g][x^{n-1}]$ are linearly independent. Indeed, an equation $[0] = a_0[g][1] + \cdots + a_{n-1}[g][x^{n-1}]$ implies $[0] = [g][a_0 + \cdots + a_{n-1}x^{n-1}]$, and, in turn, it also yields that $a_0 = \cdots = a_{n-1} = 0$. Therefore the classes $[g][1], [g][x], \dots, [g][x^{n-1}]$ form a basis of $\mathbb{F}[x]/(f)$. Hence there exist coefficients $b_i \in \mathbb{F}$ for which

$$[1] = b_0[g][1] + \cdots + b_{n-1}[g][x^{n-1}] = [g][b_0 + \cdots + b_{n-1}x^{n-1}] \ .$$

Thus we find that the class $[0] \neq [g]$ has a multiplicative inverse, and so $\mathbb{F}[x]/(f)$ is a field, as required. ∎

We note that the converse of part (iii) of the previous theorem is also true, and its proof is left to the reader (Exercise 1.1-1.).

Example 1.2 We usually represent the elements of the residue class ring $\mathbb{F}[x]/(f)$ by their representatives, which are polynomials with degree less than $\deg f$.

1. Suppose that $\mathbb{F} = \mathbb{F}_2$ is the field of two elements, and let $f(x) = x^3 + x + 1$. Then the ring $\mathbb{F}[x]/(f)$ has 8 elements, namely

$$[0],\ [1],\ [x],\ [x+1],\ [x^2],\ [x^2+1],\ [x^2+x],\ [x^2+x+1].$$

Practically speaking, the addition between the classes is the is addition of polynomials. For instance

$$[x^2 + 1] + [x^2 + x] = [x + 1] \ .$$

When computing the product, we compute the product of the representatives, and substitute it (or reduce it) with its remainder after dividing by f. For instance,

$$[x^2 + 1] \cdot [x^2 + x] = [x^4 + x^3 + x^2 + x] = [x + 1] \ .$$

The polynomial f is irreducible over $\mathbb{F}_2$, since it has degree 3, and has no roots. Hence the residue class ring $\mathbb{F}[x]/(f)$ is a field.

2. Let $\mathbb{F} = \mathbb{R}$ and let $f(x) = x^2 - 1$. The elements of the residue class ring are the classes of the form $[ax + b]$ where $a,\ b \in \mathbb{R}$. The ring $\mathbb{F}[x]/(f)$ is not a field, since f is not irreducible. For instance, $[x + 1][x - 1] = [0]$.

Lemma 1.8 *Let $\mathbb{L}$ be a field containing a field $\mathbb{F}$ and let $\alpha \in \mathbb{L}$.*
(i) If $\mathbb{L}$ is finite-dimensional as a vector space over $\mathbb{F}$, then there is a non-zero polynomial $f \in \mathbb{F}[x]$ such that α is a root of f.
(ii) Assume that there is a polynomial $f \in \mathbb{F}[x]$ with $f(\alpha) = 0$, and let g be such a polynomial with minimal degree. Then the polynomial g is irreducible in $\mathbb{F}[x]$. Further, if $h \in \mathbb{F}[x]$ with $h(\alpha) = 0$ then g is a divisor of h.

Proof. (i) For a sufficiently large n, the elements $1, \alpha, \ldots, \alpha^n$ are linearly dependent over $\mathbb{F}$. A linear dependence gives a polynomial $0 \neq f \in \mathbb{F}[x]$ such that $f(\alpha) = 0$.

(ii) If $g = g_1 g_2$, then, as $0 = g(\alpha) = g_1(\alpha)g_2(\alpha)$, the element α is a root of either g_1 or g_2. As g was chosen to have minimal degree, one of the polynomials $g_1,\ g_2$ is a unit, and so g is irreducible. Finally, let $h \in \mathbb{F}[x]$ such that $h(\alpha) = 0$. Let $q,\ r \in \mathbb{F}[x]$ be the polynomials as in Lemma 1.3 for which $h(x) = q(x)g(x) + r(x)$. Substituting α for x into the last equation, we obtain $r(\alpha) = 0$, which is only possible if $r = 0$. ∎

Definition 1.9 *The polynomial $g \in \mathbb{F}[x]$ in the last lemma is said to be a* ***minimal polynomial*** *of α.*

It follows from the previous lemma that the minimal polynomial is unique up to a scalar multiple. It will often be helpful to assume that the leading coefficient (the coefficient of the term with the highest degree) of the minimal polynomial g is 1.

Corollary 1.10 *Let $\mathbb{L}$ be a field containing $\mathbb{F}$, and let $\alpha \in \mathbb{L}$. Suppose that $f \in \mathbb{F}[x]$ is irreducible and that $f(\alpha) = 0$. Then f is a minimal polynomial of α.*

Proof. Suppose that g is a minimal polynomial of α. By the previous lemma, $g \mid f$ and g is irreducible. This is only possible if the polynomials f and g are associates. ∎

Let $\mathbb{L}$ be a field containing $\mathbb{F}$ and let $\alpha \in \mathbb{L}$. Let $\mathbb{F}(\alpha)$ denote the smallest subfield of $\mathbb{L}$ that contains $\mathbb{F}$ and α.

Theorem 1.11 *Let $\mathbb{L}$ be a field containing $\mathbb{F}$ and let $\alpha \in \mathbb{L}$. Suppose that $f \in \mathbb{F}[x]$ is a minimal polynomial of α. Then the field $\mathbb{F}(\alpha)$ is isomorphic to the field $\mathbb{F}[x]/(f)$. More precisely, there exists an isomorphism $\phi : \mathbb{F}[x]/(f) \to \mathbb{F}(\alpha)$ such that $\phi(a) = a$, for all $a \in \mathbb{F}$, and $\phi([x]_f) = \alpha$. The map ϕ is also an isomorphism of vector spaces over $\mathbb{F}$, and so* $\dim_{\mathbb{F}} \mathbb{F}(\alpha) = \deg f$.

Proof. Let us consider the map $\psi : \mathbb{F}[x] \to \mathbb{L}$, which maps a polynomial $g \in \mathbb{F}[x]$ into $g(\alpha)$. This is clearly a ring homomorphism, and $\psi(\mathbb{F}[x]) \subseteq \mathbb{F}(\alpha)$. We claim that $\psi(g) = \psi(h)$ if and only if $[g]_f = [h]_f$. Indeed, $\psi(g) = \psi(h)$ holds if and only if $\psi(g - h) = 0$, that is, if $g(\alpha) - h(\alpha) = 0$, which, by Lemma 1.8, is equivalent to $f \mid g - h$, and this amounts to saying that $[g]_f = [h]_f$. Suppose that ϕ is the map $\mathbb{F}[x]/(f) \to \mathbb{F}(\alpha)$ induced by ψ, that is, $\phi([g]_f) := \psi(g)$. By the argument above, the map ϕ is one-to-one. Routine computation

shows that ϕ is a ring, and also a vector space, homomorphism. As $\mathbb{F}[x]/(f)$ is a field, its homomorphic image $\phi(\mathbb{F}[x]/(f))$ is also a field. The field $\phi(\mathbb{F}[x]/(f))$ contains $\mathbb{F}$ and α, and so necessarily $\phi(\mathbb{F}[x]/(f)) = \mathbb{F}(\alpha)$. ∎

Euclidean algorithm and the greatest common divisor

Let $f(x),\ g(x) \in \mathbb{F}[x]$ be polynomials such that $g(x) \neq 0$. Set $f_0 = f$, $f_1 = g$ and define the polynomials q_i and f_i using division with reminder as follows:

$$f_0(x) = q_1(x)f_1(x) + f_2(x)\ ,$$

$$f_1(x) = q_2(x)f_2(x) + f_3(x)\ ,$$

$$\vdots$$

$$f_{k-2}(x) = q_{k-1}(x)f_{k-1}(x) + f_k(x)\ ,$$

$$f_{k-1}(x) = q_k(x)f_k(x) + f_{k+1}(x)\ .$$

Note that if $1 < i < k$ then $\deg f_{i+1}$ is smaller than $\deg f_i$. We form this sequence of polynomials until we obtain that $f_{k+1} = 0$. By Lemma 1.3, this defines a finite process. Let n be the maximum of $\deg f$ and $\deg g$. As, in each step, we decrease the degree of the polynomials, we have $k \leq n + 1$. The computation outlined above is usually referred to as the ***Euclidean algorithm***. A version of this algorithm for the ring of integers is described in Section 33.2.

Reference to NA!

We say that the polynomial $h(x)$ is the ***greatest common divisor*** of the polynomials $f(x)$ and $g(x)$, if $h(x) \mid f(x)$, $h(x) \mid g(x)$, and, if a polynomial $h_1(x)$ is a divisor of f and g, then $h_1(x)$ is a divisor of $h(x)$. The usual notation for the greatest common divisor of $f(x)$ and $g(x)$ is $\gcd(f(x), g(x))$. It follows from Theorem 1.4 that $\gcd(f(x), g(x))$ exists and it is unique up to a scalar multiple.

Theorem 1.12 *Suppose that $f(x),\ g(x) \in \mathbb{F}[x]$ are polynomials, that $g(x) \neq 0$, and let n be the maximum of $\deg f$ and $\deg g$. Assume, further, that the number k and the polynomial f_k are defined by the procedure above. Then*
(i) $\gcd(f(x), g(x)) = f_k(x)$.
(ii) There are polynomials $F(x),\ G(x)$ with degree at most n such that

$$f_k(x) = F(x)f(x) + G(x)g(x)\ . \tag{1.1}$$

(iii) With a given input $f,\ g$, the polynomials $F(x),\ G(x),\ f_k(x)$ can be computed using $O(n^3)$ field operations in $\mathbb{F}$.

Proof. (i) Going backwards in the Euclidean algorithm, it is easy to see that the polynomial f_k divides each of the f_i, and so it divides both f and g. The same way, if a polynomial $h(x)$ divides f and g, then it divides f_i, for all i, and, in particular, it divides f_k. Thus $\gcd(f(x), g(x)) = f_k(x)$.

(ii) The claim is obvious if $f = 0$, and so we may assume without loss of generality that $f \neq 0$. Starting at the beginning of the Euclidean sequence, it is easy to see that there are polynomials $F_i(x),\ G_i(x) \in \mathbb{F}[x]$ such that

$$F_i(x)f(x) + G_i(x)g(x) = f_i(x). \tag{1.2}$$

We observe that (1.2) also holds if we substitute $F_i(x)$ by its remainder $F_i^*(x)$ after dividing by g and substitute $G_i(x)$ by its remainder $G_i^*(x)$ after dividing by f. In order to see this, we compute

$$F_i^*(x)f(x) + G_i^*(x)g(x) \equiv f_i(x) \pmod{f(x)g(x)},$$

and notice that the degree of the polynomials on both sides of this congruence is smaller than $(\deg f)(\deg g)$. This gives

$$F_i^*(x)f(x) + G_i^*(x)g(x) = f_i(x).$$

(iii) Once we determined the polynomials f_{i-1}, f_i, F_i^* and G_i^*, the polynomials f_{i+1}, F_{i+1}^* and G_{i+1}^* can be obtained using $O(n^2)$ field operations in $\mathbb{F}$. Initially we have $F_1^* = 1$ and $G_2^* = -q_1$. As $k \leq n+1$, the claim follows. ∎

Remark. Traditionally, the Euclidean algorithm is only used to compute the greatest common divisor. The version that also computes the polynomials $F(x)$ and $G(x)$ in (1.1) is usually called the extended Euclidean algorithm. In Chapter **??** the reader can find a discussion of the Euclidean algorithm for polynomials. It is relatively easy to see that the polynomials $f_k(x)$, $F(x)$, and $G(x)$ in (1.1) can, in fact, be computed using $O(n^2)$ field operations. The cost of the asymptotically best method is $\widetilde{O}(n)$.

The derivative of a polynomial is often useful when investigating multiple factors. The ***derivative*** of the polynomial

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \in \mathbb{F}[x]$$

is the polynomial

$$f'(x) = a_1 + 2a_2x + \cdots + na_nx^{n-1}\ .$$

It follows immediately from the definition that the map $f(x) \mapsto f'(x)$ is an $\mathbb{F}$-linear mapping $\mathbb{F}[x] \to \mathbb{F}[x]$. Further, for $f(x),\ g(x) \in \mathbb{F}[x]$ and $a \in \mathbb{F}$, the equations $(f(x)+g(x))' = f'(x)+g'(x)$ and $(af(x))' = af'(x)$ hold. The derivative of a product can be computed using the ***Leibniz rule***: for all $f(x),\ g\ (x) \in \mathbb{F}[x]$ we have $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$. As the derivation is a linear map, in order to show that the Leibniz rule is valid, it is enough to verify it for polynomials of the form $f(x) = x^i$ and $g(x) = x^j$. It is easy to see that, for such polynomials, the Leibniz rule is valid.

The derivative $f'(x)$ is sensitive to multiple factors in the irreducible factorisation of $f(x)$.

Lemma 1.13 *Let $\mathbb{F}$ be an arbitrary field, and assume that $f(x) \in \mathbb{F}[x]$ and $f(x) = u^k(x)v(x)$ where $u(x),\ v(x) \in \mathbb{F}[x]$. Then $u^{k-1}(x)$ divides the derivative $f'(x)$ of the polynomial $f(x)$.*

Proof. Using induction on k and the Leibniz rule, we find $(u^k(x))' = ku^{k-1}(x)u'(x)$. Thus, applying the Leibniz rule again, $f'(x) = u^{k-1}(x)(ku'(x)v(x) + u^k(x)v'(x))$. Hence $u^{k-1}(x) \mid f'(x)$. ∎

In many cases the converse of the last lemma also holds.

Lemma 1.14 *Let $\mathbb{F}$ be an arbitrary field, and assume that $f(x) \in \mathbb{F}[x]$ and $f(x) = u(x)v(x)$ where the polynomials $u(x)$ and $v(x)$ are relatively prime. Suppose further that $u'(x) \neq 0$ (for instance $\mathbb{F}$ has characteristic 0 and $u(x)$ is non-constant). Then the derivative $f'(x)$ is not divisible by $u(x)$.*

Proof. By the Leibniz rule, $f'(x) = u(x)v'(x) + u'(x)v(x) \equiv u'(x)v(x) \pmod{u(x)}$. Since $\deg u'(x)$ is smaller than $\deg u(x)$, we obtain that $u'(x)$ is not divisible by $u(x)$, and neither is the product $u'(x)v(x)$, as $u(x)$ and $v(x)$ are relatively prime. ■

The Chinese remainder theorem for polynomials

Using the following theorem, the ring $\mathbb{F}[x]/(f)$ can be assembled from rings of the form $\mathbb{F}[x]/(g)$ where $g \mid f$.

Theorem 1.15 *(Chinese remainder theorem for polynomials) Let $f_1, \ldots, f_k \in \mathbb{F}[x]$ pairwise relatively prime polynomials with positive degree and set $f = f_1 \cdots f_k$. Then the rings $\mathbb{F}[x]/(f)$ and $\mathbb{F}[x]/(f_1) \oplus \cdots \oplus \mathbb{F}[x]/(f_k)$ are isomorphic. The mapping realizing the isomorphism is*

$$\phi : [g]_f \mapsto ([g]_{f_1}, \ldots, [g]_{f_k}), \quad g \in \mathbb{F}[x] \ .$$

Proof. First we note that the map ϕ is well-defined. If $h \in [g]_f$, then $h = g + f^* f$, which implies that h and g give the same remainder after division by the polynomial f_i, that is, $[h]_{f_i} = [g]_{f_i}$.

The mapping ϕ is clearly a ring homomorphism, and it is also a linear mapping between two vector spaces over $\mathbb{F}$. The mapping ϕ is one-to-one; for, if $\phi([g]) = \phi([h])$, then $\phi([g - h]) = (0, \ldots, 0)$, that is, $f_i \mid g - h$ $(1 \le i \le k)$, which gives $f \mid g - h$ and $[g] = [h]$.

The dimensions of the vector spaces $\mathbb{F}[x]/(f)$ and $\mathbb{F}[x]/(f_1) \oplus \cdots \oplus \mathbb{F}[x]/(f_k)$ coincide: indeed, both spaces have dimension $\deg f$. Lemma 1.1 implies that ϕ is an isomorphism between vector spaces. It only remains to show that ϕ^{-1} preserves the multiplication; this, however, is left to the reader. ■

Exercises

1.1-1 Let $f \in \mathbb{F}[x]$ be polynomial. Show that the residue class ring $\mathbb{F}[x]/(f)$ has no zero divisors if and only if f is irreducible.

1.1-2 Let R be a commutative ring with an identity. A subset $I \subseteq R$ is said to be an ***ideal***, if I is an additive subgroup, and $a \in I$, $b \in R$ imply $ab \in I$. Show that R is a field if and only if its ideals are exactly $\{0\}$ and R.

1.1-3 Let $a_1, \ldots, a_k \in R$. Let $(a_1, \ldots, a_k)$ denote the smallest ideal in R that contains the elements a_i. Show that $(a_1, \ldots, a_k)$ always exists, and it consists of the elements of the form $b_1 a_1 + b_2 a_2 + \cdots + b_k a_k$ where $b_1, \ldots, b_k \in R$.

1.1-4 A commutative ring R with an identity and without zero divisors is said to be a ***principal ideal domain*** if, for each ideal I of R, there is an element $a \in I$ such that (using the notation of the previous exercise) $I = (a)$. Show that $\mathbf{Z}$ and $\mathbb{F}[x]$ where $\mathbb{F}$ is a field, are principal ideal domains.

1.1-5 Suppose that S is a commutative ring with an identity, that I an ideal in S, and that $a, b \in S$. Define a relation on S as follows: $a \equiv b \pmod{I}$ if and only if $a - b \in I$. Verify the following:

a.) The relation $\equiv$ is an equivalence relation on S.

b.) Let $[a]_I$ denote the equivalence class containing an element a, and let S/I denote the set of equivalence classes. Set $[a]_I + [b]_I := [a + b]_I$, and $[a]_I[b]_I := [ab]_I$. Show that, with respect to these operations, S/I is a commutative ring with an identity. *Hint:* Follow the argument in the proof of Theorem 1.7.

1.1-6 Let $\mathbb{F}$ be a field and let $f(x), g(x) \in \mathbb{F}[x]$ such that $\gcd(f(x), g(x)) = 1$. Show that there exists a polynomial $h(x) \in \mathbb{F}[x]$ such that $h(x)g(x) \equiv 1 \pmod{f(x)}$. *Hint:* Use the

Euclidean algorithm.

1.2. Finite fields

Finite fields, that is, fields with a finite number of elements, play an important rôle in mathematics and in several of its application areas, for instance, in computing. They are also fundamental in many important constructions. In this section we summarise the most important results in the theory of finite fields, putting an emphasis on the problem of their construction.

In this section p denotes a prime number, and q denotes a power of p with a positive integer exponent.

Theorem 1.16 *Suppose that $\mathbb{F}$ is a finite field. Then there is a prime number p such that the prime field of $\mathbb{F}$ is isomorphic to $\mathbb{F}_p$ (the field of residue classes modulo p). Further, the field $\mathbb{F}$ is a finite dimensional vector space over $\mathbb{F}_p$, and the number of its elements is a power of p. In fact, if $\dim_{\mathbb{F}_p} \mathbb{F} = d$, then $|\mathbb{F}| = p^d$.*

Proof. The characteristic of $\mathbb{F}$ must be a prime, say p, as a field with characteristic zero must have infinitely many elements. Thus the prime field P of $\mathbb{F}$ is isomorphic to $\mathbb{F}_p$. Since P is a subfield, the field $\mathbb{F}$ is a vector space over P. Let $\alpha_1, \ldots, \alpha_d$ be a basis of $\mathbb{F}$ over P. Then each $\alpha \in \mathbb{F}$ can be written uniquely in the form $\sum_{j=1}^{d} a_i \alpha_i$ where $a_i \in P$. Hence $|\mathbb{F}| = p^d$. ■

In a field $\mathbb{F}$, the set of non-zero elements (the multiplicative group of $\mathbb{F}$) is denoted by $\mathbb{F}^*$. From Theorem 1.2 we immediately obtain the following result.

Theorem 1.17 *If $\mathbb{F}$ is a finite field, then its multiplicative group $\mathbb{F}^*$ is cyclic.*

A generator of the group $\mathbb{F}^*$ is said to be a ***primitive element***. If $|\mathbb{F}| = q$ and α is a primitive element of $\mathbb{F}$, then the elements of $\mathbb{F}$ are $0, \alpha, \alpha^2, \ldots, \alpha^{q-1} = 1$.

Corollary 1.18 *Suppose that $\mathbb{F}$ is a finite field with order p^d and let α be a primitive element of $\mathbb{F}$. Let $g \in \mathbb{F}_p[x]$ be a minimal polynomial of α over $\mathbb{F}_p$. Then g is irreducible in $\mathbb{F}_p[x]$, the degree of g is d, and $\mathbb{F}$ is isomorphic to the field $\mathbb{F}_p[x]/(g)$.*

Proof. Since the element α is primitive in $\mathbb{F}$, we have $\mathbb{F} = \mathbb{F}_p(\alpha)$. The rest of the lemma follows from Lemma 1.8 and from Theorem 1.11. ■

Theorem 1.19 *Let $\mathbb{F}$ be a finite field with order q. Then*
(i) (Fermat's little theorem) *If $\beta \in \mathbb{F}^*$, then $\beta^{q-1} = 1$.*
(ii) If $\beta \in \mathbb{F}$, then $\beta^q = \beta$.

Proof. (i) Suppose that $\alpha \in \mathbb{F}^*$ is a primitive element. Then we may choose an integer i such that $\beta = \alpha^i$. Therefore

$$\beta^{q-1} = (\alpha^i)^{q-1} = (\alpha^{q-1})^i = 1^i = 1.$$

(ii) Clearly, if $\beta = 0$ then this claim is true, while, for $\beta \neq 0$, the claim follows from part (i). ■

Theorem 1.20 *Let $\mathbb{F}$ be a field with q elements. Then*

$$x^q - x = \prod_{\alpha \in \mathbb{F}} (x - \alpha) \ .$$

Proof. By Theorem 1.19 and Lemma 1.5, the product on the right-hand side is a divisor of the polynomial $x^q - x \in \mathbb{F}[x]$. Now the assertion follows, as the degrees and the leading coefficients of the two polynomials in the equation coincide. ∎

Corollary 1.21 *Arbitrary two finite fields with the same number of elements are isomorphic.*

Proof. Suppose that $q = p^d$, and that both $\mathbb{K}$ and $\mathbb{L}$ are fields with q elements. Let β be a primitive element in $\mathbb{L}$. Then Corollary 1.18 implies that a minimal polynomial $g(x) \in \mathbb{F}_p[x]$ of β over $\mathbb{F}_p$ is irreducible (in $\mathbb{F}_p[x]$) with degree d. Further, $\mathbb{L} \cong \mathbb{F}_p[x]/(g(x))$. By Lemma 1.8 and Theorem 1.19, the minimal polynomial g is a divisor of the polynomial $x^q - x$. Applying Theorem 1.20 to $\mathbb{K}$, we find that the polynomial $x^q - x$, and also its divisor $g(x)$, can be factored as a product of linear terms in $\mathbb{K}[x]$, and so $g(x)$ has at least one root α in $\mathbb{K}$. As $g(x)$ is irreducible in $\mathbb{F}_p[x]$, it must be a minimal polynomial of α (see Corollary 1.10), and so $\mathbb{F}_p(\alpha)$ is isomorphic to the field $\mathbb{F}_p[x]/(g(x))$. Comparing the number of elements in $\mathbb{F}_p(\alpha)$ and in $\mathbb{K}$, we find that $\mathbb{F}_p(\alpha) = \mathbb{K}$, and further, that $\mathbb{K}$ and $\mathbb{L}$ are isomorphic. ∎

In the sequel, we let $\mathbb{F}_q$ denote the field with q elements, provided it exists. In order to prove the existence of such a field for each prime-power q, the following two facts will be useful.

Lemma 1.22 *If p is a prime number and j is an integer such that $0 < j < p$, then $p \mid \binom{p}{j}$.*

Proof. On the one hand, the number $\binom{p}{j}$ is an integer. On the other hand, $\binom{p}{j} = p(p-1)\cdots(p-j+1)/j!$ is a fraction such that, for $0 < j < p$, its numerator is divisible by p, but its denominator is not. ∎

Lemma 1.23 *Let R be a commutative ring and let p be a prime such that $pr = 0$ for all $r \in R$. Then the map $\Phi_p : R \to R$ mapping $r \mapsto r^p$ is a ring homomorphism.*

Proof. Suppose that $r,\ s \in R$. Clearly,

$$\Phi_p(rs) = (rs)^p = r^p s^p = \Phi_p(r)\Phi_p(s) \ .$$

By the previous lemma,

$$\Phi_p(r+s) = (r+s)^p = \sum_{j=0}^{p} \binom{p}{j} r^{p-j} s^j = r^p + s^p = \Phi_p(r) + \Phi_p(s) \ .$$

We obtain in the same way that $\Phi_p(r-s) = \Phi_p(r) - \Phi_p(s)$. ∎

The homomorphism Φ_p in the previous lemma is called the ***Frobenius endomorphism***.

Theorem 1.24 *Assume that the polynomial $g(x) \in \mathbb{F}_q[x]$ is irreducible, and, for a positive integer d, it is a divisor of the polynomial $x^{q^d} - x$. Then the degree of $g(x)$ divides d.*

Proof. Let n be the degree of $g(x)$, and suppose, by contradiction, that $d = tn + s$ where $0 < s < n$. The assumption that $g(x) \mid x^{q^d} - x$ can be rephrased as $x^{q^d} \equiv x \pmod{g(x)}$. However, this means that, for an arbitrary polynomial $u(x) = \sum_{i=0}^{N} u_i x^i \in \mathbb{F}_q[x]$, we have

$$u(x)^{q^d} = \sum_{i=0}^{N} u_i^{q^d} x^{iq^d} = \sum_{i=0}^{N} u_i (x^{q^d})^i \equiv \sum_{i=0}^{N} u_i x^i = u(x) \pmod{g(x)}.$$

Note that we applied Lemma 1.23 to the ring $R = \mathbb{F}_q[x]/(g(x))$, and Theorem 1.19 to $\mathbb{F}_q$. The residue class ring $\mathbb{F}_q[x]/(g(x))$ is isomorphic to the field $\mathbb{F}_{q^n}$, which has q^n elements. Let $u(x) \in \mathbb{F}_q[x]$ be a polynomial for which $u(x) \pmod{g(x)}$ is a primitive element in the field $\mathbb{F}_{q^n}$. That is, $u(x)^{q^n-1} \equiv 1 \pmod{g(x)}$, but $u(x)^j \not\equiv 1 \pmod{g(x)}$ for $j = 1, \ldots, q^n - 2$. Therefore,

$$u(x) \equiv u(x)^{q^d} = u(x)^{q^{m+s}} = (u(x)^{q^{nt}})^{q^s} \equiv u(x)^{q^s} \pmod{g(x)},$$

and so $u(x)(u(x)^{q^s-1} - 1) \equiv 0 \pmod{g(x)}$. Since the residue class ring $\mathbb{F}_q[x]/(g(x))$ is a field, $u(x) \not\equiv 0 \pmod{g(x)}$, but we must have $u(x)^{q^s-1} \equiv 1 \pmod{g(x)}$. As $0 \le q^s - 1 < q^n - 1$, this contradicts to the primitivity of $u(x) \pmod{g(x)}$. ■

Theorem 1.25 *For an arbitrary prime p and positive integer d, there exists a field with p^d elements.*

Proof. We use induction on d. The claim clearly holds if $d = 1$. Now let $d > 1$ and let r be a prime divisor of d. By the induction hypothesis, there is a field with $q = p^{(d/r)}$ elements. By Theorem 1.24, each of the irreducible factors, in $\mathbb{F}_q[x]$, of the the polynomial $f(x) = x^{q^r} - x$ has degree either 1 or r. Further, $f'(x) = (x^{q^r} - x)' = -1$, and so, by Lemma 1.13, $f(x)$ is square-free. Over $\mathbb{F}_q$, the number of linear factors of $f(x)$ is at most q, and so is the degree of their product. Hence there exist at least $(q^r - q)/r \ge 1$ polynomials with degree r that are irreducible in $\mathbb{F}_q[x]$. Let $g(x)$ be such a polynomial. Then the field $\mathbb{F}_q[x]/(g(x))$ is isomorphic to the field with $q^r = p^d$ elements. ■

Corollary 1.26 *For each positive integer d, there is an irreducible polynomial $f \in \mathbb{F}_p[x]$ with degree d.*

Proof. Take a minimal polynomial over $\mathbb{F}_p$ of a primitive element in $\mathbb{F}_{p^d}$. ■

A little bit later, in Theorem 1.31, we will prove a stronger statement: a random polynomial in $\mathbb{F}_p[x]$ with degree d is irreducible with high probability.

Subfields of finite fields

The following theorem describes all subfields of a finite field.

Theorem 1.27 *The field $\mathbb{F} = \mathbb{F}_{p^n}$ contains a subfield isomorphic to $\mathbb{F}_{p^k}$, if and only if $k \mid n$. In this case, there is exactly one subfield in $\mathbb{F}$ that is isomorphic to $\mathbb{F}_{p^k}$.*

Proof. The condition that $k \mid n$ is necessary, since the larger field is a vector space over the smaller field, and so $p^n = (p^k)^l$ must hold with a suitable integer l.

Conversely, suppose that $k \mid n$, and let $f \in \mathbb{F}_p[x]$ be an irreducible polynomial with degree k. Such a polynomial exists by Corollary 1.26. Let $q = p^k$. Applying Theorem 1.19, we obtain, in $\mathbb{F}_p[x]/(f)$, that $x^q \equiv x \pmod{f}$, which yields $x^{p^n} = x^{q^l} \equiv x \pmod{f}$. Thus f must be a divisor of the polynomial $x^{p^n} - x$. Using Theorem 1.20, we find that f has a root

α in $\mathbb{F}$. Now we may prove in the usual way that the subfield $\mathbb{F}_p(\alpha)$ is isomorphic to $\mathbb{F}_{p^k}$.

The last assertion is valid, as the elements of $\mathbb{F}_q$ are exactly the roots of $x^q - x$ (Theorem 1.20), and this polynomial can have, in an arbitrary field, at most q roots. ■

The structure of irreducible polynomials

Next we prove an important property of the irreducible polynomials over finite fields.

Theorem 1.28 *Assume that $\mathbb{F}_q \subseteq \mathbb{F}$ are finite fields, and let $\alpha \in \mathbb{F}$. Let $f \in \mathbb{F}_q[x]$ be the minimal polynomial of α over $\mathbb{F}_q$ with leading coefficient 1, and suppose that* $\deg f = d$. *Then*

$$f(x) = (x - \alpha)(x - \alpha^q) \cdots (x - \alpha^{q^{d-1}}).$$

Moreover, the elements $\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}$ are pairwise distinct.

Proof. Let $f(x) = a_0 + a_1 x + \cdots + x^d$. If $\beta \in \mathbb{F}$ with $f(\beta) = 0$, then, using Lemma 1.23 and Theorem 1.19, we obtain

$$0 = f(\beta)^q = (a_0 + a_1\beta + \cdots + \beta^d)^q = a_0^q + a_1^q\beta^q + \cdots + \beta^{dq} = a_0 + a_1\beta^q + \cdots + \beta^{qd} = f(\beta^q).$$

Thus β^q is also a root of f.

As α is a root of f, the argument in the previous paragraph shows that so are the elements $\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}$. Hence it suffices to show, that they are pairwise distinct. Suppose, by contradiction, that $\alpha^{q^i} = \alpha^{q^j}$ and that $0 \le i < j < d$. Let $\beta = \alpha^{q^i}$ and let $l = j - i$. By assumption, $\beta = \beta^{q^l}$, which, by Lemma 1.8, means that $f(x) \mid x^{q^l} - x$. From Theorem 1.24, we obtain, in this case, that $d \mid l$, which is a contradiction, as $l < d$. ■

This theorem shows that a polynomial f which is irreducible over a finite field cannot have multiple roots. Further, all the roots of f can be obtained from a single root taking q-th powers repeatedly.

Automorphisms

In this section we characterise certain automorphisms of finite fields.

Definition 1.29 *Suppose that $\mathbb{F}_q \subseteq \mathbb{F}$ are finite fields. The map $\Psi : \mathbb{F} \to \mathbb{F}$ is an $\mathbb{F}_q$-**automorphism** of the field $\mathbb{F}$, if it is an isomorphism between rings, and $\Psi(a) = a$ holds for all $a \in \mathbb{F}_q$.*

Recall that the map $\Phi = \Phi_q : \mathbb{F} \to \mathbb{F}$ is defined as follows: $\Phi(\alpha) = \alpha^q$ where $\alpha \in \mathbb{F}$.

Theorem 1.30 *The set of $\mathbb{F}_q$-automorphisms of the field $\mathbb{F} = \mathbb{F}_{q^d}$ is formed by the maps $\Phi, \Phi^2, \ldots, \Phi^d = id$.*

Proof. By Lemma 1.23, the map $\Phi : \mathbb{F} \to \mathbb{F}$ is a ring homomorphism. The map Φ is obviously one-to-one, and hence it is also an isomorphism. It follows from Theorem 1.19, that Φ leaves the elements $\mathbb{F}_q$ fixed. Thus the maps Φ^j are $\mathbb{F}_q$-automorphisms of $\mathbb{F}$.

Suppose that $f(x) = a_0 + a_1 x + \cdots + x^d \in \mathbb{F}_q[x]$, and $\beta \in \mathbb{F}$ with $f(\beta) = 0$, and that Ψ is an $\mathbb{F}_q$-automorphism of $\mathbb{F}$. We claim that $\Psi(\beta)$ is a root of f. Indeed,

$$0 = \Psi(f(\beta)) = \Psi(a_0) + \Psi(a_1)\Psi(\beta) + \cdots + \Psi(\beta)^d = f(\Psi(\beta)) \ .$$

Let β be a primitive element of $\mathbb{F}$ and assume now that $f \in \mathbb{F}_q[x]$ is a minimal polynomial of β. By the observation above and by Theorem 1.28, $\Psi(\beta) = \beta^{q^j}$, with some $0 \le j < d$, that is, $\Psi(\beta) = \Phi^j(\beta)$. Hence the images of a generating element of $\mathbb{F}$ under the automorphisms Ψ and Φ^j coincide, which gives $\Psi = \Phi^j$. ■

The construction of finite fields

Let $q = p^n$. By Theorem 1.7 and Corollary 1.26, the field $\mathbb{F}_q$ can be written in the form $\mathbb{F}[x]/(f)$, where $f \in \mathbb{F}[x]$ is an irreducible polynomial with degree n. In practical applications of field theory, for example in computer science, this is the most common method of constructing a finite field. Using, for instance, the polynomial $f(x) = x^3 + x + 1$ in Example 1.2., we may construct the field $\mathbb{F}_8$. The following theorem shows that we have a good chance of obtaining an irreducible polynomial by a random selection.

Theorem 1.31 *Let $f(x) \in \mathbb{F}_q[x]$ be a uniformly distributed random polynomial with degree $k > 1$ and leading coefficient 1. (Being uniformly distributed means that the probability of choosing f is $1/q^k$.) Then f is irreducible over $\mathbb{F}_q$ with probability at least $1/k - 1/q^{k/2}$.*

Proof. First we estimate the number of elements $\alpha \in \mathbb{F}_{q^k}$ for which $\mathbb{F}_q(\alpha) = \mathbb{F}_{q^k}$. We claim that the number of such elements is at least

$$|\mathbb{F}_{q^k}| - \sum_{r|k} |\mathbb{F}_{q^{k/r}}| \,,$$

where the summation runs for the distinct prime divisors r of k. Indeed, if α does not generate, over $\mathbb{F}_q$, the field $\mathbb{F}_{q^k}$, then it is contained in a maximal subfield of $\mathbb{F}_{q^k}$, and these maximal subfields are, by Theorem 1.27, exactly the fields of the form $\mathbb{F}_{q^{k/r}}$. The number of distinct prime divisors of k are at most $\lg k$, and so the number of such elements α is at least $q^k - (\lg k)q^{k/2}$. The minimal polynomials with leading coefficients 1 over $\mathbb{F}_q$ of such elements α have degree k and they are irreducible. Such a polynomial is a minimal polynomial of exactly k elements α (Theorem 1.28). Hence the number of distinct irreducible polynomials with degree k and leading coefficient 1 in $\mathbb{F}_q[x]$ is at least

$$\frac{q^k}{k} - \frac{(\lg k)q^{k/2}}{k} \geq \frac{q^k}{k} - q^{k/2} \,,$$

from which the claim follows. ∎

If, having $\mathbb{F}_q$, we would like to construct one of its extensions $\mathbb{F}_{q^k}$, then it is worth selecting a ***random*** polynomial

$$f(x) = a_0 + a_1 x + \cdots + a_{k-1}x^{k-1} + x^k \in \mathbb{F}_q[x].$$

In other words, we select uniformly distributed random coefficients $a_0, \ldots, a_{k-1} \in \mathbb{F}_q$ independently. The polynomial so obtained is irreducible with a high probability (in fact, with probability at least $1/k - \epsilon$ if q^k is large). Further, in this case, we also have $\mathbb{F}_q[x]/(f) \cong \mathbb{F}_{q^k}$. We expect that we will have to select about k polynomials before we find an irreducible one.

We have seen in Theorem 1.2 that field extensions can be obtained using irreducible polynomials. It is often useful if these polynomials have some further nice properties. The following lemma claims the existence of such polynomials.

Lemma 1.32 *Let r be a prime. In a finite field $\mathbb{F}_q$ there exists an element which is not an r-th power if and only if $q \equiv 1 \pmod{r}$. If $b \in \mathbb{F}_q$ is such an element, then the polynomial $x^r - b$ is irreducible in $\mathbb{F}_q[x]$, and so $\mathbb{F}_q[x]/(x^r - b)$ is a field with q^r elements.*

Proof. Suppose first that $r \nmid q-1$ and let s be a positive integer such that $sr \equiv 1 \pmod{q-1}$. If $b \in \mathbb{F}_q$ such that $b \neq 0$, then $(b^s)^r = b^{sr} = bb^{sr-1} = b$, while if $b = 0$, then $b = 0^r$. Hence,

in this case, each of the elements of $\mathbb{F}_q$ is an r-th power.

Next we assume that $r \mid q-1$, and we let a be a primitive element in $\mathbb{F}_q$. Then, in $\mathbb{F}_q$, the r-th powers are exactly the following $1+(q-1)/r$ elements: $0, (a^r)^0, (a^r)^1, \ldots, (a^r)^{(q-1)/r-1}$. Suppose now that $r^s \mid q-1$, but $r^{s+1} \nmid q-1$. Then the order of an element $b \in \mathbb{F}_q \setminus \{0\}$ is divisible by r^s if and only if b is not an r-th power. Let b be such an element, and let $g(x) \in \mathbb{F}_q[x]$ be an irreducible factor of the polynomial $x^r - b$. Suppose that the degree of $g(x)$ is d; clearly, $d \le r$. Then $\mathbb{K} = \mathbb{F}_q[x]/(g(x))$ is a field with q^d elements and, in $\mathbb{K}$, the equation $[x]^r = b$ holds. Therefore the order of $[x]$ is divisible by r^{s+1}. Consequently, $r^{s+1} \mid q^d - 1$. As $q-1$ is not divisible by r^{s+1}, we have $r \mid (q^d-1)/(q-1) = 1+q+\cdots+q^{d-1}$. In other words $1+q+\ldots+q^{d-1} \equiv 0 \pmod{r}$. On the other hand, as $q \equiv 1 \pmod{r}$, we find $1+q+\cdots+q^{d-1} \equiv d \pmod{r}$, and hence $d \equiv 0 \pmod{r}$, which, since $0 < d \le r$, can only happen if $d = r$. ∎

In certain cases, we can use the previous lemma to boost the probability of finding an irreducible polynomial.

Proposition 1.33 *Let r be a prime such that $r \mid q-1$. Then, for a random element $b \in \mathbb{F}_q^*$, the polynomial $x^r - b$ is irreducible in $\mathbb{F}_q[x]$ with probability at least $1 - 1/r$.*

Proof. Under the conditions, the r-th powers in $\mathbb{F}_q^*$ constitute the cyclic subgroup with order $(q-1)/r$. Thus a random element $b \in \mathbb{F}_q^*$ is an r-th power with probability $1/r$, and hence the assertion follows from Lemma 1.32. ∎

Remark. Assume that $r \mid (q-1)$, and, if $r = 2$, then assume also that $4 \mid (q-1)$. In this case there is an element b in $\mathbb{F}_q$ that is not an r-th power. We claim that that the residue class $[x]$ is not an r-th power in $\mathbb{F}_q[x]/(x^r - b) \cong \mathbb{F}_q^r$. Indeed, by the argument in the proof of Lemma 1.32, it suffices to show that $r^2 \nmid (q^r-1)/(q-1)$. By our assumptions, this is clear if $r = 2$. Now assume that $r > 2$, and write $q \equiv 1 + rt \pmod{r^2}$. Then, for all integers $i \ge 0$, we have $q^i \equiv 1 + irt \pmod{r^2}$, and so, by the assumptions,

$$\frac{q^r-1}{q-1} = 1+q+\cdots+q^{r-1} \equiv r + \frac{r(r-1)}{2}rt \equiv r \pmod{r^2}\,.$$

Exercises

1.2-1 Show that the polynomial $x^{q+1} - 1$ can be factored as a product of linear factors over the field $\mathbb{F}_{q^2}$.

1.2-2 Show that the polynomial $f(x) = x^4 + x + 1$ is irreducible over $\mathbb{F}_2$, that is, $\mathbb{F}_2[x]/(f) \cong \mathbb{F}_{16}$. What is the order of the element $[x]_f$ in the residue class ring? Is it true that the element $[x]_f$ is primitive in $\mathbb{F}_{16}$?

1.2-3 Determine the irreducible factors of $x^{31} - 1$ over the field $\mathbb{F}_2$.

1.2-4 Determine the subfields of $\mathbb{F}_{3^6}$.

1.2-5 Let a and b be positive integers. Show that there exists a finite field $\mathbb{K}$ containing $\mathbb{F}_q$ such that $\mathbb{F}_{q^a} \subseteq \mathbb{K}$ and $\mathbb{F}_{q^b} \subseteq \mathbb{K}$. What can we say about the number of elements in $\mathbb{K}$?

1.2-6 Show that the number of irreducible polynomials with degree k and leading coefficient 1 over $\mathbb{F}_q$ is at most q^k/k.

1.2-7 *(a)* Let $\mathbb{F}$ be a field, let V be an n-dimensional vector space over $\mathbb{F}$, and let $A : V \to V$ be a linear transformation whose minimal polynomial coincides with its characteristic polynomial. Show that there exists a vector $v \in V$ such that the images $v, Av, \ldots, A^{n-1}v$ are linearly independent.

(b) A set $S = \{\alpha, \alpha^q, \ldots, \alpha^{q^{d-1}}\}$ is said to be a ***normal basis*** of $\mathbb{F}_{q^d}$ over $\mathbb{F}_q$, if $\alpha \in \mathbb{F}_{q^d}$ and S is a linearly independent set over $\mathbb{F}_q$. Show that $\mathbb{F}_{q^d}$ has a normal basis over $\mathbb{F}_q$. *Hint:* Show that a minimal polynomial of the $\mathbb{F}_q$-linear map $\Phi : \mathbb{F}_{q^d} \to \mathbb{F}_{q^d}$ is $x^d - 1$, and use part (a).

1.3. Factoring polynomials over finite fields

One of the problems that we often have to solve when performing symbolic computation is the ***factorisation*** problem. Factoring an algebraic expression means writing it as a product of simpler expressions. Experience shows that this can be very helpful in the solution of a large variety of algebraic problems. In this section, we consider a class of factorisation algorithms that can be used to factor polynomials in one variable over finite fields.

The input of the ***polynomial factorisation problem*** is a polynomial $f(x) \in \mathbb{F}_q[x]$. Our aim is to compute a factorisation

$$f = f_1^{e_1} f_2^{e_2} \cdots f_s^{e_s} \tag{1.3}$$

of f where the polynomials $f_1, \ldots, f_s$ are pairwise relatively prime and irreducible over $\mathbb{F}_q$, and the exponents e_i are positive integers. By Theorem 1.4, f determines the polynomials f_i and the exponents e_i essentially uniquely.

Example 1.3 Let $p = 23$ and let

$$f(x) = x^6 - 3x^5 + 8x^4 - 11x^3 + 8x^2 - 3x + 1 \,.$$

Then it is easy to compute modulo 23 that

$$f(x) = (x^2 - x + 10)(x^2 + 5x + 1)(x^2 - 7x + 7) \,.$$

None of the factors $x^2 - x + 10$, $x^2 + 5x + 1$, $x^2 - 7x + 7$ has a root in $\mathbb{F}_{23}$, and so they are necessarily irreducible in $\mathbb{F}_{23}[x]$.

The factorisation algorithms are important computational tools, and so they are implemented in most of the computer algebra systems (Mathematica, Maple, etc). These algorithms are often used in the area of error-correcting codes and in cryptography.

Our aim in this section is to present some of the basic ideas and building blocks that can be used to factor polynomials over finite fields. We will place an emphasis on the existence of polynomial time algorithms. The discussion of the currently best known methods is, however, outside the scope of this book.

1.3.1. Square-free factorisation

The factorisation problem in the previous section can efficiently be reduced to the special case when the polynomial f to be factored is square-free; that is, in (1.3), $e_i = 1$ for all i. The basis of this reduction is Lemma 1.13 and the following simple result. Recall that the derivative of a polynomial $f(x)$ is denoted by $f'(x)$.

Lemma 1.34 *Let $f(x) \in \mathbb{F}_q[x]$ be a polynomial. If $f'(x) = 0$, then there exists a polynomial $g(x) \in \mathbb{F}_q[x]$ such that $f(x) = g(x)^p$.*

Proof. Suppose that $f(x) = \sum_{i=0}^{n} a_i x^i$. Then $f'(x) = \sum_{i=1}^{n} a_i i x^{i-1}$. If the coefficient $a_i i$ is zero in $\mathbb{F}_q$ then either $a_i = 0$ or $p \mid i$. Hence, if $f'(x) = 0$ then $f(x)$ can be written as $f(x) = \sum_{j=0}^{k} b_j x^{pj}$. Let $q = p^d$; then choosing $c_j = b_j^{p^{d-1}}$, we have $c_j^p = b_j^{p^d} = b_j$, and so $f(x) = (\sum_{j=0}^{k} c_j x^j)^p$. ■

If $f'(x) = 0$, then, using the previous lemma, a factorisation of $f(x)$ into square-free factors can be obtained from that of the polynomial $g(x)$, which has smaller degree. On the other hand, if $f'(x) \neq 0$, then, by Lemma 1.13, the polynomial $f(x)/\gcd(f(x), f'(x))$ is already square-free and we only have to factor $\gcd(f(x), f'(x))$ into square-free factors. The division of polynomials and computing the greatest common divisor can be performed in polynomial time, by Theorem 1.12. In order to compute the polynomial $g(x)$, we need the solutions, in $\mathbb{F}_q$, of equations of the form $y^p = a$ with $a \in \mathbb{F}_q$. If $q = p^s$, then $y = a^{p^{s-1}}$ is a solution of such an equation, which, using ***fast exponentiation*** (repeated squaring, see 33.6.1), can be obtained in polynomial time.

Reference to NA!

One of the two reduction steps can always be performed if f is divisible by a square of a polynomial with positive degree.

Usually a polynomial can be written as a product of square-free factors in many different ways. For the sake of uniqueness, we define the ***square-free factorisation*** of a polynomial $f \in \mathbb{F}[x]$ as the factorisation

$$f = f_1^{e_1} \cdots f_s^{e_s} ,$$

where $e_1 < \cdots < e_s$ are integers, and the polynomials f_i are relatively prime and square-free. Hence we collect together the irreducible factors of f with the same multiplicity. The following algorithm computes a square-free factorisation of f. Besides the observations we made in this section, we also use Lemma 1.14. This lemma, combined with Lemma 1.13, guarantees that the product of the irreducible factors with multiplicity one of a polynomial f over a finite field is $f/\gcd(f, f')$.

SQUARE-FREE-FACTORISATION(f)

```
1  g ← f
2  S ← ∅
3  m ← 1
4  i ← 1
5  while deg g ≠ 0
6        do if g' = 0
7              then g ← ∛g
8                   i ← i · p
9              else h ← g/gcd(g, g')
10                  g ← g/h
11                  if deg h ≠ 0
12                     then S ← S ∪ (h, m)
13                  m ← m + i
14 return S
```

The degree of the polynomial g decreases after each execution of the main loop, and the subroutines used in this algorithm run in polynomial time. Thus the method above can

be performed in polynomial time.

1.3.2. Distinct degree factorisation

Suppose that f is a square-free polynomial. Now we factor f as

$$f(x) = h_1(x)h_2(x)\cdots h_t(x)\,, \tag{1.4}$$

where, for $i = 1, \ldots, t$, the polynomial $h_i(x) \in \mathbb{F}_q[x]$ is a product of irreducible polynomials with degree i. Though this step is not actually necessary for the solution of the factorisation problem, it is worth considering, as several of the known methods can efficiently exploit the structure of the polynomials h_i. The following fact serves as the starting point of the distinct degree factorisation.

Theorem 1.35 *The polynomial $x^{q^d} - x$ is the product of all the irreducible polynomials $f \in \mathbb{F}_q[x]$, each of which is taken with multiplicity 1, that have leading coefficient 1 and whose degree divides d.*

Proof. As $(x^{q^d} - x)' = -1$, all the irreducible factors of this polynomial occur with multiplicity one. If $f \in \mathbb{F}_q[x]$ is irreducible and divides $x^{q^d} - x$, then, by Theorem 1.24, the degree of f divides d.

Conversely, let $f \in \mathbb{F}_q[x]$ be an irreducible polynomial with degree k such that $k \mid d$. Then, by Theorem 1.27, f has a root in $\mathbb{F}_{q^d}$, which implies $f \mid x^{q^d} - x$. ∎

The theorem offers an efficient method for computing the polynomials $h_i(x)$. First we separate h_1 from f, and then, step by step, we separate the product of the factors with higher degrees.

DISTINCT-DEGREE-FACTORISATION(f)

```
1  F ← f
3  for i ← 1 to deg f
4      do h_i ← gcd(F, x^{q^i} − x)
7         F ← F/h_i
8  return h_1, ..., h_{deg f}
```

If, in this algorithm, the polynomial $F(x)$ is constant, then we may stop, as the further steps will not give new factors. As the polynomial $x^{q^i} - x$ may have large degree, computing $\gcd(F(x), x^{q^i} - x)$ must be performed with particular care. The important idea here is that the residue $x^{q^i} \pmod{F(x)}$ can be computed using fast exponentiation.

The algorithm outlined above is suitable for testing whether a polynomial is irreducible, which is one of the important problems that we encounter when constructing finite fields. The algorithm presented here for distinct degree factorisation can solve this problem efficiently. For, it is obvious that a polynomial f with degree k is irreducible, if, in the factorisation (1.4), we have $h_k(x) = f(x)$.

The following algorithm for testing whether a polynomial is irreducible is somewhat more efficient than the one sketched in the previous paragraph and handles correctly also the inputs that are not square-free.

IRREDUCIBILITY-TEST(f)

```
1  n ← deg f
2  if x^{p^n} ≢ x (mod f)
3     then return "no"
4  for the prime divisors r of n
5     do if x^{p^{n/r}} ≡ x (mod f)
6           then return "no"
7  return "yes"
```

In lines 2 and 5, we check whether n is the smallest among the positive integers k for which f divides $x^{q^k} - x$. By Theorem 1.35, this is equivalent to the irreducibility of f. If f survives the test in line 2, then, by Theorem 1.35, we know that f is square-free and k must divide n. Using at most $\lg n + 1$ fast exponentiations modulo f, we can thus decide if f is irreducible.

Theorem 1.36 *If the field $\mathbb{F}_q$ is given and $k > 1$ is an integer, then the field $\mathbb{F}_{q^k}$ can be constructed using a randomised Las Vegas algorithm which runs in time polynomial in $\lg q$ and k.*

Proof. The algorithm is the following.

FINITE-FIELD-CONSTRUCTION(q^k)

```
1  for i ← 0 to k − 1
2     do a_i ← a random element (uniformly distributed) of F_q
3  f ← x^k + Σ_{i=0}^{k−1} a_i x^i
4  if IRREDUCIBILITY-TEST(f) = "yes"
5     then return F_q[x]/(f)
6     else return "fail"
```

In lines 1–3, we choose a uniformly distributed random polynomial with leading coefficient 1 and degree k. Then, in line 4, we efficiently check if $f(x)$ is irreducible. By Theorem 1.31, the polynomial f is irreducible with a reasonably high probability. ■

1.3.3. The Cantor-Zassenhaus algorithm

In this section we consider the special case of the factorisation problem in which q is odd and the polynomial $f(x) \in \mathbb{F}_q[x]$ is of the form

$$f = f_1 f_2 \cdots f_s, \tag{1.5}$$

where the f_i are pairwise relatively prime irreducible polynomials in $\mathbb{F}_q[x]$ with the same degree d, and we also assume that $s \geq 2$. Our motivation for investigating this special case is that a square-free distinct degree factorisation reduces the general factorisation problem to such a simpler problem. If q is even, then Berlekamp's method, presented in Section 1.3.4, gives a deterministic polynomial time solution. There is a variation of the method discussed in the present section that works also for even q; see Exercise *1-2*..

The second dot after Exercise 18-2 comes from a macro!

Lemma 1.37 *Suppose that q is odd. Then there are $(q^2-1)/2$ pairs $(c_1, c_2) \in \mathbb{F}_q \times \mathbb{F}_q$ such that exactly one of $c_1^{(q-1)/2}$ and $c_2^{(q-1)/2}$ is equal to 1.*

Proof. Suppose that a is a primitive element in $\mathbb{F}_q$; that is, $a^{q-1} = 1$, but $a^k \neq 1$ for $0 < k < q-1$. Then $\mathbb{F}_q \setminus \{0\} = \{a^s | s = 0, \ldots, q-2\}$, and further, as $\left(a^{(q-1)/2}\right)^2 = 1$, but $a^{(q-1)/2} \neq 1$, we obtain that $a^{(q-1)/2} = -1$. Therefore $a^{s(q-1)/2} = (-1)^s$, and so half of the element $c \in \mathbb{F}_q \setminus \{0\}$ give $c^{(q-1)/2} = 1$, while the other half give $c^{(q-1)/2} = -1$. If $c = 0$ then clearly $c^{(q-1)/2} = 0$. Thus there are $((q-1)/2)((q+1)/2)$ pairs (c_1, c_2) such that $c_1^{(q-1)/2} = 1$, but $c_2^{(q-1)/2} \neq 1$, and, obviously, we have the same number of pairs for which the converse is valid. Thus the number of pairs that satisfy the condition is $(q-1)(q+1)/2 = (q^2-1)/2$. ∎

Theorem 1.38 *Suppose that q is odd and the polynomial $f(x) \in \mathbb{F}_q[x]$ is of the form (1.5) and has degree n. Choose a uniformly distributed random polynomial $u(x) \in \mathbb{F}_q[x]$ with degree less than n. (That is, choose pairwise independent, uniformly distributed scalars $u_0, \ldots, u_{n-1}$, and consider the polynomial $u(x) = \sum_{i=0}^{n-1} u_i x^i$.) Then, with probability at least $(q^{2d}-1)/(2q^{2d}) \geq 4/9$, the greatest common divisor*

$$\gcd(u(x)^{\frac{q^d-1}{2}} - 1, f(x))$$

is a proper divisor of $f(x)$.

Proof. The element $u(x) \pmod{f_i(x)}$ corresponds to an element of the residue class field $\mathbb{F}[x]/(f_i(x)) \cong \mathbb{F}_{q^d}$. By the Chinese remainder theorem (Theorem 1.15), choosing the polynomial $u(x)$ uniformly implies that the residues of $u(x)$ modulo the factors $f_i(x)$ are independent and uniformly distributed random polynomials. By Lemma 1.37, the probability that exactly one of the residues of the polynomial $u(x)^{(q^d-1)/2} - 1$ modulo $f_1(x)$ and $f_2(x)$ is zero is precisely $(q^{2d}-1)/(2q^{2d})$. In this case the greatest common divisor in the theorem is indeed a divisor of f. For, if $u(x)^{(q^d-1)/2} - 1 \equiv 0 \pmod{f_1(x)}$, but this congruence is not valid modulo $f_2(x)$, then the polynomial $u(x)^{(q^d-1)/2} - 1$ is divisible by the factor $f_1(x)$, but not divisible by $f_2(x)$, and so its greatest common divisor with $f(x)$ is a proper divisor of $f(x)$. The function

$$\frac{q^{2d}-1}{2q^{2d}} = \frac{1}{2} - \frac{1}{2q^{2d}}$$

is strictly increasing in q^d, and it takes its smallest possible value if q^d is the smallest odd prime-power, namely 3. The minimum is, thus, $1/2 - 1/18 = 4/9$. ∎

The previous theorem suggests the following randomised Las Vegas polynomial time algorithm for factoring a polynomial of the form (1.5) to a product of two factors.

CANTOR-ZASSENHAUS-ODD(f, d)

```
1  n ← deg f
2  for i ← 0 to n − 1
3      do u_i ← a random element (uniformly distributed) of F_q
4  u ← Σ_{i=0}^{n−1} u_i x^i
5  g ← gcd(u^{(q^d−1)/2} − 1, f)
6  if 0 < deg g < deg f
7     then return(g, f/g)
8     else return "fail"
```

If one of the polynomials in the output is not irreducible, then, as it is of the form (1.5), it can be fed, as input, back into the algorithm. This way we obtain a polynomial time randomised algorithm for factoring f.

In the computation of the greatest common divisor, the residue $u(x)^{(q^d-1)/2} \pmod{f(x)}$ should be computed using fast exponentiation.

Now we can conclude that the general factorisation problem (1.3) over a field with odd order can be solved using a randomised polynomial time algorithm.

1.3.4. Berlekamp's algorithm

Here we will describe an algorithm that reduces the problem of factoring polynomials to the problem of searching through the underlying field or its prime field. We assume that

$$f(x) = f_1^{e_1}(x) \cdots f_s^{e_s}(x) \ ,$$

where the $f_i(x)$ are pairwise non-associate, irreducible polynomials in $\mathbb{F}_q[x]$, and also that $\deg f(x) = n$. The Chinese remainder theorem (Theorem 1.15) gives an isomorphism between the rings $\mathbb{F}_q[x]/(f)$ and

$$\mathbb{F}_q[x]/(f_1^{e_1}) \oplus \cdots \oplus \mathbb{F}_q[x]/(f_s^{e_s}).$$

The isomorphism is given by the following map:

$$[u(x)]_f \leftrightarrow ([u(x)]_{f_1^{e_1}}, \ldots, [u(x)]_{f_s^{e_s}}) \ ,$$

where $u(x) \in \mathbb{F}_q[x]$.

The most important technical tools in Berlekamp's algorithm are the p-th and q-th power maps in the residue class ring $\mathbb{F}_q[x]/(f(x))$. Taking p-th and q-th powers on both sides of the isomorphism above given by the Chinese remainder theorem, we obtain the following maps:

$$[u(x)]^p \quad \leftrightarrow \quad ([u(x)^p]_{f_1^{e_1}}, \ldots, [u(x)^p]_{f_s^{e_s}}) \ , \tag{1.6}$$

$$[u(x)]^q \quad \leftrightarrow \quad ([u(x)^q]_{f_1^{e_1}}, \ldots, [u(x)^q]_{f_s^{e_s}}) \ . \tag{1.7}$$

The ***Berlekamp subalgebra*** B_f of the polynomial $f = f(x)$ is the subring of the residue class ring $\mathbb{F}_q[x]/(f)$ consisting of the fixed points of the q-th power map. Further, the ***absolute Berlekamp subalgebra*** A_f of f consists of the fixed points of the p-th power map. In symbols,

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u(x)^q]_f = [u(x)]_f\} \ ,$$

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u(x)^p]_f = [u(x)]_f\} \ .$$

It is easy to see that $A_f \subseteq B_f$. The term subalgebra is used here, because both types of Berlekamp subalgebras are subrings in the residue class ring $\mathbb{F}_q[x]/(f(x))$ (that is they are closed under addition and multiplication modulo $f(x)$), and, in addition, B_f is also linear

subspace over $\mathbb{F}_q$, that is, it is closed under multiplication by the elements of $\mathbb{F}_q$. The absolute Berlekamp subalgebra A_f is only closed under multiplication by the elements of the prime field $\mathbb{F}_p$.

The Berlekamp subalgebra B_f is a subspace, as the map $u \mapsto u^q - u \pmod{f(x)}$ is an $\mathbb{F}_q$-linear map of $\mathbb{F}_q[x]/g(x)$ into itself, by Lemma 1.23 and Theorem 1.19. Hence a basis of B_f can be computed as a solution of a homogeneous system of linear equations over $\mathbb{F}_q$, as follows.

For all $i \in \{0, \ldots, n-1\}$, compute the polynomial $h_i(x)$ with degree at most $n-1$ that satisfies $x^{iq} - x^i \equiv h_i(x) \pmod{f(x)}$. For each i, such a polynomial h_i can be determined by fast exponentiation using $O(\lg q)$ multiplications of polynomials and divisions with remainder. Set $h_i(x) = \sum_{j=0}^{n} h_{ij} x^j$. The class $[u]_f$ of a polynomial $u(x) = \sum_{i=0}^{n-1} u_i x^i$ with degree less than n lies in the Berlekamp subalgebra if and only if

$$\sum_{i=0}^{n-1} u_i h_i(x) = 0 \,,$$

which, considering the coefficient of x^j for $j = 0, \ldots, n-1$, leads to the following system of n homogeneous linear equations in n variables:

$$\sum_{i=0}^{n-1} h_{ij} u_i = 0, \quad (j = 0, \ldots, n-1) \,.$$

Similarly, computing a basis of the absolute Berlekamp subalgebra over $\mathbb{F}_p$ can be carried out by solving a system of nd homogeneous linear equations in nd variables over the prime field $\mathbb{F}_p$, as follows. We represent the elements of $\mathbb{F}_q$ in the usual way, namely using polynomials with degree less than d in $\mathbb{F}_p[y]$. We perform the operations modulo $g(y)$, where $g(y) \in \mathbb{F}_p[y]$ is an irreducible polynomial with degree d over the prime field $\mathbb{F}_p$. Then the polynomial $u[x] \in \mathbb{F}_q[x]$ of degree less than n can be written in the form

$$\sum_{i=0}^{n-1} \sum_{j=0}^{d-1} u_{ij} y^j x^i,$$

where $u_{ij} \in \mathbb{F}_p$. Let, for all $i \in \{0, \ldots, n-1\}$ and for all $j \in \{0, \ldots, d-1\}$, $h_{ij}(x) \in \mathbb{F}_q[x]$ be the unique polynomial with degree at most $(n-1)$ for which $h_{ij}(x) \equiv (y^j x^i)^p - y^j x^i \pmod{f(x)}$. The polynomial $h_{ij}(x)$ is of the form $\sum_{k=0}^{n-1} \sum_{l=0}^{d-1} h_{ij}^{kl} y^l x^k$. The criterion for being a member of the absolute Berlekamp subalgebra of $[u]$ with $u[x] = \sum_{i=0}^{n-1} \sum_{j=0}^{d-1} u_{ij} y^j x^i$ is

$$\sum_{i=0}^{n-1} \sum_{j=0}^{d-1} u_{ij} h_{ij}(x) = 0 \,,$$

which, considering the coefficients of the monomials $y^l x^k$, is equivalent to the following system of equations:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{d-1} h_{ij}^{kl} u_{ij} = 0 \quad (k = 0, \ldots, n-1,\ l = 0, \ldots, d-1) \,.$$

This is indeed a homogeneous system of linear equations in the variables u_{ij}. Systems of

linear equations over fields can be solved in polynomial time (see Section 31.4), the operations in the ring $\mathbb{F}_q[x]/(f(x))$ can be performed in polynomial time, and the fast exponentiation also runs in polynomial time. Thus the following theorem is valid.

Reference to NA!

Theorem 1.39 *Let $f \in \mathbb{F}_q[x]$. Then it is possible to compute the Berlekamp subalgebras $B_f \leq \mathbb{F}_q[x]/(f(x))$ and $A_f \leq \mathbb{F}_q[x]/(f(x))$, in the sense that an $\mathbb{F}_q$-basis of B_f and $\mathbb{F}_p$-basis of A_f can be obtained, using polynomial time deterministic algorithms.*

By (1.6) and (1.7),

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u^q(x)]_{f_i^{e_i}} = [u(x)]_{f_i^{e_i}} \ (i = 1, \ldots, s)\} \tag{1.8}$$

and

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ [u^p(x)]_{f_i^{e_i}} = [u(x)]_{f_i^{e_i}} \ (i = 1, \ldots, s)\} \ . \tag{1.9}$$

The following theorem shows that the elements of the Berlekamp subalgebra can be characterised by their Chinese remainders.

Theorem 1.40

$$B_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ \exists c_i \in \mathbb{F}_q \text{ such that } [u(x)]_{f_i^{e_i}} = [c_i]_{f_i^{e_i}} \ (i = 1, \ldots, s)\}$$

and

$$A_f = \{[u(x)]_f \in \mathbb{F}_q[x]/(f) : \ \exists c_i \in \mathbb{F}_p \text{ such that } [u(x)]_{f_i^{e_i}} = [c_i]_{f_i^{e_i}} \ (i = 1, \ldots, s)\} \ .$$

Proof. Using the Chinese remainder theorem, and equations (1.8), (1.9), we are only required to prove that

$$u^q(x) \equiv u(x) \pmod{g^e(x)} \Longleftrightarrow \exists c \in \mathbb{F}_q \text{ such that } u(x) \equiv c \pmod{g^e(x)} \ ,$$

and

$$u^p(x) \equiv u(x) \pmod{g^e(x)} \Longleftrightarrow \exists c \in \mathbb{F}_p \text{ such that } u(x) \equiv c \pmod{g^e(x)}$$

where $g(x) \in \mathbb{F}_q[x]$ is an irreducible polynomial, $u(x) \in \mathbb{F}_q[x]$ is an arbitrary polynomial and e is a positive integer. In both of the cases, the direction $\Leftarrow$ is a simple consequence of Theorem 1.19. As $\mathbb{F}_p = \{a \in \mathbb{F}_q \mid a^p = a\}$, the implication $\Rightarrow$ concerning the absolute Berlekamp subalgebra follows from that concerning the Berlekamp subalgebra, and so it suffices to consider the latter.

The residue class ring $\mathbb{F}_q[x]/(g(x))$ is a field, and so the polynomial $x^q - x$ has at most q roots in $\mathbb{F}_q[x]/(g(x))$. However, we already obtain q distinct roots from Theorem 1.19, namely the elements of $\mathbb{F}_q$ (the constant polynomials modulo $g(x)$). Thus

$$u^q(x) \equiv u(x) \pmod{g(x)} \Longleftrightarrow \exists c \in \mathbb{F}_q \text{ such that } u(x) \equiv c \pmod{g(x)} \ .$$

Hence, if $u^q(x) \equiv u(x) \pmod{g^e(x)}$, then $u(x)$ is of the form $u(x) = c + h(x)g(x)$ where $h(x) \in \mathbb{F}_q[x]$. Let N be an arbitrary positive integer. Then

$$u(x) \equiv u^q(x) \equiv u^{q^N}(x) \equiv (c + h(x)g(x))^{q^N} \equiv c + h(x)^{q^N} g(x)^{q^N} \equiv c \pmod{g^{q^N}(x)} \ .$$

If we choose N large enough so that $q^N \geq e$ holds, then, by the congruence above, $u(x) \equiv c \pmod{g^e(x)}$ also holds. ■

An element $[u(x)]_f$ of B_f or A_f is said to be ***non-trivial*** if there is no element $c \in \mathbb{F}_q$ such that $u(x) \equiv c \pmod{f(x)}$. By the previous theorem and the Chinese remainder theorem, this holds if and only if there are i, j such that $c_i \neq c_j$. Clearly a necessary condition is that $s > 1$, that is, $f(x)$ must have at least two irreducible factors.

Lemma 1.41 *Let $[u(x)]_f$ be a non-trivial element of the Berlekamp subalgebra B_f. Then there is an element $c \in \mathbb{F}_q$ such that the polynomial* $\gcd(u(x) - c, f(x))$ *is a proper divisor of $f(x)$. If $[u(x)]_f \in A_f$, then there exists such an element c in the prime field $\mathbb{F}_p$.*

Proof. Let i and j be integers such that $c_i \neq c_j \in \mathbb{F}_q$, $u(x) \equiv c_i \pmod{f_i^{e_i}(x)}$, and $u(x) \equiv c_j \pmod{f_j^{e_j}(x)}$. Then, choosing $c = c_i$, the polynomial $u(x) - c$ is divisible by $f_i^{e_i}(x)$, but not divisible by $f_j^{e_j}(x)$. If, in addition, $u(x) \in A_f$, then also $c = c_i \in \mathbb{F}_p$. ■

Assume that we have a basis of A_f at hand. At most one of the basis elements can be trivial, as a trivial element is a scalar multiple of 1. If $f(x)$ is not a power of an irreducible polynomial, then there will surely be a non-trivial basis element $[u(x)]_f$, and so, using the idea in the previous lemma, $f(x)$ can be factored two factors.

Theorem 1.42 *A polynomial $f(x) \in \mathbb{F}_q[x]$ can be factored with a deterministic algorithm whose running time is polynomial in p,* $\deg f$, *and* $\lg q$.

Proof. It suffices to show that f can be factored to *two* factors within the given time bound. The method can then be repeated.

BERLEKAMP-DETERMINISTIC(f)

```
1  S ← a basis of A_f
2  if |S| > 1
3     then u ← a non-trivial element of S
4     for c ∈ F_p
5        do g ← gcd(u − c, f)
6           if 0 < deg g < deg f
7              then return (g, f/g)
8     else return "a power of an irreducible"
```

In the first stage, in line 1, we determine a basis of the absolute Berlekamp subalgebra. The cost of this is polynomial in $\deg f$ and $\lg q$. In the second stage (lines 2–8), after taking a non-trivial basis element $[u(x)]_f$, we compute the greatest common divisors $\gcd(u(x) - c, f(x))$ for all $c \in \mathbb{F}_p$. The cost of this is polynomial in p and $\deg f$.

If there is no non-trivial basis-element, then A_f is 1-dimensional and f is the e_1-th power of the irreducible polynomial f_1 where f_1 and e_1 can, for instance, be determined using the ideas presented in Section 1.3.1. ■

The time bound in the previous theorem is *not polynomial* in the input size, as it contains p instead of $\lg p$. However, if p is small compared to the other parameters (for instance in coding theory we often have $p = 2$), then the running time of the algorithm will be polynomial in the input size.

Corollary 1.43 *Suppose that p can be bounded by a polynomial function of* $\deg f$ *and*

lg q. *Then the irreducible factorisation of f can be obtained in polynomial time.*

The previous two results are due to E. R. Berlekamp. The most important open problem in the area discussed here is the existence of a deterministic polynomial time method for factoring polynomials. The question is mostly of theoretical interest, since the randomised polynomial time methods, such as the a Cantor -Zassenhaus algorithm, are very efficient in practice.

Berlekamp's randomised algorithm

We can obtain a good randomised algorithm using Berlekamp subalgebras. Suppose that q is odd, and, as before, $f \in \mathbb{F}_q[x]$ is the polynomial to be factored.

Let $[u(x)]_f$ be a random element in the Berlekamp subalgebra B_f. An argument, similar to the one in the analysis of the Cantor-Zassenhaus algorithm shows that, provided $f(x)$ has at least two irreducible factors, the greatest common divisor $\gcd(u(x)^{(q-1)/2} - 1, f(x))$ is a proper divisor of $f(x)$ with probability at least 4/9. Now we present a variation of this idea that uses less random bits: instead of choosing a random element from B_f, we only choose a random element from $\mathbb{F}_q$.

Lemma 1.44 *Suppose that q is odd and let a_1 and a_2 be two distinct elements of $\mathbb{F}_q$. Then there are at least $(q-1)/2$ elements $b \in \mathbb{F}_q$ such that exactly one of the elements $(a_1+b)^{(q-1)/2}$ and $(a_2+b)^{(q-1)/2}$ is* 1.

Proof. Using the argument at the beginning of the proof of Lemma 1.37, one can easily see that there are $(q-1)/2$ elements in the set $\mathbb{F}_q \setminus \{1\}$ whose $(q-1)/2$-th power is -1. It is also quite easy to check, for a given element $c \in \mathbb{F}_q \setminus \{1\}$, that there is a unique $b \neq -a_2$ such that $c = (a_1+b)/(a_2+b)$. Indeed, the required b is the solution of a linear equation.

By the above, there are $(q-1)/2$ elements $b \in \mathbb{F}_q \setminus \{-a_2\}$ such that

$$\left(\frac{a_1+b}{a_2+b}\right)^{(q-1)/2} = -1 \ .$$

For such a b, one of the elements $(a_1+b)^{(q-1)/2}$ and $(a_2+b)^{(q-1)/2}$ is equal to 1 and the other is equal to -1. ∎

Theorem 1.45 *Suppose that q is odd and the polynomial $f(x) \in \mathbb{F}_q[x]$ has at least two irreducible factors in $\mathbb{F}_q[x]$. Let $u(x)$ be a non-trivial element in the Berlekamp subalgebra B_f. If we choose a uniformly distributed random element $b \in \mathbb{F}_q$, then, with probability at least $(q-1)/(2q) \geq 1/3$, the greatest common divisor* $\gcd((u(x)+b)^{(q-1)/2} - 1, f(x))$ *is a proper divisor of the polynomial $f(x)$.*

Proof. Let $f(x) = \prod_{i=1}^{s} f_i^{e_i}(x)$, where the factors $f_i(x)$ are pairwise distinct irreducible polynomials. The element $[u(x)]_f$ is a non-trivial element of the Berlekamp subalgebra, and so there are indices $0 < i, j \leq s$ and elements $c_i \neq c_j \in \mathbb{F}_q$ such that $u(x) \equiv c_i \pmod{f_i^{e_i}(x)}$ and $u(x) \equiv c_j \pmod{f_j^{e_j}(x)}$. Using Lemma 1.44 with $a_1 = c_i$ and $a_2 = c_j$, we find, for a random element $b \in \mathbb{F}_q$, that the probability that exactly one of the elements $(c_i+b)^{(q-1)/2} - 1$ and $(c_j+b)^{(q-1)/2} - 1$ is zero is at least $(q-1)/(2q)$. If, for instance, $(c_i+b)^{(q-1)/2} - 1 = 0$, but $(c_j+b)^{(q-1)/2} - 1 \neq 0$, then $(u(x)+b)^{(q-1)/2} - 1 \equiv 0 \pmod{f_i^{e_i}(x)}$ but $(u(x)+b)^{(q-1)/2} - 1 \neq 0$

$(\mathrm{mod}\ f_j^{e_j}(x))$, that is, the polynomial $(u(x)+b)^{(q-1)/2}-1$ is divisible by $f_i^{e_i}(x)$, but not divisible by $f_j^{e_j}(x)$. Thus the greatest common divisor $\gcd(f(x),(u(x)+b)^{(q-1)/2}-1)$ is a proper divisor of f.

The quantity $(q-1)/(2q) = 1/2 - 1/(2q)$ is a strictly increasing function in q, and so it takes its smallest value for the smallest odd prime-power, namely 3. The minimum is $1/3$. ■

The previous theorem gives the following algorithm for factoring a polynomial to two factors.

BERLEKAMP-RANDOMISED(f)

```
1  S ← a basis of B_f
2  if |S| > 1
3     then u ← a non-trivial elements of S
4          c ← a random element (uniformly distributed) of F_q
5          g ← gcd((u − c)^{(q−1)/2} − 1, f)
6          if 0 < deg g < deg f
7             then return (g, f/g)
8             else return "fail"
9     else return "a power of an irreducible"
```

Exercises

1.3-1 Let $f(x) \in \mathbb{F}_p[x]$ be an irreducible polynomial, and let α be an element of the field $\mathbb{F}_p[x]/(f(x))$. Give a polynomial time algorithm for computing α^{-1}. *Hint:* Use the result of Exercise 1.1-6.

1.3-2 Let $f(x) = x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{F}_2[x]$. Using the DISTINCT-DEGREE-FACTORISATION algorithm, determine the factorisation (1.4) of f.

1.3-3 Follow the steps of the Cantor-Zassenhaus algorithm to factor the polynomial $x^2 + 2x + 9 \in \mathbb{F}_{11}[x]$.

1.3-4 Let $f(x) = x^2 - 3x + 2 \in \mathbb{F}_5[x]$. Show that $\mathbb{F}_5[x]/(f(x))$ coincides with the absolute Berlekamp subalgebra of f, that is, $A_f = \mathbb{F}_5[x]/(f(x))$.

1.3-5 Let $f(x) = x^3 - x^2 + x - 1 \in \mathbb{F}_7[x]$. Using Berlekamp's algorithm, determine the irreducible factors of f: first find a non-trivial element in the Berlekamp subalgebra A_f, then use it to factor f.

1.4. Lattice reduction

Our aim in the rest of this chapter is to present the Lenstra-Lenstra-Lovász algorithm for factoring polynomials with rational coefficients. First we study a geometric problem, which is interesting also in its own right, namely finding short lattice vectors. Finding a shortest non-zero lattice vector is hard: by a result of Ajtai, if this problem could be solved in polynomial time with a randomised algorithm, then so could all the problems in the complexity class *NP*. For a lattice with dimension n, the lattice reduction method presented in this chapter outputs, in polynomial time, a lattice vector whose length is not greater than $2^{(n-1)/4}$ times the length of a shortest non-zero lattice vector.

1.4.1. Lattices

First, we recall a couple of concepts related to real vector spaces. Let $\mathbb{R}^n$ denote the collection of real vectors of length n. It is routine to check that $\mathbb{R}^n$ is a vector space over the field $\mathbb{R}$. The ***scalar product*** of two vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ in $\mathbb{R}^n$ is defined as the number $(u, v) = u_1v_1 + u_2v_2 + \cdots + u_nv_n$. The quantity $|u| = \sqrt{(u, u)}$ is called the ***length*** of the vector u. The vectors u and v are said to be ***orthogonal*** if $(u, v) = 0$. A basis $b_1, \ldots, b_n$ of the space $\mathbb{R}^n$ is said to be orthonormal, if, for all i, $(b_i, b_i) = 1$ and, for all i and j such that $i \neq j$, we have $(b_i, b_j) = 0$.

The rank and the determinant of a real matrix, and definite matrices are discussed in Section 31.1.

Reference to NA!

Definition 1.46 *A set $L \subseteq \mathbb{R}^n$ is said to be a **lattice**, if L is a subgroup with respect to addition, and L is discrete, in the sense that each bounded region of $\mathbb{R}^n$ contains only finitely many points of L. The **rank** of the lattice L is the dimension of the subspace generated by L. Clearly, the rank of L coincides with the cardinality of a maximal linearly independent subset of L. If L has rank n, then L is said to be a **full** lattice. The elements of L are called **lattice vectors** or **lattice points**.*

Definition 1.47 *Let $b_1, \ldots, b_r$ be linearly independent elements of a lattice $L \subseteq \mathbb{R}^n$. If all the elements of L can be written as linear combinations of the elements $b_1, \ldots, b_r$ with integer coefficients, then the collection $b_1, \ldots, b_r$ is said to be a **basis** of L.*

In this case, as the vectors $b_1, \ldots, b_r$ are linearly independent, all vectors of $\mathbb{R}^n$ can uniquely be written as real linear combinations of $b_1, \ldots, b_r$.

By the following theorem, the lattices are precisely those additive subgroups of $\mathbb{R}^n$ that have bases.

Theorem 1.48 *Let $b_1, \ldots, b_r$ be linearly independent vectors in $\mathbb{R}^n$ and let L be the set of integer linear combinations of $b_1, \ldots, b_r$. Then L is a lattice and the vectors $b_1, \ldots, b_r$ form a basis of L. Conversely, if L is a lattice in $\mathbb{R}^n$, then it has a basis.*

Proof. Obviously, L is a subgroup, that is, it is closed under addition and subtraction. In order to show that it is discrete, let us assume that $n = r$. This assumption means no loss of generality, as the subspace spanned by $b_1, \ldots, b_r$ is isomorphic to $\mathbb{R}^r$. In this case, $\phi : (\alpha_1, \ldots, \alpha_n) \mapsto \alpha_1 b_1 + \ldots + \alpha_n b_n$ is an invertible linear map of $\mathbb{R}^n$ onto itself. Consequently, both ϕ and ϕ^{-1} are continuous. Hence the image of a discrete set under ϕ is also discrete. As $L = \phi(\mathbf{Z}^n)$, it suffices to show that $\mathbf{Z}^n$ is discrete in $\mathbb{R}^n$. This, however, is obvious: if K is a bounded region in $\mathbb{R}^n$, then there is a positive integer ρ, such that the absolute value of each of the coordinates of the elements of K is at most ρ. Thus $\mathbf{Z}^n$ has at most $(2\lfloor\rho\rfloor + 1)^n$ elements in K.

The second assertion is proved by induction on n. If $L = \{0\}$, then we have nothing to prove. Otherwise, by discreteness, there is a shortest non-zero vector, b_1 say, in L. We claim that the vectors of L that lie on the line $\{\lambda b_1 \mid \lambda \in \mathbb{R}\}$ are exactly the integer multiples of b_1. Indeed, suppose that λ is a real number and consider the vector $\lambda b_1 \in L$. As usual, $\{\lambda\}$ denotes the fractional part of λ. Then $0 \neq |\{\lambda\}b_1| < |b_1|$, yet $\{\lambda\}b_1 = \lambda b_1 - [\lambda]b_1$, that is $\{\lambda\}b_1$ is the difference of two vectors of L, and so is itself in L. This, however, contradicts to the fact that b_1 was a shortest non-zero vector in L. Thus our claim holds.

The claim verified in the previous paragraph shows that the theorem is valid when $n = 1$. Let us, hence, assume that $n > 1$. We may write an element of $\mathbb{R}^n$ as the sum of two vectors, one of them is parallel to b_1 and the other one is orthogonal to b_1:

$$v = v^* + \frac{(v, b_1)}{(b_1, b_1)} b_1 \ .$$

Simple computation shows that $(v^*, b_1) = 0$, and the map $v \mapsto v^*$ is linear. Let $L^* = \{v^* | v \in L\}$. We show that L^* is a lattice in the subspace, or hyperplane, $H \cong \mathbb{R}^{n-1}$ formed by the vectors orthogonal to b_1. The map $v \mapsto v^*$ is linear, and so L^* is closed under addition and subtraction. In order to show that it is discrete, let K be a bounded region in H. We are required to show that only finitely many points of L^* are in K. Let $v \in L$ be a vector such that $v^* \in K$. Let λ be the integer that is closest to the number $(v, b_1)/(b_1, b_1)$ and let $v' = v - \lambda b_1$. Clearly, $v' \in L$ and $v'^* = v^*$. Further, we also have that $|(v', b_1)/(b_1, b_1)| = |(v - \lambda b_1, b_1)/(b_1, b_1)| \leq 1/2$, and so the vector v' lies in the bounded region $K \times \{\mu b_1 : \ -1/2 \leq \mu \leq 1/2\}$. However, there are only finitely many vectors $v' \in L$ in this latter region, and so K also has only finitely many lattice vectors $v^* = v'^* \in L^*$.

We have, thus, shown that L^* is a lattice in H, and, by the induction hypothesis, it has a basis. Let $b_2, \ldots, b_r \in L$ be lattice vectors such that the vectors $b_2^*, \ldots, b_r^*$ form a basis of the lattice L^*. Then, for an arbitrary lattice vector $v \in L$, the vector v^* can be written in the form $\sum_{i=2}^r \lambda_i b_i^*$ where the coefficients λ_i are integers. Then $v' = v - \sum_{i=2}^r \lambda_i b_i \in L$ and, as the map $v \mapsto v^*$ is linear, we have $v'^* = 0$. This, however, implies that v' is a lattice vector on the line λb_1, and so $v' = \lambda_1 b_1$ with some integer λ_1. Therefore $v = \sum_{i=1}^r \lambda_i b_i$, that is, v is an integer linear combination of the vectors $b_1, \ldots, b_r$. Thus the vectors $b_1, \ldots, b_r$ form a basis of L. ■

A lattice L is always full in the linear subspace spanned by L. Thus, without loss of generality, we will consider only full lattices, and, in the sequel, by a lattice we will always mean a *full lattice* .

Example 1.4 Two familiar lattices in $\mathbb{R}^2$:
1. The ***square lattice*** is the lattice in $\mathbb{R}^2$ with basis $b_1 = (1, 0), \ b_2 = (0, 1)$.
2. The ***triangular lattice*** is the lattice with basis $b_1 = (1, 0)$, $b_2 = (1/2, (\sqrt{3})/2)$.

The following simple fact will often be used.

Lemma 1.49 *Let L be a lattice in $\mathbb{R}^n$, and let $b_1, \ldots, b_n$ be a basis of L. If we reorder the basis vectors $b_1, \ldots, b_n$, or if we add to a basis vector an integer linear combination of the other basis vectors, then the collection so obtained will also form a basis of L.*

Proof. Straightforward. ■

Let $b_1, \ldots, b_n$ be a basis in L. The Gram matrix of $b_1, \ldots, b_n$ is the matrix $B = (B_{ij})$ with entries $B_{ij} = (b_i, b_j)$. The matrix B is positive definite, since it is of the form $A^T A$ where A is a full-rank matrix (see Theorem 31.6). Consequently, $\det B$ is a positive real number. Reference to NA!

Lemma 1.50 *Let $b_1, \ldots, b_n$ and $w_1, \ldots, w_n$ be bases of a lattice L and let B and W be the matrices $B_{ij} = (b_i, b_j)$ and $W_{ij} = (w_i, w_j)$. Then the determinants of B and W coincide.*

Proof. For all $i = 1, \ldots, n$, the vector w_i is of the form $w_i = \sum_{j=1}^n \alpha_{ij} b_j$ where the α_{ij} are

integers. Let A be the matrix with entries $A_{ij} = \alpha_{ij}$. Then, as

$$(w_i, w_j) \quad = \quad (\sum_{k=1}^{n} \alpha_{ik} b_k, \sum_{l=1}^{n} \alpha_{jl} b_l) = \sum_{k=1}^{n} \alpha_{ik} \sum_{l=1}^{n} (b_k, b_l) \alpha_{jl},$$

we have $W = ABA^T$, and so $\det W = \det B (\det A)^2$. The number $\det W / \det B = (\det A)^2$ is a non-negative integer, since the entries of A are integers. Swapping the two bases, the same argument shows that $\det B / \det W$ is also a non-negative integer. This can only happen if $\det B = \det W$. ∎

Definition 1.51 (The determinant of a lattice). *The determinant of a lattice L is* $\det L = \sqrt{\det B}$ *where B is the Gram matrix of a basis of L.*

By the previous lemma, $\det L$ is independent of the choice of the basis. The quantity $\det L$ has a geometric meaning, as $\det L$ is the volume of the solid body, the so-called parallelepiped, formed by the vectors $\{\sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n \le 1\}$.

Remark 1.52 *Assume that the coordinates of the vectors b_i in an orthonormal basis of $\mathbb{R}^n$ are $\alpha_{i1}, \ldots, \alpha_{in}$ $(i = 1, \ldots, n)$. Then the Gram matrix B of the vectors $b_1, \ldots, b_n$ is $B = AA^T$ where A is the matrix $A_{ij} = \alpha_{ij}$. Consequently, if $b_1, \ldots, b_n$ is a basis of a lattice L, then* $\det L = |\det A|$.

Proof. The assertion follows from the equations $(b_i, b_j) = \sum_{k=1}^{n} \alpha_{ik} \alpha_{jk}$. ∎

1.4.2. Short lattice vectors

We will need a fundamental result in convex geometry. In order to prepare for this, we introduce some simple notation. Let $H \subseteq \mathbb{R}^n$. The set H is said to be ***centrally symmetric***, if $v \in H$ implies $-v \in H$. The set H is ***convex,*** if $u, v \in H$ implies $\lambda u + (1 - \lambda) v \in H$ for all $0 \le \lambda \le 1$.

Theorem 1.53 (Minkowski's Convex Body Theorem). *Let L be a lattice in $\mathbb{R}^n$ and let $K \subseteq \mathbb{R}^n$ be a centrally symmetric, bounded, closed, convex set. Suppose that the volume of K is at least $2^n \det L$. Then $K \cap L \neq \{0\}$.*

Proof. By the conditions, the volume of the set $(1/2)K := \{(1/2)v : \ v \in K\}$ is at least $\det L$. Let $b_1, \ldots, b_n$ be a basis of the lattice L and let $P = \{\sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n < 1\}$ be the corresponding half-open parallelepiped. Then each of the vectors in $\mathbb{R}^n$ can be written uniquely in the form $x + z$ where $x \in L$ and $z \in P$. For an arbitrary lattice vector $x \in L$, we let

$$K_x = (1/2)K \cap (x + P) = (1/2)K \cap \{x + z : \ z \in P\} \ .$$

As the sets $(1/2)K$ and P are bounded, so is the set

$$(1/2)K - P = \{u - v : \ u \in (1/2) \cdot K, \ v \in P\} .$$

As L is discrete, L only has finitely many points in $(1/2)K - P$; that is, $K_x = \emptyset$, except for finitely many $x \in L$. Hence $S = \{x \in L : \ K_x \neq \emptyset\}$ is a finite set, and, moreover, the set $(1/2)K$ is the disjoint union of the sets K_x $(x \in S)$. Therefore, the total volume of these sets

is at least $\det L$. For a given $x \in S$, we set $P_x = K_x - x = \{z \in P : \ x + z \in (1/2)K\}$. Consider the closure $\overline{P}$ and $\overline{P_x}$ of the sets P and P_x, respectively:

$$\overline{P} = \left\{ \sum_{i=1}^{n} \alpha_i b_i : \ 0 \le \alpha_1, \ldots, \alpha_n \le 1 \right\}$$

and $\overline{P_x} = \left\{ z \in \overline{P} : \ x + z \in (1/2)K \right\}$. The total volume of the closed sets $\overline{P_x} \subseteq \overline{P}$ is at least as large as the volume of the set $\overline{P}$, and so these sets cannot be disjoint: there are $x \neq y \in S$ and $z \in \overline{P}$ such that $z \in \overline{P_x} \cap \overline{P_y}$, that is, $x + z \in (1/2)K$ and $y + z \in (1/2)K$. As $(1/2) \cdot K$ is centrally symmetric, we find that $-y - z \in (1/2) \cdot K$. As $(1/2)K$ is convex, we also have $(x-y)/2 = ((x+z)+(-y-z))/2 \in (1/2)K$. Hence $x-y \in K$. On the other hand, the difference $x - y$ of two lattice points lies in $L \setminus \{0\}$. ■

Minkowski's theorem is sharp. For, let $\epsilon > 0$ be an arbitrary positive number, and let $L = \mathbf{Z}^n$ be the lattice of points with integer coordinates in $\mathbb{R}^n$. Let K be the set of vectors $(v_1, \ldots, v_n) \in \mathbb{R}^n$ for which $-1 + \epsilon \le v_i \le 1 - \epsilon$ $(i = 1, \ldots, n)$. Then K is bounded, closed, convex, centrally symmetric with respect to the origin, its volume is $(1 - \epsilon)^n 2^n \det L$, yet $L \cap K = \{0\}$.

Corollary 1.54 *Let L be a lattice in $\mathbb{R}^n$. Then L has a lattice vector $v \neq 0$ whose length is at most $\sqrt{n}\sqrt[n]{\det L}$.*

Proof. Let K be the following centrally symmetric cube with side length $s = 2\sqrt[n]{\det L}$:

$$K = \{(v_1, \ldots, v_n) \in \mathbb{R}^n : \ -s/2 \le v_i \le s/2, \ i = 1, \ldots, n\} .$$

The volume of the cube K is exactly $2^n \det L$, and so it contains a non-zero lattice vector. However, the vectors in K have length at most $\sqrt{n}\sqrt[n]{\det L}$. ■

We remark that, for $n > 1$, we can find an even shorter lattice vector, if we replace the cube in the proof of the previous assertion by a suitable ball.

1.4.3. Gauss' algorithm for two-dimensional lattices

Our goal is to design an algorithm that finds a non-zero short vector in a given lattice. In this section we consider this problem for two-dimensional lattices, which is the simplest non-trivial case. Then there is an elegant, instructive, and efficient algorithm that finds short lattice vectors. This algorithm also serves as a basis for the higher-dimensional cases. Let L be a lattice with basis b_1, b_2 in $\mathbb{R}^2$.

GAUSS(b_1, b_2)

1 $(a, b) \leftarrow (b_1, b_2)$
2 **forever**
3 **do** $b \leftarrow$ the shortest lattice vector on the line $b - \lambda a$
4 **if** $|b| < |a|$
5 **then** $b \leftrightarrow a$
6 **else return** (a, b)

In order to analyse the procedure, the following facts will be useful.

Lemma 1.55 *Suppose that a and b are two linearly independent vectors in the plane $\mathbb{R}^2$, and let L be the lattice generated by them. The vector b is a shortest non-zero vector of L on the line $b - \lambda a$ if and only if*

$$|(b,a)/(a,a)| \le 1/2 \ . \tag{1.10}$$

Proof. We write b as the sum of a vector parallel to a and a vector orthogonal to a:

$$b = (b,a)/(a,a)a + b^* \ . \tag{1.11}$$

Then, as the vectors a and b^* are orthogonal,

$$|b - \lambda a|^2 = \left|\left(\frac{(b,a)}{(a,a)} - \lambda\right)a + b^*\right|^2 = \left(\frac{(b,a)}{(a,a)} - \lambda\right)^2 |a|^2 + |b^*|^2 \ .$$

This quantity takes its smallest value for the integer λ that is the closest to the number $(b,a)/(a,a)$. Hence $\lambda = 0$ gives the minimal value if and only if (1.10) holds. ∎

Lemma 1.56 *Suppose that the linearly independent vectors a and b form a basis for a lattice $L \subseteq \mathbb{R}^2$ and that inequality (1.10) holds. Assume, further, that*

$$|b|^2 \ge (3/4)|a|^2 \ . \tag{1.12}$$

Write b, as in (1.11), as the sum of the vector $((b,a)/(a,a))a$, which is parallel to a, and the vector $b^* = b - ((b,a)/(a,a))a$, which is orthogonal to a. Then

$$|b^*|^2 \ge (1/2)|a|^2 \ . \tag{1.13}$$

Further, either b or a is a shortest non-zero vector in L.

Proof. By the assumptions,

$$|a|^2 \le \frac{4}{3}|b|^2 = \frac{4}{3}|b^*|^2 + \frac{4}{3}\left((b,a)/(a,a)\right)^2 |a|^2 \le \frac{4}{3}|b^*|^2 + (1/3)|a|^2 \ .$$

Rearranging the last displayed line, we obtain $|b^*|^2 \ge (1/2)|a|^2$.

The length of a vector $0 \ne v = \alpha a + \beta b \in L$ can be computed as

$$|\alpha a + \beta b|^2 = |\beta b^*|^2 + (\alpha + \beta(b,a)/(a,a))^2 |a|^2 \ge \beta^2 |b^*|^2 \ge (1/2)\beta^2 |a|^2,$$

which implies $|v| > |a|$ whenever $|\beta| \ge 2$. If $\beta = 0$ and $\alpha \ne 0$, then $|v| = |\alpha| \cdot |a| \ge |a|$. Similarly, $\alpha = 0$ and $\beta \ne 0$ gives $|v| = |\beta| \cdot |b| \ge |b|$. It remains to consider the case when $\alpha \ne 0$ and $\beta = \pm 1$. As $|-v| = |v|$, we may assume that $\beta = 1$. In this case, however, v is of the form $v = b - \lambda a$ ($\lambda = -\alpha$), and, by Lemma 1.55, the vector b is a shortest lattice vector on this line. ∎

Theorem 1.57 *Let v be a shortest non-zero lattice vector in L. Then Gauss' algorithm terminates after $O(1 + \lg(|b_1|/|v|))$ iterations, and the resulting vector a is a shortest non-zero vector in L.*

Proof. First we verify that, during the course of the algorithm, the vectors a and b will always form a basis for the lattice L. If, in line 3, we replace b by a vector of the form $b' = b - \lambda a$, then, as $b = b' + \lambda a$, the pair $a,\ b'$ remains a basis of L. The swap in line 5 only concerns the order of the basis vectors. Thus a and b is always a basis of L, as we claimed.

By Lemma 1.55, inequality (1.10) holds after the first step (line 3) in the loop, and so we may apply Lemma 1.56 to the scenario before lines 4–5. This shows that if none of a and b is shortest, then $|b|^2 \leq (3/4)|a|^2$. Thus, except perhaps for the last execution of the loop, after each swap in line 5, the length of a is decreased by a factor of at least $\sqrt{3/4}$. Thus we obtain the bound for the number of executions of the loop. Lemma 1.56 implies also that the vector a at the end is a shortest non-zero vector in L. ∎

Gauss' algorithm gives an efficient polynomial time method for computing a shortest vector in the lattice $L \subseteq \mathbb{R}^2$. The analysis of the algorithm gives the following interesting theoretical consequence.

Corollary 1.58 *Let L be a lattice in $\mathbb{R}^2$, and let a be a shortest non-zero lattice vector in L. Then $|a|^2 \leq (2/\sqrt{3})\det L$.*

Proof. Let b be a vector in L such that b is linearly independent of a and (1.10) holds. Then

$$|a|^2 \leq |b|^2 = |b^*|^2 + \left(\frac{(b,a)}{(a,a)}\right)^2 |a|^2 \leq |b^*|^2 + \frac{1}{4}|a|^2 \ ,$$

which yields $(3/4)|a|^2 \leq |b^*|^2$. The area of the fundamental parallelogram can be computed using the well-known formula

$$\text{area} = \text{base} \cdot \text{height},$$

and so $\det L = |a||b^*|$. The number $|b^*|$ can now be bounded by the previous inequality. ∎

1.4.4. A Gram-Schmidt orthogonalisation and weak reduction

Let $b_1, \ldots, b_n$ be a linearly independent collection of vectors in $\mathbb{R}^n$. For an index i with $i \in \{1, \ldots, n\}$, we let b_i^* denote the component of b_i that is orthogonal to the subspace spanned by $b_1, \ldots, b_{i-1}$. That is,

$$b_i = b_i^* + \sum_{j=1}^{i-1} \lambda_{ij} b_j \ ,$$

where

$$(b_i^*, b_j) = 0 \quad \text{for} \quad j = 1, \ldots, i-1 \ .$$

Clearly $b_1^* = b_1$. The vectors $b_1^*, \ldots, b_{i-1}^*$ span the same subspace as the vectors $b_1, \ldots, b_{i-1}$, and so, with suitable coefficients μ_{ij}, we may write

$$b_i = b_i^* + \sum_{j=1}^{i-1} \mu_{ij} b_j^* \ , \tag{1.14}$$

and

$$(b_i^*, b_j^*) = 0, \quad \text{if } j \neq i \ .$$

By the latter equations, the vectors $b_1^*, \ldots, b_{i-1}^*, b_i^*$ form an orthogonal system, and so

$$\mu_{ij} = \frac{(b_i, b_j^*)}{(b_j^*, b_j^*)} \quad (j = 1, \ldots, i-1) \,. \tag{1.15}$$

The set of the vectors $b_1^*, \ldots, b_n^*$ is said to be the ***Gram-Schmidt orthogonalisation*** of the vectors $b_1, \ldots, b_n$.

Lemma 1.59 *Let $L \subseteq \mathbb{R}^n$ be a lattice with basis $b_1, \ldots, b_n$. Then*

$$\det L = \prod_{i=1}^{n} |b_i^*| \,.$$

Proof. Set $\mu_{ii} = 1$ and $\mu_{ij} = 0$, if $j > i$. Then $b_i^* = \sum_{k=1}^{n} \mu_{ik} b_k$, and so

$$(b_i^*, b_j^*) = \sum_{k=1}^{n} \mu_{ik} \sum_{l=1}^{n} (b_k, b_l) \mu_{jl},$$

that is, $B^* = MBM^T$ where B and B^* are the Gram matrices of the collections $b_1, \ldots, b_n$ and $b_1^*, \ldots, b_n^*$, respectively, and M is the matrix with entries μ_{ij}. The matrix M is a lower triangular matrix with ones in the main diagonal, and so $\det M = \det M^T = 1$. As B^* is a diagonal matrix, we obtain $\prod_{i=1}^{n} |b_i^*|^2 = \det B^* = (\det M)(\det B)(\det M^T) = \det B$. ■

Corollary 1.60 (Hadamard inequality). $\prod_{i=1}^{n} |b_i| \geq \det L$.

Proof. The vector b_i can be written as the sum of the vector b_i^* and a vector orthogonal to b_i^*, and hence $|b_i^*| \leq |b_i|$. ■

The vector b_i^* is the component of b_i orthogonal to the subspace spanned by the vectors $b_1, \ldots, b_{i-1}$. Thus b_i^* does not change if we subtract a linear combination of the vectors $b_1, \ldots, b_{i-1}$ from b_i. If, in this linear combination, the coefficients are integers, then the new sequence $b_1, \ldots, b_n$ will be a basis of the same lattice as the original. Similarly to the first step of the loop in Gauss' algorithm, we can make the numbers μ_{ij} in (1.15) small. The input of the following procedure is a basis $b_1, \ldots, b_n$ of a lattice L.

WEAK-REDUCTION$(b_1, \ldots, b_n)$

1 **for** $j \leftarrow n-1$ **downto** 1
2 **do for** $i \leftarrow j+1$ **to** n
3 $b_i \leftarrow b_i - \lambda b_j$, where λ is the integer nearest the number $(b_i, b_j^*)/(b_j^*, b_j^*)$
4 **return** $(b_1, \ldots, b_n)$

Definition 1.61 (Weakly reduced basis). *A basis $b_1, \ldots, b_n$ of a lattice is said to be* ***weakly reduced*** *if the coefficients μ_{ij} in (1.15) satisfy*

$$|\mu_{ij}| \leq \frac{1}{2} \quad \textit{for} \quad 1 \leq j < i \leq n \,.$$

Lemma 1.62 *The basis given by the procedure* WEAK-REDUCTION *is weakly reduced.*

Proof. By the remark preceding the algorithm, we obtain that the vectors $b_1^*, \ldots, b_n^*$ never change. Indeed, we only subtract linear combinations of vectors with index less than i from b_i. Hence the inner instruction does not change the value of (b_k, b_l^*) with $k \neq i$. The values of the (b_i, b_l^*) do not change for $l > j$ either. On the other hand, the instruction achieves, with the new b_i, that the inequality $|\mu_{ij}| \leq 1/2$ holds:

$$|(b_i - \lambda b_j^*, b_j^*)| = |(b_i, b_j^*) - \lambda(b_j^*, b_j^*)| = |(b_i, b_j^*) - \lambda(b_j^*, b_j^*)| \leq \frac{1}{2}(b_j^*, b_j^*) \ .$$

By the observations above, this inequality remains valid during the execution of the procedure. ■

1.4.5. Lovász-reduction

First we define, in an arbitrary dimension, a property of the bases that usually turns out to be useful. The definition will be of a technical nature. Later we will see that these bases are interesting, in the sense that they consist of short vectors. This property will make them widely applicable.

Definition 1.63 *A basis $b_1, \ldots, b_n$ of a lattice L is said to be **(Lovász-)reduced** if*

- *it is weakly reduced,*

and, using the notation introduced for the Gram-Schmidt orthogonalisation,

- $|b_i^*|^2 \leq (/3)|b_{i+1}^* + \mu_{i+1,i} b_i^*|^2$ *for all* $1 \leq i < n$.

Let us observe the analogy of the conditions above to the inequalities that we have seen when investigating Gauss' algorithm. For $i = 1$, $a = b_1$ and $b = b_2$, being weakly reduced ensures that b is a shortest vector on the line $b - \lambda a$. The second condition is equivalent to the inequality $|b|^2 \geq (3/4)|a|^2$, but here it is expressed in terms of the Gram-Schmidt basis. For a general index i, the same is true, if a plays the rôle of the vector b_i, and b plays the rôle of the component of the vector b_{i+1} that is orthogonal to the subspace spanned by $b_1, \ldots, b_{i-1}$.

LOVÁSZ-REDUCTION$(b_1, \ldots, b_n)$

```
1  forever
2        do (b_1, ..., b_n) ←WEAK-REDUCTION(b_1, ..., b_n)
3        find an index i for which the second condition of being reduced is violated
4        if there is such an i
5           then b_i ↔ b_{i+1}
6           else return (b_1, ..., b_n)
```

Theorem 1.64 *Suppose that in the lattice $L \subseteq \mathbb{R}^n$ each of the pairs of the lattice vectors has an integer scalar product. Then the swap in the 5th line of the* LOVÁSZ-REDUCTION *occurs at most $\lg_{4/3}(B_1 \cdots B_{n-1})$ times where B_i is the upper left $(i \times i)$-subdeterminant of the Gram matrix of the initial basis $b_1, \ldots, b_n$.*

Proof. The determinant B_i is the determinant of the Gram matrix of $b_1, \ldots, b_i$, and, by the observations we made at the discussion of the Gram-Schmidt orthogonalisation,

$B_i = \prod_{j=1}^{i} |b_j^*|^2$. This, of course, implies that $B_i = B_{i-1}|b_i^*|^2$ for $i > 1$. By the above, the procedure Weak-Reduction cannot change the vectors b_i^*, and so it does not change the product $\prod_{j=1}^{n-1} B_j$ either. Assume, in line 5 of the procedure, that a swap $b_i \leftrightarrow b_{i+1}$ takes place. Observe that, unless $j = i$, the sets $\{b_1, \ldots, b_j\}$ do not change, and neither do the determinants B_j. The rôle of the vector b_i^* is taken over by the vector $b_{i+1}^* + \mu_{i,i+1} b_i$, whose length, because of the conditions of the swap, is at most $\sqrt{3/4}$ times the length of b_i^*. That is, the new B_i is at most 3/4 times the old. By the observation above, the new value of $B = \prod_{j=1}^{n-1} B_j$ will also be at most 3/4 times the old one. Then the assertion follows from the fact that the quantity B remains a positive integer. ∎

Corollary 1.65 *Under the conditions of the previous theorem, the cost of the procedure* Lovász-Reduction *is at most* $O(n^5 \lg nC)$ *arithmetic operations with rational numbers where* C *is the maximum of* 2 *and the quantities* $|(b_i, b_j)|$ *with* $i, j = 1, \ldots, n$.

Proof. It follows from the Hadamard inequality that

$$B_i \le \prod_{j=1}^{i} \sqrt{(b_1, b_j)^2 + \ldots + (b_i, b_j)^2} \le (\sqrt{i}C)^i \le (\sqrt{n}C)^n \ .$$

Hence $B_1 \cdots B_{n-1} \le (\sqrt{n}C)^{n(n-1)}$ and $\lg_{4/3}(B_1 \ldots B_{n-1}) = O(n^2 \lg nC)$. By the previous theorem, this is the number of iterations in the algorithm. The cost of the Gram–Schmidt orthogonalisation is $O(n^3)$ operations, and the cost of weak reduction is $O(n^2)$ scalar product computations, each of which can be performed using $O(n)$ operations (provided the vectors are represented by their coordinates in an orthogonal basis). ∎

One can show that the length of the integers that occur during the run of the algorithm (including the numerators and the denominators of the fractions in the Gram–Schmidt orthogonalisation) will be below a polynomial bound.

1.4.6. Properties of reduced bases

Theorem 1.67 of this section gives a summary of the properties of reduced bases that turn out to be useful in their applications. We will find that a reduced basis consists of relatively short vectors. More precisely, $|b_1|$ will approximate, within a constant factor depending only on the dimension, the length of a shortest non-zero lattice vector.

Lemma 1.66 *Let us assume that the vectors* $b_1, \ldots, b_n$ *form a reduced basis of a lattice* L. *Then, for* $1 \le j \le i \le n$,

$$(b_i^*, b_i^*) \ge 2^{j-i} (b_j^*, b_j^*) \ . \tag{1.16}$$

In particular,

$$(b_i^*, b_i^*) \ge 2^{1-i} (b_1^*, b_1^*) \ . \tag{1.17}$$

Proof. Substituting $a = b_i^*$, $b = b_{i+1}^* + ((b_{i+1}, b_i^*))/((b_i^*, b_i^*) b_i^*)$, Lemma 1.56 gives, for all $1 \le i < n$, that

$$(b_{i+1}^*, b_{i+1}^*) \ge (1/2)(b_i^*, b_i^*) \ .$$

Thus, inequality (1.16) follows by induction. ∎

Now we can formulate the fundamental theorem of reduced bases.

Theorem 1.67 *Assume that the vectors $b_1, \ldots, b_n$ form a reduced basis of a lattice L. Then*

(i) $|b_1| \le 2^{(n-1)/4} (\det L)^{(1/n)}$.

(ii) $|b_1| \le 2^{(n-1)/2} |b|$ *for all lattice vectors* $0 \ne b \in L$. *In particular, the length of* b_1 *is not greater than* $2^{(n-1)/2}$ *times the length of a shortest non-zero lattice vector.*

(iii) $|b_1| \cdots |b_n| \le 2^{(n(n-1))/4} \det L$.

Proof. (i) Using inequality (1.17),

$$(\det L)^2 = \prod_{i=1}^{n} (b_i^*, b_i^*) \ge \prod_{i=1}^{n} (2^{1-i}(b_1, b_1)) = 2^{\frac{-n(n-1)}{2}} (b_1, b_1)^n \ ,$$

and so assertion (i) holds.

(ii) Let $b = \sum_{i=1}^{n} z_i b_i \in L$ with $z_i \in \mathbf{Z}$ be a lattice vector. Assume that z_j is the last non-zero coefficient and write $b_j = b_j^* + v$ where v is a linear combination of the vectors $b_1, \ldots, b_{j-1}$. Hence $b = z_j b_j^* + w$ where w lies in the subspace spanned by $b_1, \ldots, b_{j-1}$. As b_j^* is orthogonal to this subspace,

$$(b, b) = z_j^2 (b_j^*, b_j^*) + (w, w) \ge (b_j^*, b_j^*) \ge 2^{1-j} (b_1, b_1) \ge 2^{1-n} (b_1, b_1) \ ,$$

and so assertion (ii) is valid.

(iii) First we show that $(b_i, b_i) \le 2^{i-1} (b_i^*, b_i^*)$. This inequality is obvious if $i = 1$, and so we assume that $i > 1$. Using the decomposition (1.14) of the vector b_i and the fact that the basis is weakly reduced, we obtain that

$$\begin{aligned} (b_i, b_i) &= \sum_{j=1}^{i} \left(\frac{(b_i, b_j^*)}{(b_j^*, b_j^*)} \right)^2 (b_j^*, b_j^*) \le (b_i^*, b_i^*) + \frac{1}{4} \sum_{j=1}^{i-1} (b_j^*, b_j^*) \le (b_i^*, b_i^*) + \frac{1}{4} \sum_{j=1}^{i-1} 2^{i-j} (b_i^*, b_i^*) \\ &\le (2^{i-2} + 1)(b_i^*, b_i^*) \le 2^{i-1} (b_i^*, b_i^*) \ . \end{aligned}$$

Multiplying these inequalities for $i = 1, \ldots, n$,

$$\prod_{i=1}^{n} (b_i, b_i) \le \prod_{i=1}^{n} 2^{i-1} (b_i^*, b_i^*) = 2^{\frac{n(n-1)}{2}} \prod_{i=1}^{n} (b_i^*, b_i^*) = 2^{\frac{n(n-1)}{2}} (\det L)^2 \ ,$$

which is precisely the inequality in (iii). ∎

It is interesting to compare assertion (i) in the previous theorem and Corollary 1.54 after Minkowski's theorem. Here we obtain a weaker bound for the length of b_1, but this vector can be obtained by an efficient algorithm. Essentially, the existence of the basis that satisfies assertion (iii) was first shown by Hermite using the tools in the proofs of Theorems 1.48 and 1.67. Using a Lovász-reduced basis, the cost of finding a shortest vector in a lattice with dimension n is at most polynomial in the input size and in 3^{n^2}; see Exercise 1.4-4..

Exercises

1.4-1 *The triangular lattice is optimal.* Show that the bound in Corollary 1.58 is sharp. More precisely, let $L \subseteq \mathbb{R}^2$ be a full lattice and let $0 \ne a \in L$ be a shortest vector in L. Verify

that the inequality $|a|^2 = (2/\sqrt{3})\det L$ holds if and only if L is similar to the triangular lattice.

1.4-2 *The denominators of the Gram-Schmidt numbers.* Let us assume that the Gram matrix of a basis $b_1, \dots, b_n$ has only integer entries. Show that the numbers μ_{ij} in (1.15) can be written in the form $\mu_{ij} = \zeta_{ij} / \prod_{k=1}^{j-1} B_k$ where the ζ_{ij} are integers and B_k is the determinant of the Gram matrix of the vectors $b_1, \dots, b_k$.

1.4-3 *The length of the vectors in a reduced basis.* Let $b_1, \dots, b_n$ be a reduced basis of a lattice L and let us assume that the numbers (b_i, b_i) are integers. Give an upper bound depending only on n and $\det L$ for the length of the vectors b_i. More precisely, prove that

$$|b_i| \le 2^{\frac{n(n-1)}{4}} \det L \,.$$

1.4-4 *The coordinates of a shortest lattice vector.* Let $b_1, \dots, b_n$ be a reduced basis of a lattice L. Show that each of the shortest vectors in L is of the form $\sum z_i b_i$ where $z_i \in \mathbf{Z}$ and $|z_i| \le 3^n$. Consequently, for a bounded n, one can find a shortest non-zero lattice vector in polynomial time.

Hint: Assume, for some lattice vector $v = \sum z_i b_i$, that $|v| \le |b_1|$. Let us write v in the basis $b_1^*, \dots, b_n^*$:

$$v = \sum_{j=1}^{n} (z_j + \sum_{i=j+1}^{n} \mu_{ij} z_i) b_j^* \,.$$

It follows from the assumption that each of the components of v (in the orthogonal basis) is at most as long as $b_1 = b_1^*$:

$$\left| z_j + \sum_{i=j+1}^{n} \mu_{ij} z_i \right| \le \frac{|b_1^*|}{|b_j^*|} \,.$$

Use then the inequalities $|\mu_{ij}| \le 1/2$ and (1.17).

1.5. Factoring polynomials in $\mathbb{Q}[x]$

In this section we study the problem of factoring polynomials with rational coefficients. The input of the ***factorisation problem*** is a polynomial $f(x) \in \mathbb{Q}[x]$. Our goal is to compute a factorisation

$$f = f_1^{e_1} f_2^{e_2} \cdots f_s^{e_s}, \tag{1.18}$$

where the polynomials $f_1, \dots, f_s$ are pairwise relatively prime, and irreducible over $\mathbb{Q}$, and the numbers e_i are positive integers. By Theorem 1.4, f determines, essentially uniquely, the polynomials f_i and the exponents e_i.

1.5.1. Preparations

First we reduce the problem (1.18) to another problem that can be handled more easily.

Lemma 1.68 *We may assume that the polynomial $f(x)$ has integer coefficients and it has leading coefficient* 1.

Proof. Multiplying by the common denominator of the coefficients, we may assume that $f(x) = a_0 + a_1x + \cdots + a_nx^n \in \mathbf{Z}[x]$. Performing the substitution $y = a_nx$, we obtain the polynomial

$$g(y) = {a_n}^{n-1} f\left(\frac{y}{a_n}\right) = y^n + \sum_{i=0}^{n-1} a_n^{n-i-1} a_i y^i,$$

which has integer coefficients and its leading coefficient is 1. Using a factorisation of $g(y)$, a factorisation of $f(x)$ can be obtained efficiently. ■

Primitive polynomials, Gauss' lemma

Definition 1.69 *A polynomial $f(x) \in \mathbf{Z}[x]$ is said to be **primitive**, if the greatest common divisor of its coefficients is* 1.

A polynomial $f(x) \in \mathbf{Z}[x] \setminus \{0\}$ can be written in a unique way as the product of an integer and a primitive polynomial in $\mathbf{Z}[x]$. Indeed, if a is the greatest common divisor of the coefficients, then $f(x) = a(1/a)f(x)$. Clearly, $(1/a)f(x)$ is a primitive polynomial with integer coefficients.

Lemma 1.70 (Gauss' Lemma). *If $u(x),\ v(x) \in \mathbf{Z}[x]$ are primitive polynomials, then so is the product $u(x)v(x)$.*

Proof. We argue by contradiction and assume that p is a prime number that divides all the coefficients of uv. Set $u(x) = \sum_{i=0}^{n} u_i x^i$, $v(x) = \sum_{j=0}^{m} v_j x^j$ and let i_0 and j_0 be the smallest indices such that $p \nmid u_{i_0}$ and $p \nmid v_{j_0}$. Let $k_0 = i_0 + j_0$ and consider the coefficient of x^{k_0} in the product $u(x)v(x)$. This coefficient is

$$\sum_{i+j=k_0} u_i v_j = u_{i_0} v_{j_0} + \sum_{i=0}^{i_0-1} u_i v_{k_0-i} + \sum_{j=0}^{j_0-1} u_{k_0-j} v_j \ .$$

Both of the sums on the right-hand side of this equation are divisible by p, while $u_{i_0} v_{j_0}$ is not, and hence the coefficient of x^{k_0} in $u(x)v(x)$ cannot be divisible by p after all. This, however, is a contradiction. ■

Proposition 1.71 *Let us assume that $g(x),\ h(x) \in \mathbb{Q}[x]$ are polynomials with rational coefficients and leading coefficient* 1 *such that the product $g(x)h(x)$ has integer coefficients. Then the polynomials $g(x)$ and $h(x)$ have integer coefficients.*

Proof. Let us multiply $g(x)$ and $h(x)$ by the least common multiple c_g and c_h, respectively, of the denominators of their coefficients. Then the polynomials $c_g g(x)$ and $c_h h(x)$ are primitive polynomials with integer coefficients. Hence, by Gauss' Lemma, so is the product $c_g c_h g(x)h(x) = (c_g g(x))(c_h h(x))$. As the coefficients of $g(x)h(x)$ are integers, each of its coefficients is divisible by the integer $c_g c_h$. Hence $c_g c_h = 1$, and so $c_g = c_h = 1$. Therefore $g(x)$ and $h(x)$ are indeed polynomials with integer coefficients. ■

One can show similarly, for a polynomial $f(x) \in \mathbf{Z}[x]$, that factoring $f(x)$ in $\mathbf{Z}[x]$ is equivalent to factoring the primitive part of $f(x)$ in $\mathbb{Q}[x]$ and factoring an integer, namely the greatest common divisor of the coefficients

Mignotte's bound

As we work over an infinite field, we have to pay attention to the size of the results in our computations.

Definition 1.72 *The **norm** of a polynomial* $f(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{C}[x]$ *with complex coefficients is the real number* $\|f(x)\| = \sqrt{\sum_{i=0}^{n} |a_i|^2}$.

The inequality $\max_{i=0}^{n} |a_i| \leq \|f(x)\|$ implies that a polynomial $f(x)$ with integer coefficients can be described using $O(n \lg \|f(x)\|)$ bits.

Lemma 1.73 *Let* $f(x) \in \mathbb{C}[x]$ *be a polynomial with complex coefficients. Then, for all* $c \in \mathbb{C}$, *we have*

$$\|(x - c)f(x)\| = \|(\overline{c}x - 1)f(x)\| ,$$

where $\overline{c}$ *is the usual conjugate of the complex number* c.

Proof. Let us assume that $f(x) = \sum_{i=0}^{n} a_i x^i$ and set $a_{n+1} = a_{-1} = 0$. Then

$$(x - c)f(x) = \sum_{i=0}^{n+1} (a_{i-1} - ca_i)x^i ,$$

and hence

$$\begin{aligned} \|(x - c)f(x)\|^2 &= \sum_{i=0}^{n+1} |a_{i-1} - ca_i|^2 = \sum_{i=0}^{n+1} (|a_{i-1}|^2 + |ca_i|^2 - a_{i-1}\overline{ca_i} - \overline{a_{i-1}}ca_i) \\ &= \|f(x)\|^2 + |c|^2\|f(x)\|^2 - \sum_{i=0}^{n+1} (a_{i-1}\overline{ca_i} + \overline{a_{i-1}}ca_i) . \end{aligned}$$

Performing similar computations with the right-hand side of the equation in the lemma, we obtain that

$$(\overline{c}x - 1)f(x) = \sum_{i=0}^{n+1} (\overline{c}a_{i-1} - a_i)x^i ,$$

and so

$$\begin{aligned} \|(\overline{c}x - 1)f(x)\|^2 &= \sum_{i=0}^{n+1} |\overline{c}a_{i-1} - a_i|^2 = \sum_{i=0}^{n+1} (|\overline{c}a_{i-1}|^2 + |a_i|^2 - \overline{c}a_{i-1}\overline{a_i} - c\overline{a_{i-1}}a_i) \\ &= \|f(x)\|^2 + |c|^2\|f(x)\|^2 - \sum_{i=0}^{n+1} (a_{i-1}\overline{ca_i} + \overline{a_{i-1}}ca_i) . \end{aligned}$$

The proof of the lemma is now complete. ∎

Theorem 1.74 (Mignotte). *Let us assume that the polynomials* $f(x), g(x) \in \mathbb{C}[x]$ *have complex coefficients and leading coefficient* 1 *and that* $g(x)|f(x)$. *If* $\deg(g(x)) = m$, *then* $\|g(x)\| \leq 2^m \|f(x)\|$.

Proof. By the fundamental theorem of algebra, $f(x) = \prod_{i=1}^{n}(x - \alpha_i)$ where $\alpha_1, \ldots, \alpha_n$ are the complex roots of the polynomial $f(x)$ (with multiplicity). Then there is a subset $I \subseteq \{1, \ldots, n\}$ such that $g(x) = \prod_{i \in I}(x - \alpha_i)$. First we claim, for an arbitrary set $J \subseteq \{1, \ldots, n\}$, that

$$\prod_{i \in J} |\alpha_i| \leq \|f(x)\| \, . \tag{1.19}$$

If J contains an integer i with $\alpha_i = 0$, then this inequality will trivially hold. Let us hence assume that $\alpha_i \neq 0$ for every $i \in J$. Set $\overline{J} = \{1, \ldots, n\} \setminus J$ and $h(x) = \prod_{i \in \overline{J}}(x - \alpha_i)$. Applying Lemma 1.73 several times, we obtain that

$$\|f(x)\| = \|\prod_{i \in J}(x - \alpha_i)h(x)\| = \|\prod_{i \in J}(\overline{\alpha_i}x - 1)h(x)\| = |\prod_{i \in J}\overline{\alpha_i}| \cdot \|u(x)\| \, ,$$

where $u(x) = \prod_{i \in J}(x - 1/\overline{\alpha_i})h(x)$. As the leading coefficient of $u(x)$ is 1, $\|u(x)\| \geq 1$, and so

$$|\prod_{i \in J}\alpha_i| = |\prod_{i \in J}\overline{\alpha_i}| = \|f(x)\|/\|u(x)\| \leq \|f(x)\| \, .$$

Let us express the coefficients of $g(x)$ using its roots:

$$\begin{aligned} g(x) &= \prod_{i \in I}(x - \alpha_i) = \sum_{J \subseteq I}\left((-1)^{|J|}\prod_{j \in J}\alpha_j x^{m-|J|}\right) \\ &= \sum_{i=0}^{m}(-1)^{m-i}\left(\sum_{J \subseteq I, |J|=m-i}\prod_{j \in J}\alpha_j\right)x^i \, . \end{aligned}$$

For an arbitrary polynomial $t(x) = t_0 + \cdots + t_k x^k$, the inequality $\|t(x)\| \leq |t_0| + \cdots + |t_k|$ is valid. Therefore, using inequality (1.19), we find that

$$\begin{aligned} \|g(x)\| &\leq \sum_{i=0}^{m}\left|\sum_{J \subseteq I, |J|=m-i}\prod_{j \in J}\alpha_j\right| \\ &\leq \sum_{J \subseteq I}\left|\prod_{j \in J}\alpha_j\right| \leq 2^m\|f(x)\| \, . \end{aligned}$$

The proof is now complete. ∎

Corollary 1.75 *The bit size of the irreducible factors in $\mathbb{Q}[x]$ of an $f(x) \in \mathbf{Z}[x]$ with leading coefficient 1 is polynomial in the bit size of $f(x)$.*

Resultant and good reduction

Let $\mathbb{F}$ be an arbitrary field, and let $f(x),\ g(x) \in \mathbb{F}[x]$ be polynomials with degree n and m, respectively: $f = a_0 + a_1 x + \ldots + a_n x^n$, $g = b_0 + b_1 x + \ldots + b_m x^m$ where $a_n \neq 0 \neq b_m$. We recall the concept of the ***resultant*** from Chapter **??**. The resultant of f and g is the determinant of

the $((m+n)\times(m+n))$-matrix

$$M = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & \cdots & a_n & & & \\ & a_0 & a_1 & a_2 & \cdots & a_{n-1} & a_n & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_0 & a_1 & \cdots & a_{n-2} & a_{n-1} & a_n \\ b_0 & b_1 & \cdots & b_{m-1} & b_m & & & & \\ & b_0 & b_1 & \cdots & b_{m-1} & b_m & & & \\ & & b_0 & b_1 & \cdots & b_{m-1} & b_m & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & b_0 & b_1 & \cdots & b_{m-1} & b_m. \end{pmatrix}. \tag{1.20}$$

The matrix above is usually referred to as the Sylvester matrix. The blank spaces in the Sylvester matrix represent zero entries.

The resultant provides information about the common factors of f and g. One can use it to express, particularly elegantly, the fact that two polynomials are relatively prime:

$$\gcd(f(x), g(x)) = 1 \Leftrightarrow Res(f, g) \neq 0\ . \tag{1.21}$$

Corollary 1.76 *Let $f(x) = a_0 + a_1 x + \cdots + a_n x^n \in \mathbf{Z}[x]$ be a square-free (in $\mathbb{Q}[x]$), non-constant polynomial. Then $Res(f(x), f'(x))$ is an integer. Further, assume that p is a prime not dividing na_n. Then the polynomial $f(x) \pmod p$ is square-free in $\mathbb{F}_p[x]$ if and only if p does not divide $Res(f(x), f'(x))$.*

Proof. The entries of the Sylvester matrix corresponding to $f(x)$ and $f'(x)$ are integers, and so is its determinant. The polynomial f has no multiple roots over $\mathbb{Q}$, and so, by Exercise 1.5-1., $\gcd(f(x), f'(x)) = 1$, which gives, using (1.21), that $Res(f(x), f'(x)) \neq 0$. Let $F(x)$ denote the polynomial f reduced modulo p. Then it follows from our assumptions that $Res(F(x), F'(x))$ is precisely the residue of $Res(f(x), f'(x))$ modulo p. By Exercise 1.5-1., the polynomial $F(x)$ is square-free precisely when $\gcd(F(x), F'(x)) = 1$, which is equivalent to $Res(F(x), F'(x)) \neq 0$. This amounts to saying that p does not divide the integer $Res(f(x), f'(x))$. ∎

Corollary 1.77 *If $f(x) \in \mathbf{Z}[x]$ is a square-free polynomial with degree n, then there is a prime $p = O((n \lg n + 2n \lg \|f\|)^2)$ (that is, the absolute value of p is polynomial in the bit size of f) such that the polynomial $f(x) \pmod p$ is square-free in $\mathbb{F}_p[x]$.*

Proof. By the Prime Number Theorem (Theorem 33.37), for large enough K, the product of the primes in the interval $[1, K]$ is at least $2^{(0.9K/\ln K)}$.

Reference to NA!

Set $K = \big((n+1)\lg n + 2n \lg \|f\|\big)^2$. If K is large enough, then

$$p_1 \cdots p_l \geq 2^{(0.9K/\ln K)} > 2^{\sqrt{K}} \geq n^{n+1}\|f\|^{2n} \geq n^{n+1}\|f\|^{2n-1}|a_n| \tag{1.22}$$

where $p_1, \ldots, p_l$ are primes not larger than K, and a_n is the leading coefficient of f.

Let us suppose, for the primes $p_1, \ldots, p_l$, that $f(x) \pmod{p_i}$ is not square-free in $\mathbb{F}_{p_i}[x]$. Then the product $p_1 \cdots p_l$ divides $Res(f(x), f'(x)) \cdot na_n$, and so

$$p_1 \cdots p_l \leq |Res(f, f')| \cdot |na_n| \leq \|f\|^{n-1} \cdot \|f'\|^n \cdot |na_n| \leq n^{n+1}\|f\|^{2n-1}|a_n|\ .$$

(In the last two inequalities, we used the Hadamard inequality, and the fact that $\|f'(x)\| \leq n\|f(x)\|$.) This contradicts to inequality (1.22), which must be valid because of the choice of K. ■

We note that using the Prime Number Theorem more carefully, one can obtain a stronger bound for p.

Hensel lifting

We present a general procedure that can be used to obtain, given a factorisation modulo a prime p, a factorisation modulo p^N of a polynomial with integer coefficients.

Theorem 1.78 (Hensel's lemma). *Suppose that* $f(x),\ g(x),\ h(x) \in \mathbf{Z}[x]$ *are polynomials with leading coefficient* 1 *such that* $f(x) \equiv g(x)h(x) \pmod{p}$, *and, in addition,* $g(x) \pmod{p}$ *and* $h(x) \pmod{p}$ *are relatively prime in* $\mathbb{F}_p[x]$. *Then, for an arbitrary positive integer* t, *there are polynomials* $g_t(x),\ h_t(x) \in \mathbf{Z}[x]$ *such that*

- *both of the leading coefficients of* $g_t(x)$ *and* $h_t(x)$ *are equal to* 1,
- $g_t(x) \equiv g(x) \pmod{p}$ *and* $h_t(x) \equiv h(x) \pmod{p}$,
- $f(x) \equiv g_t(x)h_t(x) \pmod{p^t}$.

Moreover, the polynomials $g_t(x)$ *and* $h_t(x)$ *satisfying the conditions above are unique modulo* p^t.

Proof. From the conditions concerning the leading coefficients, we obtain that $\deg f(x) = \deg g(x) + \deg h(x)$, and, further, that $\deg g_t(x) = \deg g(x)$ and $\deg h_t(x) = \deg h(x)$, provided the suitable polynomials $g_t(x)$ and $h_t(x)$ indeed exist. The existence is proved by induction on t. In the initial step, $t = 1$ and the choice $g_1(x) = g(x)$ and $h_1(x) = h(x)$ is as required.

The induction step $t \to t+1$: let us assume that there exist polynomials $g_t(x)$ and $h_t(x)$ that are well-defined modulo p^t and satisfy the conditions. If the polynomials $g_{t+1}(x)$ and $h_{t+1}(x)$ exist, then they must satisfy the conditions imposed on $g_t(x)$ and $h_t(x)$. As $g_t(x)$ and $h_t(x)$ are unique modulo p^t, we may write $g_{t+1}(x) = g_t(x) + p^t\delta_g(x)$ and $h_{t+1}(x) = h_t(x) + p^t\delta_h(x)$ where $\delta_g(x)$ and $\delta_h(x)$ are polynomials with integer coefficients. The condition concerning the leading coefficients guarantees that $\deg \delta_g(x) < \deg g(x)$ and that $\deg \delta_h(x) < \deg h(x)$.

By the induction hypothesis, $f(x) = g_t(x)h_t(x) + p^t\lambda(x)$ where $\lambda(x) \in \mathbf{Z}[x]$. The observations about the degrees of the polynomials $g_t(x)$ and $h_t(x)$ imply that the degree of $\lambda(x)$ is smaller than $\deg f(x)$. Now we may compute that

$$\begin{aligned} g_{t+1}(x)h_{t+1}(x) - f(x) &= g_t(x)h_t(x) - f(x) + p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) + p^{2t}\delta_g(x)\delta_h(x) \\ &\equiv -p^t\lambda(x) + p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) \pmod{p^{2t}}. \end{aligned}$$

As $2t > t+1$, the congruence above holds modulo p^{t+1}. Thus $g_{t+1}(x)$ and $h_{t+1}(x)$ satisfy the conditions if and only if

$$p^t h_t(x)\delta_g(x) + p^t g_t(x)\delta_h(x) \equiv p^t\lambda(x) \pmod{p^{t+1}}.$$

This, however, amounts to saying, after cancelling p^t from both sides, that

$$h_t(x)\delta_g(x) + g_t(x)\delta_h(x) \equiv \lambda(x) \pmod{p}.$$

Using the congruences $g_t(x) \equiv g(x) \pmod{p}$ and $h_t(x) \equiv h(x) \pmod{p}$ we obtain that this is equivalent to the congruence

$$h(x)\delta_g(x) + g(x)\delta_h(x) \equiv \lambda(x) \pmod{p}. \tag{1.23}$$

Considering the inequalities $\deg \delta_g(x) < \deg g_t(x)$ and $\deg \delta_h(x) < \deg h_t(x)$ and the fact that in $\mathbb{F}_p[x]$ the polynomials $g(x) \pmod{p}$ and $h(x) \pmod{p}$ are relatively prime, we find that equation (1.23) can be solved uniquely in $\mathbb{F}_p[x]$. For, if $u(x)$ and $v(x)$ form a solution to $u(x)g(x) + v(x)h(x) \equiv 1 \pmod{p}$, then, by Theorem 1.12, the polynomials

$$\delta_g(x) \quad = \quad v(x)\lambda(x) \pmod{g(x)},$$

and

$$\delta_h(x) \quad = \quad u(x)\lambda(x) \pmod{h(x)}$$

form a solution of (1.23). The uniqueness of the solution follows from the bounds on the degrees, and from the fact that $g(x) \pmod{p}$ and $h(x) \pmod{p}$ relatively prime. The details of this are left to the reader. ■

Corollary 1.79 *Assume that p, and the polynomials $f(x)$, $g(x)$, $h(x) \in \mathbf{Z}[x]$ satisfy the conditions of Hensel's lemma. Set* $\deg f = n$ *and let N be a positive integer. Then the polynomials $g_N(x)$ and $h_N(x)$ can be obtained using $O(Nn^2)$ arithmetic operations modulo p^N.*

Proof. The proof of Theorem 1.78 suggests the following algorithm.

HENSEL-LIFTING (f, g, h, p, N)

1 $(u(x), v(x)) \leftarrow$ is a solution, in $\mathbb{F}_p[x]$, of $u(x)g(x) + v(x)h(x) \equiv 1 \pmod{p}$
2 $(G(x), H(x)) \leftarrow (g(x), h(x))$
3 **for** $t \leftarrow 1$ **to** $N - 1$
4 **do** $\lambda(x) \leftarrow (f(x) - G(x) \cdot H(x))/p^t$
5 $\delta_g(x) \leftarrow v(x)\lambda(x)$ reduced modulo $g(x)$ (in $\mathbb{F}_p[x]$)
6 $\delta_h(x) \leftarrow u(x)\lambda(x)$ reduced modulo $h(x)$ (in $\mathbb{F}_p[x]$)
7 $(G(x), H(x)) \leftarrow (G(x) + p^t\delta_g(x), H(x) + p^t\delta_h(x))$ (in $(\mathbf{Z}/(p^{t+1}))[x]$)
8 **return** $(G(x), H(x))$

The polynomials u and v can be obtained using $O(n^2)$ operations in $\mathbb{F}_p$ (see Theorem 1.12 and the remark following it). An iteration $t \to t + 1$ consists of a constant number of operations with polynomials, and the cost of one run of the main loop is $O(n^2)$ operations (modulo p and p^{t+1}). The total cost of reaching $t = N$ is $O(Nn^2)$ operations. ■

1.5.2. The Berlekamp-Zassenhaus algorithm

The factorisation problem (1.18) was efficiently reduced to the case in which the polynomial f has integer coefficients and leading coefficient 1. We may also assume that $f(x)$ has no multiple factors in $\mathbb{Q}[x]$. Indeed, in our case $f'(x) \neq 0$, and so the possible multiple factors of f can be separated using the idea that we already used over finite fields as follows. By

Lemma 1.13, the polynomial $g(x) = f(x)/(f(x), f'(x))$ is already square-free, and, using Lemma 1.14, it suffices to find its factors with multiplicity one. From Proposition 1.71, we can see that $g(x)$ has integer coefficients and leading coefficient 1. Computing the greatest common divisor and dividing polynomials can be performed efficiently, and so the reduction can be carried out in polynomial time. (In the computation of the greatest common divisor, the intermediate expression swell can be avoided using the techniques presented in Chapter **??**.)

In the sequel we assume that the polynomial

$$f(x) = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbf{Z}[x]$$

we want to factor is square-free, its coefficients are integers, and its leading coefficient is 1.

The fundamental idea of the Berlekamp-Zassenhaus algorithm is that we compute the irreducible factors of $f(x)$ modulo p^N where p is a suitably chosen prime and N is large enough. If, for instance, $p^N > 2 \cdot 2^{n-1}\|f\|$, and we have already computed the coefficients of a factor modulo p^N, then, by Mignotte's theorem, we can obtain the coefficients of a factor in $\mathbb{Q}[x]$.

From now on, we will also assume that p is a prime such that the polynomial $f(x) \pmod p$ is square-free in $\mathbb{F}_p[x]$. Using linear search such a prime p can be found in polynomial time (Corollary 1.77). One can even assume that p is polynomial in the bit size of $f(x)$.

The irreducible factors in $\mathbb{F}_p[x]$ of the polynomial $f(x) \pmod p$ can be found using Berlekamp's deterministic method (Theorem 1.42). Let $g_1(x), \ldots, g_r(x) \in \mathbf{Z}[x]$ be polynomials, all with leading coefficient 1, such that the $g_i(x) \pmod p$ are the irreducible factors of the polynomial $f(x) \pmod p$ in $\mathbb{F}_p[x]$.

Using the technique of Hensel's lemma (Theorem 1.78) and Corollary 1.79, the system $g_1(x), \ldots, g_r(x)$ can be lifted modulo p^N. To simplify the notation, we assume now that $g_1(x), \ldots, g_r(x) \in \mathbf{Z}[x]$ are polynomials with leading coefficients 1 such that

$$f(x) \equiv g_1(x) \cdots g_r(x) \pmod{p^N}$$

and the $g_i(x) \pmod p$ are the irreducible factors of the polynomial $f(x) \pmod p$ in $\mathbb{F}_p[x]$.

Let $h(x) \in \mathbf{Z}[x]$ be an irreducible factor with leading coefficient 1 of the polynomial $f(x)$ in $\mathbb{Q}[x]$. Then there is a uniquely determined set $I \subseteq \{1, \ldots, r\}$ for which

$$h(x) \equiv \prod_{i \in I} g_i(x) \pmod{p^N} .$$

Let N be the smallest integer such that $p^N \geq 2 \cdot 2^{n-1}\|f(x)\|$. Mignotte's bound shows that the polynomial $\prod_{i \in I} g_i(x) \pmod{p^N}$ on the right-hand side, if its coefficients are represented by the residues with the smallest absolute values, coincides with h.

We found that determining the irreducible factors of $f(x)$ is equivalent to finding minimal subsets $I \subseteq \{1, \ldots, r\}$ for which there is a polynomial $h(x) \in \mathbf{Z}[x]$ with leading coefficient 1 such that $h(x) \equiv \prod_{i \in I} g_i(x) \pmod{p^N}$, the absolute values of the coefficients of $h(x)$ are at most $2^{n-1}\|f(x)\|$, and, moreover, $h(x)$ divides $f(x)$. This can be checked by examining at most 2^{r-1} sets I. The cost of examining a single I is polynomial in the size of f.

To summarise, we obtained the following method to factor, in $\mathbb{Q}[x]$, a square-free polynomial $f(x)$ with integer coefficients and leading coefficient 1.

BERLEKAMP-ZASSENHAUS(f)

1 $p \leftarrow$ a prime p such that $f(x) \pmod p$ is square-free in $\mathbb{F}_p[x]$
  and $p = O((n \lg n + 2n \lg \|f\|)^2)$
2 $\{g_1, \ldots, g_r\} \leftarrow$ the irreducible factors of $f(x) \pmod p$ in $\mathbb{F}_p[x]$
  (using Berlekamp's deterministic method)
3 $N \leftarrow \lfloor \log_p(2^{\deg f} \cdot \|f\|) \rfloor + 1$
4 $\{g_1, \ldots, g_r\} \leftarrow$ the Hensel lifting of the system $\{g_1, \ldots, g_r\}$ modulo p^N
5 $\mathcal{I} \leftarrow$ the collection of minimal subsets $I \neq \emptyset$ of $\{1, \ldots r\}$ such that
  $g_I \leftarrow \prod_{i \in I} g_i$ reduced modulo p^N divides f
6 **return** $\{\prod_{i \in I} g_i \ : \ I \in \mathcal{I}\}$

Theorem 1.80 *Let $f(x) = x^n + \sum_{i=0}^{n-1} a_i x^i \in \mathbf{Z}[x]$ be a square-free polynomial with integer coefficients and leading coefficient 1, and let p be a prime number such that the polynomial $f(x) \pmod p$ is square-free in $\mathbb{F}_p[x]$ and $p = O((n \lg n + 2n \lg \|f\|)^2)$. Then the irreducible factors of the polynomial f in $\mathbb{Q}[x]$ can be obtained by the Berlekamp-Zassenhaus algorithm. The cost of this algorithm is polynomial in n, $\lg \|f(x)\|$ and 2^r where r is the number of irreducible factors of the polynomial $f(x) \pmod p$ in $\mathbb{F}_p[x]$.*

Example 1.5 (Swinnerton-Dyer polynomials) Let

$$f(x) = \prod (x \pm \sqrt{2} \pm \sqrt{3} \pm \cdots \pm \sqrt{p_l}) \in \mathbf{Z}[x] \ ,$$

where $2, 3, \ldots, p_l$ are the first l prime numbers, and the product is taken over all possible 2^l combinations of the signs + and −. The degree of $f(x)$ is $n = 2^l$, and one can show that it is irreducible in $\mathbb{Q}[x]$. On the other hand, for all primes p, the polynomial $f(x) \pmod p$ is the product of factors with degree at most 2. Therefore these polynomials represent hard cases for the Berlekamp-Zassenhaus algorithm, as we need to examine about $2^{n/2-1}$ sets I to find out that f is irreducible.

1.5.3. The LLL algorithm

Our goal in this section is to present the Lenstra-Lenstra-Lovász algorithm (LLL algorithm) for factoring polynomials $f(x) \in \mathbb{Q}[x]$. This was the first polynomial time method for solving the polynomial factorisation problem over $\mathbb{Q}$. Similarly to the Berlekamp-Zassenhaus method, the LLL algorithm starts with a factorisation of f modulo p and then uses Hensel lifting. In the final stages of the work, it uses lattice reduction to find a proper divisor of f, provided one exists. The powerful idea of the LLL algorithm is that it replaced the search, which may have exponential complexity, in the Berlekamp-Zassenhaus algorithm by an efficient lattice reduction.

Let $f(x) \in \mathbf{Z}[x]$ be a square-free polynomial with leading coefficient 1 such that $\deg f = n > 1$, and let p be a prime such that the polynomial $f(x) \pmod p$ is square free in $\mathbb{F}_p[x]$ and $p = O((\lg n + 2n \lg \|f\|)^2)$.

Lemma 1.81 *Suppose that $f(x) \equiv g_0(x)v(x) \pmod{p^N}$ where $g_0(x)$ and $v(x)$ are polynomials with integer coefficients and leading coefficient 1. Let $g(x) \in \mathbf{Z}[x]$ with $\deg g(x) = m < n$ and assume that $g(x) \equiv g_0(x)u(x) \pmod{p^N}$ for some polynomial $u(x)$ such that $u(x)$ has integer coefficients and $\deg u(x) = \deg g(x) - \deg g_0(x)$. Let us further assume that $\|g(x)\|^n \|f(x)\|^m < p^N$. Then $\gcd(f(x), g(x)) \neq 1$ in $\mathbb{Q}[x]$.*

Proof. Let $d = \deg v(x)$. By the assumptions,

$$f(x)u(x) \equiv g_0(x)u(x)v(x) \equiv g(x)v(x) \pmod{p^N} .$$

Suppose that $u(x) = \alpha_0 + \alpha_1 x + \ldots + \alpha_{m-1}x^{m-1}$ and $v(x) = \beta_0 + \beta_1 x + \ldots + \beta_{n-1}x^{n-1}$. (We know that $\beta_d = 1$. If $i > d$, then $\beta_i = 0$, and similarly, if $j > \deg u(x)$, then $\alpha_j = 0$.) Rewriting the congruence, we obtain

$$x^d g(x) + \sum_{j \neq d} \beta_j x^j g(x) - \sum_i \alpha_i x^i f(x) \equiv 0 \pmod{p^N} .$$

Considering the coefficient vectors of the polynomials $x^j g(x)$ and $x^i f(x)$, this congruence amounts to saying that adding to the $(m+d)$-th row of the Sylvester matrix (1.20) a suitable linear combination of the other rows results in a row in which all the elements are divisible by p^N. Consequently, $\det M \equiv 0 \pmod{p^N}$. The Hadamard inequality (Corollary 1.60) yields that $|\det M| \leq \|f\|^m \|g\|^n < p^N$, but this can only happen if $\det M = 0$. However, $\det M = Res(f(x), g(x))$, and so, by (1.21), $\gcd(f(x), g(x)) \neq 1$. ∎

The application of lattice reduction

Set

$$N = \lceil \log_p (2^{2n^2} \|f(x)\|^{2n}) \rceil = O(n^2 + n \lg \|f(x)\|) .$$

Further, we let $g_0(x) \in \mathbf{Z}[x]$ be a polynomial with leading coefficient 1 such that $g_0(x) \pmod{p^N}$ is an irreducible factor of $f(x) \pmod{p^N}$. Set $d = \deg g_0(x) < n$. Define the set L as follows:

$$L = \{g(x) \in \mathbf{Z}[x] : \ \deg g(x) \leq n-1, \ \exists h(x) \in \mathbf{Z}[x], \text{ with } g \equiv hg_0 \pmod{p^N}\} . \quad (1.24)$$

Clearly, L is closed under addition of polynomials. We identify a polynomial with degree less than n with its coefficient vector of length n. Under this identification, L becomes a lattice in $\mathbb{R}^n$. Indeed, it is not too hard to show (Exercise 1.5-2.) that the polynomials

$$p^N 1, p^N x, \ldots, p^N x^{d-1}, \ g_0(x), x g_0(x), \ldots, x^{n-d-1} g_0(x) ,$$

or, more precisely, their coefficient vectors, form a basis of L.

Theorem 1.82 *Let $g_1(x) \in \mathbf{Z}[x]$ be a polynomial with degree less than n such that the coefficient vector of $g_1(x)$ is the first element in a Lovász-reduced basis of L. Then $f(x)$ is irreducible in $\mathbb{Q}[x]$ if and only if $\gcd(f(x), g_1(x)) = 1$.*

Proof. As $g_1(x) \neq 0$, it is clear that $\gcd(f(x), g_1(x)) = 1$ whenever $f(x)$ is irreducible. In order to show the implication in the other direction, let us assume that $f(x)$ is reducible and let $g(x)$ be a proper divisor of $f(x)$ such that $g(x) \pmod{p}$ is divisible by $g_0(x) \pmod{p}$ in

$\mathbb{F}_p[x]$. Using Hensel's lemma (Theorem 1.78), we conclude that $g(x) \pmod{p^N}$ is divisible by $g_0(x) \pmod{p^N}$, that is, $g(x) \in L$. Mignotte's theorem (Theorem 1.74) shows that

$$\|g(x)\| \leq 2^{n-1}\|f(x)\| .$$

Now, if we use the properties of reduced bases (second assertion of Theorem 1.67), then we obtain

$$\|g_1(x)\| \leq 2^{(n-1)/2}\|g(x)\| < 2^n\|g(x)\| \leq 2^{2n}\|f(x)\| ,$$

and so

$$\|g_1(x)\|^n\|f(x)\|^{\deg g_1} \leq \|g_1(x)\|^n\|f(x)\|^n < 2^{2n^2}\|f(x)\|^{2n} \leq p^N .$$

We can hence apply Lemma 1.81, which gives $\gcd(g_1(x), f(x)) \neq 1$. ∎

Based on the previous theorem, the LLL algorithm can be outlined as follows (we only give a version for factoring to two factors). The input is a square-free polynomial $f(x) \in \mathbf{Z}[x]$ with integer coefficients and leading coefficient 1 such that $\deg f = n > 1$.

LLL-Polynomial-Factorisation(f)

```
1  p ← a prime p such that f(x) (mod p) is square-free in F_p[x]
       and p = O((n lg n + 2n lg ||f||)^2)
2  w(x) ← an irreducible factor f(x) (mod p) in F_p[x]
       (using Berlekamp's deterministic method)
3  if deg w = n
4     then return "irreducible"
5     else N ← ⌈log_p((2^{2n^2} ||f(x)||^{2n})⌉ = O(n^2 + n lg(||f(x)||)
6          (g_0, h_0) ← Hensel-Lifting(f, w, f/w (mod p), p, N)
7          (b_1, ..., b_n) ← a basis of the lattice L ⊆ R^n in (1.24)
8          (g_1, ..., g_n) ← Lovász-Reduction(b_1, ..., b_n)
9          f* ← gcd(f, g_1)
10         if deg f* > 0
11            then return (f*, f/f*)
12            else return "irreducible"
```

Theorem 1.83 *Using the LLL algorithm, the irreducible factors in $\mathbb{Q}[x]$ of a polynomial $f \in \mathbb{Q}[x]$ can be obtained deterministically in polynomial time.*

Proof. The general factorisation problem, using the method introduced at the discussion of the Berlekamp-Zassenhaus procedure, can be reduced to the case in which the polynomial $f(x) \in \mathbf{Z}[x]$ is square-free and has leading coefficient 1. By the observations made there, the steps in lines 1–7 can be performed in polynomial time. In line 8, the Lovász reduction can be carried out efficiently (Corollary 1.65). In line 9, we may use a modular version of the Euclidean algorithm to avoid intermediate expression swell (see Chapter **??**).

The correctness of the method is asserted by Theorem 1.82. The LLL algorithm can be applied repeatedly to factor the polynomials in the output, in case they are not already irreducible. ∎

One can show that the Hensel lifting costs $O(Nn^2) = O(n^4 + n^3 \lg \|f\|)$ operations with moderately sized integers. The total cost of the version of the LLL algorithm above

is $O(n^5 \lg(p^N)) = O(n^7 + n^6 \lg \|f\|)$.

Exercises

1.5-1 Let $\mathbb{F}$ be a field and let $0 \neq f(x) \in \mathbb{F}[x]$. The polynomial $f(x)$ has no irreducible factors with multiplicity greater than one if and only if $\gcd(f(x), f'(x)) = 1$. *Hint:* In one direction, one can use Lemma 1.13, and use Lemma 1.14 in the other.

1.5-2 Show that the polynomials

$$p^N 1, p^N x, \ldots, p^N x^{d-1}, g_0(x), x g_0(x), \ldots, x^{n-d-1} g_0(x)$$

form a basis of the lattice in (1.24). *Hint:* It suffices to show that the polynomials $p^N x^j$ $(d \leq j < n)$ can be expressed with the given polynomials. To show this, divide $p^N x^j$ by $g_0(x)$ and compute the remainder.

Problems

1-1. The trace in finite fields

Let $\mathbb{F}_{q^k} \supseteq \mathbb{F}_q$ be finite fields. The definition of the trace map $tr = tr_{k,q}$ on $\mathbb{F}_{q^k}$ is as follows: if $\alpha \in \mathbb{F}_{q^k}$ then

$$tr(\alpha) = \alpha + \alpha^q + \cdots + \alpha^{q^{k-1}} .$$

(a) Show that the map tr is $\mathbb{F}_q$-linear and its image is precisely $\mathbb{F}_q$. *Hint:* Use the fact that tr is defined using a polynomial with degree q^{k-1} to show that tr is not identically zero.

(b) Let (α, β) be a uniformly distributed random pair of elements from $\mathbb{F}_{q^k} \times \mathbb{F}_{q^k}$. Then the probability that $tr(\alpha) \neq tr(\beta)$ is $1 - 1/q$.

1-2. The Cantor-Zassenhaus algorithm for fields of characteristic 2

Let $\mathbb{F} = \mathbb{F}_{2^m}$ and let $f(x) \in \mathbb{F}[x]$ be a polynomial of the form

$$f = f_1 f_2 \cdots f_s, \tag{1.25}$$

where the f_i are pairwise relatively prime and irreducible polynomials with degree d in $\mathbb{F}[x]$. Also assume that $s \geq 2$.

(a) Let $u(x) \in \mathbb{F}[x]$ be a uniformly distributed random polynomial with degree less than $\deg f$. Then the greatest common divisor

$$\gcd(u(x) + u^2(x) + \cdots + u^{2^{md-1}}(x), f(x))$$

is a proper divisor of $f(x)$ with probability at least $1/2$.

Hint: Apply the previous exercise taking $q = 2$ and $k = md$, and follow the argument in Theorem 1.38.

(b) Using part (a), give a randomised polynomial time method for factoring a polynomial of the form (1.25) over $\mathbb{F}$.

1-3. Divisors and zero divisors

Let $\mathbb{F}$ be a field. The ring R is said to be an $\mathbb{F}$-***algebra*** (in case $\mathbb{F}$ is clear from the context, R is simply called an algebra), if R is a vector space over $\mathbb{F}$, and $(ar)s = a(rs) = r(as)$ holds for all $r, s \in S$ and $a \in \mathbb{F}$. It is easy to see that the rings $\mathbb{F}[x]$ and $\mathbb{F}[x]/(f)$ are $\mathbb{F}$-algebras.

Let R be a finite-dimensional $\mathbb{F}$-algebra. For an arbitrary $r \in R$, we may consider the map $L_r : R \to R$ defined as $L_r(s) = rs$ for $s \in R$. The map L_r is $\mathbb{F}$-linear, and so we may speak about its minimal polynomial $m_r(x) \in \mathbb{F}[x]$, its characteristic polynomial $k_r(x) \in \mathbb{F}[x]$, and its trace $Tr(r) = Tr(L_r)$. In fact, if U is an ideal in R, then U is an invariant subspace of L_r, and so we can restrict L_r to U, and we may consider the minimal polynomial, the characteristic polynomial, and the trace of the restriction.

(a) Let $f(x),\ g(x) \in \mathbb{F}[x]$ with $\deg f > 0$. Show that the residue class $[g(x)]$ is a zero divisor in the ring $\mathbb{F}[x]/(f)$ if and only if f does not divide g and $\gcd(f(x), g(x)) \neq 1$.

(b) Let R be an algebra over $\mathbb{F}$, and let $r \in R$ be an element with minimal polynomial $f(x)$. Show that if f is not irreducible over $\mathbb{F}$, then R contains a zero divisor. To be precise, if $f(x) = g(x)h(x)$ is a non-trivial factorisation ($g, h \in \mathbb{F}[x]$), then $g(r)$ and $h(r)$ form a pair of zero divisors, that is, both of them are non-zero, but their product is zero.

1-4. Factoring polynomials over algebraic number fields

(a) Let $\mathbb{F}$ be a field with characteristic zero and let R be a finite-dimensional $\mathbb{F}$-algebra with an identity element. Let us assume that $R = S_1 \oplus S_2$ where S_1 and S_2 are non-zero $\mathbb{F}$-algebras. Let $r_1, \ldots, r_k$ be a basis of R over $\mathbb{F}$. Show that there is a j such that $m_{r_j}(x)$ is not irreducible in $\mathbb{F}[x]$.

Hint: This exercise is for readers who are familiar with the elements of linear algebra. Let us assume that the minimal polynomial of r_j is the irreducible polynomial $m(x) = x^d - a_1 x^{d-1} + \cdots + a_d$. Let $k_i(x)$ be the characteristic polynomial of L_{r_j} on the invariant subspace U_i (for $i \in \{1, 2\}$). Here U_1 and U_2 are the sets of elements of the form $(s_1, 0)$ and $(0, s_2)$, respectively where $s_i \in S_i$. Because of our conditions, we can find suitable exponents d_i such that $k_i(x) = m(x)^{d_i}$. This implies that the trace $T_i(r_j)$ of the map L_{r_j} on the subspace U_i is $T_i(r_j) = d_i a_1$. Set $e_i = \dim_{\mathbb{F}} U_i$. Obviously, $e_i = d_i d$, which gives $T_1(r_j)/e_1 = T_2(r_j)/e_2$. If the assertion of the exercise is false, then the latter equation holds for all j, and so, as the trace is linear, it holds for all $r \in R$. This, however, leads to a contradiction: if $r = (1, 0) \in S_1 \oplus S_2$ (1 denotes the unity in S_1), then clearly $T_1(r) = e_1$ and $T_2(r) = 0$.

(b) Let $\mathbb{F}$ be an ***algebraic number field***, that is, a field of the form $\mathbb{Q}(\alpha)$ where $\alpha \in \mathbb{C}$, and there is an irreducible polynomial $g(x) \in \mathbf{Z}[x]$ such that $g(\alpha) = 0$. Let $f(x) \in \mathbb{F}[x]$ be a square-free polynomial and set $R = \mathbb{F}[x]/(f)$. Show that R is a finite-dimensional algebra over $\mathbb{Q}$. More precisely, if $\deg g = m$ and $\deg f = n$, then the elements of the form $\alpha^i[x]^j$ $(0 \leq i < m,\ 0 \leq j < n)$ form a basis over $\mathbb{Q}$.

(c) Show that if f is reducible over $\mathbb{F}$, then there are $\mathbb{Q}$-algebras $S_1,\ S_2$ such that $R \cong S_1 \oplus S_2$.

Hint: Use the Chinese remainder theorem .

(d) Consider the polynomial g above and suppose that a field $\mathbb{F}$ and a polynomial $f \in \mathbb{F}[x]$ are given. Assume, further, that f is square-free and is not irreducible over $\mathbb{F}$. The polynomial f can be factored to the product of two non-constant polynomials in

polynomial time.

Hint: By the previous remarks, the minimal polynomial $m(y)$ over $\mathbb{Q}$ of at least one of the elements $\alpha^i[x]^j$ $(0 \le i \le m,\ 0 \le j \le n)$ is not irreducible in $\mathbb{Q}[y]$. Using the LLL algorithm, $m(y)$ can be factored efficiently in $\mathbb{Q}[y]$. From a factorisation of $m(y)$, a zero divisor of R can be obtained, and this can be used to find a proper divisor of f in $\mathbb{F}[x]$.

Chapter notes

The abstract algebraic concepts discussed in this chapter can be found in many textbooks; see, for instance, Hungerford's book [27].

The theory of finite fields and the related algorithms are the theme of the excellent books by Lidl and Niederreiter [36] and Shparlinski [57].

Our main algorithmic topics, namely the factorisation of polynomials and lattice reduction are thoroughly treated in the book by von zur Gathen and Gerhard [16]. We recommend the same book to the readers who are interested in the efficient methods to solve the basic problems concerning polynomials. Theorem 8.23 of that book estimates the cost of multiplying polynomials by the Schönhage-Strassen method, while Corollary 11.6 is concerned with the cost of the asymptotically fast implementation of the Euclidean algorithm. Ajtai's result about shortest lattice vectors was published in [3].

The method by Kaltofen and Shoup is a randomised algorithm for factoring polynomials over finite fields, and currently it has one of the best time bounds among the known algorithms. The expected number of $\mathbb{F}_q$-operations in this algorithm is $O(n^{1.815} \lg q)$ where $n = \deg f$. Further competitive methods were suggested by von zur Gathen and Shoup, and also by Huang and Pan. The number of operations required by the latter is $O(n^{1.80535} \lg q)$, if $\lg q < n^{0.00173}$. Among the deterministic methods, the one by von zur Gathen and Shoup is the current champion. Its cost is $\widetilde{O}(n^2 + n^{3/2}s + n^{3/2}s^{1/2}p^{1/2})$ operations in $\mathbb{F}_q$ where $q = p^s$. An important related problem is constructing the field $\mathbb{F}_{q^n}$. The fastest randomised method is by Shoup. Its cost is $\tilde{O}(n^2 + n \lg q)$. For finding a square-free factorisation, Yun gave an algorithm that requires $\widetilde{O}(n) + O(n \lg(q/p))$ field operations in $\mathbb{F}_q$.

The best methods to solve the problem of lattice reduction and that of factoring polynomials over the rationals use modular and numerical techniques. After slightly modifying the definition of reduced bases, an algorithm using $\widetilde{O}(n^{3.381} \lg^2 C)$ *bit operations* for the former problem was presented by Storjohann. (We use the original definition introduced in the paper by Lenstra, Lenstra and Lovász [35].) We also mention Schönhage's method using $\widetilde{O}(n^6 + n^4 \lg^2 l)$ bit operations for factoring polynomials with integer coefficients (l is the length of the coefficients).

Besides factoring polynomials with rational coefficients, lattice reduction can also be used to solve lots of other problems: to break knapsack cryptosystems and random number generators based on linear congruences, simultaneous Diophantine approximation, to find integer linear dependencies among real numbers (this problem plays an important rôle in experiments that attempt to find mathematical identities). These and other related problems are discussed in the book [16].

A further exciting application area is the numerical solution of Diophantine equations. One can read about these developments in in the books by Smart [62] and Gaál [14]. The

difficulty of finding a shortest lattice vector was verified in Ajtai's paper [3].

Finally we remark that the practical implementations of the polynomial methods involving lattice reduction are not competitive with the implementations of the Berlekamp-Zassenhaus algorithm, which, in the worst case, has exponential complexity. Nevertheless, the basis reduction performs very well in practice: in fact it is usually much faster than its theoretically proven speed. For some of the problems in the application areas listed above, we do not have another useful method.

The work of the authors was supported in part by grants T042481 and T042706 of the Hungarian Scientific Research Fund.

2. Automata and Formal Languages

Automata and formal languages play an important role in projecting and realizing compilers. In the first section grammars and formal languages are defined. The different grammars and languages are discussed based on Chomsky hierarchy. In the second section we deal in detail with the finite automata and the languages accepted by them, while in the third section the pushdown automata and the corresponding accepted languages are discussed. Finally, references from a rich bibliography are given.

2.1. Languages and grammars

A finite and nonempty set of symbols is called an ***alphabet***. The elements of an alphabet are ***letters***, but sometimes are named also ***symbols***. E.g. the set $\Sigma = \{a, b, c, d, 0, 1, \sigma\}$ is an alphabet, with the letters $a, b, c, d, 0, 1$ and σ.

With the letters of an alphabet words are composed. If $a_1, a_2, \ldots, a_n \in \Sigma, n \geq 0$, then $a_1 a_2 \ldots a_n$ a Σ is a ***word*** over the alphabet Σ (the letters a_i are not necessary distinct). The number of letters of a word, with their multiplicities, constitutes the ***length*** of the word. If $w = a_1 a_2 \ldots a_n$, then the length of w is $|w| = n$. If $n = 0$, then the word is an ***empty word***, which will be denoted by ε (sometimes λ in other books). The set of words over the alphabet Σ will be denoted by Σ^*:

$$\Sigma^* = \{a_1 a_2 \ldots a_n \mid a_1, a_2, \ldots, a_n \in \Sigma,\ n \geq 0\} .$$

For the set of nonempty words over Σ the notation $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ will be used. The set of words of length n over Σ will be denoted by Σ^n, and $\Sigma^0 = \{\varepsilon\}$. Then

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \cdots \cup \Sigma^n \cup \cdots \quad \text{and} \quad \Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n \cup \cdots .$$

The words $u = a_1 a_2 \ldots a_m$ and $v = b_1 b_2 \ldots b_n$ are equal (i.e. $u = v$), if $m = n$ and $a_i = b_i$, $i = 1, 2, \ldots, n$.

We define in Σ^* the binary operation called ***concatenation***. The concatenation (or product) of the words $u = a_1 a_2 \ldots a_m$ and $v = b_1 b_2 \ldots b_n$ is the word $uv = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$. It is clear that $|uv| = |u| + |v|$. This operation is associative but not commutative. Its neutral element is ε, because $\varepsilon u = u\varepsilon = u$ for all $u \in \Sigma^*$. Σ^* with the concatenation is a monoid.

We introduce the power operation. If $u \in \Sigma^*$, then $u^0 = \varepsilon$, and $u^n = u^{n-1}u$ for $n \geq 1$.

The ***reversal*** (or ***mirror image***) of the word $u = a_1 a_2 \dots a_n$ is $u^{-1} = a_n a_{n-1} \dots a_1$. The reversal of u sometimes is denoted by u^R or $\tilde{u}$. It is clear that $\left(u^{-1}\right)^{-1} = u$ and $(uv)^{-1} = v^{-1}u^{-1}$.

Word v is a ***prefix*** of the word u if there exists a word z such that $u = vz$. If $z \neq \varepsilon$ then v is a proper prefix of u. Similarly v is a ***suffix*** of u if there exists a word x such that $u = xv$. The proper suffix can also be defined. Word v is a ***subword*** of the word u if there are words p and q such that $u = pvq$. If $pq \neq \varepsilon$ then v is a ***proper subword***.

A subset L of Σ^* is called a ***language*** over the alphabet Σ. Sometimes this is called a ***formal language*** because the words are here considered without any meanings. Note that $\emptyset$ is the empty language while $\{\varepsilon\}$ is a language which contains the empty word.

2.1.1. Operations on languages

If L, L_1, L_2 are languages over Σ we define the following operations

• *union*

$$L_1 \cup L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ or } u \in L_2\} ,$$

• *intersection*

$$L_1 \cap L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ and } u \in L_2\} ,$$

• *difference*

$$L_1 \setminus L_2 = \{u \in \Sigma^* \mid u \in L_1 \text{ and } u \notin L_2\} ,$$

• *complement*

$$\overline{L} = \Sigma^* \setminus L ,$$

• *multiplication*

$$L_1 L_2 = \{uv \mid u \in L_1, v \in L_2\} ,$$

• *power*

$$L^0 = \{\varepsilon\}, \qquad L^n = L^{n-1}L, \text{ if } n \geq 1 ,$$

• *iteration* or *star operation*

$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L \cup L^2 \cup \dots \cup L^i \cup \dots ,$$

• *mirror*

$$L^{-1} = \{u^{-1} \mid u \in L\}$$

We will use also the notation L^+

$$L^+ = \bigcup_{i=1}^{\infty} L^i = L \cup L^2 \cup \dots \cup L^i \cup \dots .$$

The union, product and iteration are called ***regular operations***.

2.1.2. Specifying languages

Languages can be specified in several ways. For example a language can be specified using

1) the enumeration of its words,

2) a property, such that all words of the language have this property but other word have not,

3) a grammar.

Specifying languages by listing their elements

For example the following are languages

$L_1 = \{\varepsilon, 0, 1\}$,
$L_2 = \{a, aa, aaa, ab, ba, aba\}$.
Even if we cannot enumerate the elements of an infinite set infinite languages can be specified by enumeration if after enumerating the first some elements we can continue the enumeration using a rule. The following is such a language
$L_3 = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$.

Specifying languages by properties
The following sets are languages
$L_4 = \{a^n b^n \mid n = 0, 1, 2, \ldots\}$,
$L_5 = \{uu^{-1} \mid u \in \Sigma^*\}$,
$L_6 = \{u \in \{a, b\}^* \mid n_a(u) = n_b(u)\}$,
where $n_a(u)$ denotes the number of letters a in word u and $n_b(u)$ the number of letters b.

Specifying languages by grammars
Define the ***generative grammar*** or shortly the ***grammar***.

Definition 2.1 *A **grammar** is an ordered quadruple $G = (N, T, P, S)$, where*
- *N is the alphabet of **variables** (or **nonterminal symbols**),*
- *T is the alphabet of **terminal symbols**, where $N \cap T = \emptyset$,*
- *$P \subseteq (N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$ is a finite set, that is P is the finite set of **productions** of the form (u, v) , where $u, v \in (N \cup T)^*$ and u contains at least a nonterminal symbol,*
- *$S \in N$ is the **start symbol**.*

Remarks. Instead of the notation (u, v) sometimes $u \to v$ is used.

In the production $u \to v$ or (u, v) word u is called the left-hand side of the production while v the right-hand side. If for a grammar there are more than one production with the same left-hand side, then these production

$$u \to v_1,\ u \to v_2, \ldots, u \to v_r \quad \text{can be written as} \quad u \to v_1 \mid v_2 \mid \ldots \mid v_r\ .$$

We define on the set $(N \cup T)^*$ the relation called ***direct derivation***

$$u \Longrightarrow v, \qquad \text{if} \quad u = p_1 p p_2, \quad v = p_1 q p_2 \quad \text{and} \quad (p, q) \in P\ .$$

In fact we replace in u an appearance of the subword p by q and we get v. Another notations for the same relation can be $\vdash$ or $\models$.

If we want to emphasize the used grammar G, then the notation $\Longrightarrow$ can be replaced by $\underset{G}{\Longrightarrow}$. Relation $\overset{*}{\Longrightarrow}$ is the reflexive and transitive closure of $\Longrightarrow$, while $\overset{+}{\Longrightarrow}$ denotes its transitive closure. Relation $\overset{*}{\Longrightarrow}$ is called a ***derivation***.

From the definition of a reflexive and transitive relation we can deduce the following: $u \overset{*}{\Longrightarrow} v$, if there exist the words $w_0, w_1, \ldots, w_n \in (N \cup T)^*$, $n \geq 0$ and $u = w_0$, $w_0 \Longrightarrow w_1$, $w_1 \Longrightarrow w_2$, $\ldots, w_{n-1} \Longrightarrow w_n$, $w_n = v$. This can be written shortly $u = w_0 \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \ldots \Longrightarrow w_{n-1} \Longrightarrow w_n = v$. If $n = 0$ then $u = v$. The same way we can define the relation $u \overset{+}{\Longrightarrow} v$ except that $n \geq 1$ always, so at least one direct derivation will de used.

Definition 2.2 *The **language generated** by grammar $G = (N, T, P, S)$ is the set*

$$L(G) = \{u \in T^* \mid S \overset{*}{\Longrightarrow} u\} .$$

So $L(G)$ contains all words over the alphabet T which can be derived from the start symbol S using the productions from P.

Example 2.1 Let $G = (N, T, P, S)$ where
$N = \{S\}$,
$T = \{a, b\}$,
$P = \{S \to aSb,\ S \to ab\}$.
It is easy to see than $L(G) = \{a^n b^n \mid n \geq 1\}$ because

$$S \underset{G}{\Longrightarrow} aSb \underset{G}{\Longrightarrow} a^2 S b^2 \underset{G}{\Longrightarrow} \cdots \underset{G}{\Longrightarrow} a^{n-1} S b^{n-1} \underset{G}{\Longrightarrow} a^n b^n ,$$

where up to the last but one replacement the first production ($S \to aSb$) was used, while at the last replacement the production $S \to ab$. This derivation can be written $S \overset{*}{\underset{G}{\Longrightarrow}} a^n b^n$. Therefore $a^n b^n$ can be derived from S for all n and no other words can be derived from S.

Definition 2.3 *Two grammars G_1 and G_2 are* ***equivalent,*** *and this is denoted by $G_1 \cong G_2$ if $L(G_1) = L(G_2)$.*

Example 2.2 The following two grammars are equivalent because both of them generate the language $\{a^n b^n c^n \mid n \geq 1\}$.
$G_1 = (N_1, T, P_1, S_1)$, where
$N_1 = \{S_1, X, Y\},\ T = \{a, b, c\}$,
$P_1 = \{S_1 \to abc,\ S_1 \to aXbc,\ Xb \to bX,\ Xc \to Ybcc,\ bY \to Yb,\ aY \to aaX,\ aY \to aa\}$.
$G_2 = (N_2, T, P_2, S_2)$, where
$N_2 = \{S_2, A, B, C\}$,
$P_2 = \{S_2 \to aS_2BC,\ S_2 \to aBC,\ CB \to BC,\ aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc\}$.
First let us prove by mathematical induction that for $n \geq 2$ $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} Y b^n c^n$. If $n = 2$ then

$$S_1 \underset{G_1}{\Longrightarrow} aXbc \underset{G_1}{\Longrightarrow} abXc \underset{G_1}{\Longrightarrow} abYbcc \underset{G_1}{\Longrightarrow} aYb^2c^2 .$$

The inductive hypothesis is $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-2} Y b^{n-1} c^{n-1}$. We use production $aY \to aaX$, then $(n-1)$ times production $Xb \to bX$, and then production $Xc \to Ybcc$, afterwards again $(n-1)$ times production $bY \to Yb$. Therefore

$$S_1 \underset{G_1}{\Longrightarrow} a^{n-2} Y b^{n-1} c^{n-1} \underset{G_1}{\Longrightarrow} a^{n-1} X b^{n-1} c^{n-1} \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} b^{n-1} X c^{n-1} \underset{G_1}{\Longrightarrow} a^{n-1} b^{n-1} Y b c^n \overset{*}{\underset{G_1}{\Longrightarrow}} a^{n-1} Y b^n c^n .$$

If now we use production $aY \to aa$ we get $S_1 \overset{*}{\underset{G_1}{\Longrightarrow}} a^n b^n c^n$ for $n \geq 2$, but $S_1 \underset{G_1}{\Longrightarrow} abc$ by the production $S_1 \to abc$, so $a^n b^n c^n \in L(G_1)$ for any $n \geq 1$. We have to prove in addition that using the productions of the grammar we cannot derive only words of the form $a^n b^n c^n$. It is easy to see that a successful derivation (which ends in a word containing only terminals) can be obtained only in the presented way.
Similarly for $n \geq 2$

$$S_2 \underset{G_2}{\Longrightarrow} aS_2BC \overset{*}{\underset{G_2}{\Longrightarrow}} a^{n-1} S_2 (BC)^{n-1} \underset{G_2}{\Longrightarrow} a^n (BC)^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^n B^n C^n$$

$$\underset{G_2}{\Longrightarrow} a^n bB^{n-1}C^n \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n C^n \underset{G_2}{\Longrightarrow} a^n b^n cC^{n-1} \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n c^n \ .$$

Here orderly were used the productions $S_2 \to aS_2BC$ ($n-1$ times), $S_2 \to aBC$, $CB \to BC$ ($n-1$ times), $aB \to ab$, $bB \to bb$ ($n-1$ times), $bC \to bc$, $cC \to cc$ ($n-1$ times). But $S_2 \underset{G_2}{\Longrightarrow} aBC \underset{G_2}{\Longrightarrow} abC \underset{G_2}{\Longrightarrow} abc$, So $S_2 \overset{*}{\underset{G_2}{\Longrightarrow}} a^n b^n c^n$, $n \geq 1$. It is also easy to see than other words cannot be derived using grammar G_2.

The grammars

$G_3 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to \varepsilon\}, S)$ and

$G_4 = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$

are not equivalent because $L(G_3) \setminus \{\varepsilon\} = L(G_4)$.

Theorem 2.4 *Not all languages can be generated by grammars.*

Proof. We encode grammars for the proof as words on the alphabet $\{0, 1\}$. For a given grammar $G = (N, T, P, S)$ let $N = \{S_1, S_2, \ldots, S_n\}$, $T = \{a_1, a_2, \ldots, a_m\}$ and $S = S_1$. The encoding is the following:

the code of S_i is $10\underbrace{11\ldots11}_{i \text{ times}}01$, the code of a_i is $100\underbrace{11\ldots11}_{i \text{ times}}001$.

In the code of the grammar the letters are separated by 000, the code of the arrow is 0000, and the productions are separated by 00000.

It is enough, of course, to encode the productions only. For example, consider the grammar

$G = (\{S\}, \{a, b\}, \{S \to aSb,\ S \to ab\}, S)$.

The code of S is 10101, the code of a is 1001001, the code of b is 10011001. The code of the grammar is

$$\underbrace{10101}\,0000\,\underbrace{1001001}\,000\,\underbrace{10101}\,000\,\underbrace{10011001}\,00000\,\underbrace{10101}\,0000\,\underbrace{1001001}\,000$$

$$\underbrace{10011001}\ .$$

From this encoding results that the grammars with terminal alphabet T can be enumerated [1] as $G_1, G_2, \ldots, G_k, \ldots$, and the set of these grammars is a denumerable infinite set.

Consider now the set of all languages over T denoted by $\mathcal{L}_T = \{L \mid L \subseteq T^*\}$, that is $\mathcal{L}_T = \mathcal{P}(T^*)$. The set T^* is denumerable because its words can be ordered. Let this order $s_0, s_1, s_2, \ldots$, where $s_0 = \varepsilon$. We associate to each language $L \in \mathcal{L}_T$ an infinite binary sequence $b_0, b_1, b_2, \ldots$ the following way:

$$b_i = \begin{cases} 1, & \text{if } s_i \in L \\ 0, & \text{if } s_i \notin L \end{cases} \qquad i = 0, 1, 2, \ldots \ .$$

It is easy to see that the set of all such binary sequences is not denumerable, because each sequence can be considered as a positive number less than 1 using its binary representation

[1] Let us suppose that in the alphabet $\{0, 1\}$ there is a linear order $<$, let us say $0 < 1$. The words which are codes of grammars can be enumerated by ordering them first after their lengths, and inside the equal length words, alphabetically, using the order of their letters. But we can use equally the lexicographic order, which means that $u < v$ (u is before v) if u is a proper prefix of v or there exists the decompositions $u = xay$ and $v = xby'$, where x, y, y' are subwords, a and b letters with $a < b$.

(The decimal point is considered to be before the first digit). Conversely, to each positive number less than 1 in binary representation a binary sequence can be associated. So, the cardinality of the set of infinite binary sequences is equal to cardinality of interval $[0, 1]$, which is of continuum power. Therefore the set $\mathcal{L}_T$ is of continuum cardinality. Now to each grammar with terminal alphabet T associate the corresponding generated language over T. Since the cardinality of the set of grammars is denumerable, there will exist a language from $\mathcal{L}_T$, without associated grammar, a language which cannot be generated by a grammar. ■

2.1.3. Chomsky hierarchy of grammars and languages

Putting some restrictions on the form of productions, four type of grammars can be distinguished.

Definition 2.5 *Define for a grammar $G = (N, T, P, S)$ the following four types.*

*A grammar G is of type 0 (**phrase-structure grammar**) if there are no restrictions on productions.*

*A grammar G is of type 1 **(context-sensitive grammar)** if all of its productions are of the form $\alpha A\gamma \to \alpha\beta\gamma$, where $A \in N$, $\alpha, \gamma \in (N \cup T)^*$, $\beta \in (N \cup T)^+$. A production of the form $S \to \varepsilon$ can also be accepted if the start symbol S does not occur in the right-hand side of any production.*

*A grammar G is of type 2 **(context-free grammar)** if all of its productions are of the form $A \to \beta$, where $A \in N$, $\beta \in (N \cup T)^+$. A production of the form $S \to \varepsilon$ can also be accepted if the start symbol S does not occur in the right-hand side of any production.*

*A grammar G is of type 3 **(regular grammar)** if its productions are of the form $A \to aB$ or $A \to a$, where $a \in T$ and $A, B \in N$. A production of the form $S \to \varepsilon$ can also be accepted if the start symbol S does not occur in the right-hand side of any production.*

If a grammar G is of type i then language $L(G)$ is also of type i.

This classification was introduced by Noam Chomsky.

A language L is of type i ($i = 0, 1, 2, 3$) if there exists a grammar G of type i which generates the language L, so $L = L(G)$.

Denote by $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) the class of the languages of type i. Can be proved that

$$\mathcal{L}_0 \supset \mathcal{L}_1 \supset \mathcal{L}_2 \supset \mathcal{L}_3 .$$

By the definition of different type of languages, the inclusions ($\supseteq$) are evident, but the strict inclusions ($\supset$) must be proved.

Example 2.3 We give an example for each type of context-sensitive, context-free and regular grammars.

Context-sensitive grammar. $G_1 = (N_1, T_1, P_1, S_1)$, where $N_1 = \{S_1, A, B, C\}$, $T_1 = \{a, 0, 1\}$.

Elements of P_1 are:

$$\begin{aligned} S_1 &\to ACA, \\ AC &\to AACA \mid ABa \mid AaB, \\ B &\to AB \mid A, \\ A &\to 0 \mid 1. \end{aligned}$$

Language $L(G_1)$ contains words of the form uav with $u, v \in \{0, 1\}^*$ and $|u| \neq |v|$.

Context-free grammar. $G_2 = (N_2, T_2, P_2, S)$, where $N_2 = \{S, A, B\}$, $T_2 = \{+, *, (,), a\}$.

Elements of P_2 are:

$S \rightarrow S + A \mid A,$
$A \rightarrow A * B \mid B,$
$B \rightarrow (S) \mid a.$

Language $L(G_2)$ contains algebraic expressions which can be correctly built using letter a, operators + and $*$ and brackets.

Regular grammar. $G_3 = (N_3, T_3, P_3, S_3)$, where $N_3 = \{S_3, A, B\}$, $T_3 = \{a, b\}$.

Elements of P_3 are:

$S_3 \rightarrow aA$
$A \rightarrow aB \mid a$
$B \rightarrow aB \mid bB \mid a \mid b.$

Language $L(G_3)$ contains words over the alphabet $\{a, b\}$ with at least two letters a at the beginning.

It is easy to prove that any finite language is regular. The productions will be done to generate all words of the language. For example, if $u = a_1 a_2 \ldots a_n$ is in the language, then we introduce the productions: $S \rightarrow a_1 A_1$, $A_1 \rightarrow a_2 A_2$, $\ldots A_{n-2} \rightarrow a_{n-1} A_{n-1}$, $A_{n-1} \rightarrow a_n$, where S is the start symbol of the language and $A_1, \ldots, A_{n-1}$ are distinct nonterminals. We define such productions for all words of the language using different nonterminals for different words, excepting the start symbol S. If the empty word is also an element of the language, then the production $S \rightarrow \varepsilon$ is also considered.

The empty set is also a regular language, because the regular grammar $G = (\{S\}, \{a\}, \{S \rightarrow aS\}, S)$ generates it.

Eliminating unit productions

A production of the form $A \rightarrow B$ is called a ***unit production***, where $A, B \in N$. Unit productions can be eliminated from a grammar in such a way that the new grammar will be of the same type and equivalent to the first one.

Let $G = (N, T, P, S)$ be a grammar with unit productions. Define an equivalent grammar $G' = (N, T, P', S)$ without unit productions. The following algorithm will construct the equivalent grammar.

ELIMINATE-UNIT-PRODUCTIONS(G,G')

1 if the unit productions $A \rightarrow B$ and $B \rightarrow C$ are in P put also
the unit production $A \rightarrow C$ in P while P can be extended,
2 if the unit production $A \rightarrow B$ and the production $B \rightarrow \alpha$ ($\alpha \notin N$) are in P
put also the production $A \rightarrow \alpha$ in P,
3 let P' be the set of productions of P except unit productions.

Clearly, G and G' are equivalent. If G is of type $i \in \{0, 1, 2, 3\}$ then G' is also of type i.

Example 2.4 Use the above algorithm in the case of the grammar $G = (\{S, A, B, C\}, \{a, b\}, P, S)$, where P contains

$S \rightarrow A$, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow B$, $D \rightarrow C$,
$S \rightarrow B$, $A \rightarrow D$, $C \rightarrow Aa$,
$A \rightarrow aB$,
$A \rightarrow b.$

Using the first step of the algorithm, we get the following new unit productions:

$S \to D$ (because of $S \to A$ and $A \to D$),
$S \to C$ (because of $S \to B$ and $B \to C$),
$A \to C$ (because of $A \to B$ and $B \to C$),
$B \to B$ (because of $B \to C$ and $C \to B$),
$C \to C$ (because of $C \to B$ and $B \to C$),
$D \to B$ (because of $D \to C$ and $C \to B$).

In the second step of the algorithm will be considered only productions with A or C in the right-hand side, since productions $A \to aB$, $A \to b$ and $C \to Aa$ can be used (the other productions are all unit productions). We get the following new productions:

$S \to aB$ (because of $S \to A$ and $A \to aB$),
$S \to b$ (because of $S \to A$ and $A \to b$),
$S \to Aa$ (because of $S \to C$ and $C \to Aa$),
$A \to Aa$ (because of $A \to C$ and $C \to Aa$),
$B \to Aa$ (because of $B \to C$ and $C \to Aa$).

The new grammar $G' = (\{S, A, B, C\}, \{a, b\}, P', S)$ will have the productions:

$S \to b$, $A \to b$, $B \to Aa$, $C \to Aa$,
$S \to aB$, $A \to aB$,
$S \to Aa$ $A \to Aa$,

Grammars in normal forms

A grammar is to be said a ***grammar in normal form*** if its productions have no terminal symbols in the left-hand side.

We need the following notions. For alphabets Σ_1 and Σ_2 a *homomorphism* is a function $h : \Sigma_1^* \to \Sigma_2^*$ for which $h(u_1u_2) = h(u_1)h(u_2)$, $\forall u_1, u_2 \in \Sigma_1^*$. It is easy to see that for arbitrary $u = a_1a_2 \dots a_n \in \Sigma_1^*$ value $h(u)$ is uniquely determined by the restriction of h on Σ_1, because $h(u) = h(a_1)h(a_2) \dots h(a_n)$.

If a homomorphism h is a bijection then h an *isomorphism*.

Theorem 2.6 *To any grammar an equivalent grammar in normal form can be associated.*

Proof.

Grammars of type 2 and 3 have in left-hand side of any productions only a nonterminal, so they are in normal form. The proof has to be done for grammars of type 0 and 1 only.

Let $G = (N, T, P, S)$ be the original grammar and we define the grammar in normal form as $G' = (N', T, P', S)$.

Let $a_1, a_2, \dots, a_k$ be those terminal symbols which occur in the left-hand side of productions. We introduce the new nonterminals $A_1, A_2, \dots, A_k$. The following notation will be used: $T_1 = \{a_1, a_2, \dots, a_k\}$, $T_2 = T \setminus T_1$, $N_1 = \{A_1, A_2, \dots, A_k\}$ and $N' = N \cup N_1$.

Define the isomorphism $h : N \cup T \longrightarrow N' \cup T_2$, where

$$h(a_i) = A_i, \quad \text{if} \quad a_i \in T_1,$$
$$h(X) = X, \quad \text{if} \quad X \in N \cup T_2$$

Define the set P' of production as

$$P' = \left\{ h(\alpha) \to h(\beta) \;\middle|\; (\alpha \to \beta) \in P \right\} \cup \left\{ A_i \longrightarrow a_i \;\middle|\; i = 1, 2, \dots, k \right\}$$

In this case $\alpha \overset{*}{\underset{G}{\Longrightarrow}} \beta$ if and only if $h(\alpha) \overset{*}{\underset{G'}{\Longrightarrow}} h(\beta)$. From this the theorem immediately

results because $S \overset{*}{\underset{G}{\Longrightarrow}} u \Leftrightarrow S = h(S) \overset{*}{\underset{G'}{\Longrightarrow}} h(u) = u$. ■

Example 2.5 Let $G = (\{S, D, E\}, \{a, b, c, d, e\}, P, S)$, where P contains

$S \to aebc \mid aDbc$
$Db \to bD$
$Dc \to Ebccd$
$bE \to Eb$
$aE \to aaD \mid aae$

In the left-hand side of productions the terminals a, b, c occur, therefore consider the new nonterminals A, B, C, and include in P' also the new productions $A \to a$, $B \to b$ and $C \to c$.

Terminals a, b, c will be replaced by nonterminals A, B, C respectively, and we get the set P' as

$S \to AeBC \mid ADBC$
$DB \to BD$
$DC \to EBCCd$
$BE \to EB$
$AE \to AAD \mid AAe$
$A \to a$
$B \to b$
$C \to c$.

Let us see what words can be generated by this grammars. It is easy to see that $aebc \in L(G')$, because $S \Longrightarrow AeBC \overset{*}{\Longrightarrow} aebc$.

$S \Longrightarrow ADBC \Longrightarrow ABDC \Longrightarrow ABEBCCd \Longrightarrow AEBBCCd \Longrightarrow AAeBBCCd \overset{*}{\Longrightarrow} aaebbccd$, so $aaebbccd \in L(G')$.

We prove, using the mathematical induction, that $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1}$ for $n \geq 2$. For $n = 2$ this is the case, as we have seen before. Continuing the derivation we get $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AADB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nB^nDC(Cd)^{n-1} \Longrightarrow A^nB^nEBCCd(Cd)^{n-1} \overset{*}{\Longrightarrow} A^nEB^{n+1}CCd(Cd)^{n-1} = A^nEB^{n+1}C(Cd)^n$, and this is what we had to prove.

But $S \overset{*}{\Longrightarrow} A^{n-1}EB^nC(Cd)^{n-1} \Longrightarrow A^{n-2}AAeB^nC(Cd)^{n-1} \overset{*}{\Longrightarrow} a^neb^nc(cd)^{n-1}$. So $a^neb^nc(cd)^{n-1} \in L(G')$, $n \geq 1$. These words can be generated also in G.

2.1.4. Extended grammars

In this subsection extended grammars of type 1, 2 and 3 will be presented.

Extended grammar of type 1. All productions are of the form $\alpha \to \beta$, where $|\alpha| \leq |\beta|$, excepted possibly the production $S \to \varepsilon$.

Extended grammar of type 2. All productions are of the form $A \to \beta$, where $A \in N, \beta \in (N \cup T)^*$.

Extended grammar of type 3. All productions are of the form $A \to uB$ or $A \to u$, Where $A, B \in N, u \in T^*$.

Theorem 2.7 *To any extended grammar an equivalent grammar of the same type can be associated.*

Proof. Denote by G_{ext} the extended grammar and by G the corresponding equivalent grammar of the same type.

Type 1. Define the productions of grammar G by rewriting the productions $\alpha \to \beta$, where $|\alpha| \leq |\beta|$ of the extended grammar G_{ext} in the form $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$ allowed in the case of grammar G by the following way.

Let $X_1X_2\ldots X_m \to Y_1Y_2\ldots Y_n$ $(m \leq n)$ be a production of G_{ext}, which is not in the required form. Add to the set of productions of G the following productions, where $A_1, A_2, \ldots, A_m$ are new nonterminals:

$$\begin{array}{lcl}
X_1X_2\ldots X_m & \to & A_1X_2X_3\ldots X_m \\
A_1X_2\ldots X_m & \to & A_1A_2X_3\ldots X_m \\
 & \ldots & \\
A_1A_2\ldots A_{m-1}X_m & \to & A_1A_2\ldots A_{m-1}A_m \\
A_1A_2\ldots A_{m-1}A_m & \to & Y_1A_2\ldots A_{m-1}A_m \\
Y_1A_2\ldots A_{m-1}A_m & \to & Y_1Y_2\ldots A_{m-1}A_m \\
 & \ldots & \\
Y_1Y_2\ldots Y_{m-2}A_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-2}Y_{m-1}A_m \\
Y_1Y_2\ldots Y_{m-1}A_m & \to & Y_1Y_2\ldots Y_{m-1}Y_mY_{m+1}\ldots Y_n.
\end{array}$$

Furthermore, add to the set of productions of G without any modification the productions of G_{ext} which are of permitted form, i.e. $\gamma_1\delta\gamma_2 \to \gamma_1\gamma\gamma_2$.

Inclusion $L(G_{ext}) \subseteq L(G)$ can be proved because each used production of G_{ext} in a derivation can be simulated by productions G obtained from it. Furthermore, since the productions of G can be used only in the prescribed order, we could not obtain other words, so $L(G) \subseteq L(G_{ext})$ also is true.

Type 2. Let $G_{ext} = (N, T, P, S)$. Productions of form $A \to \varepsilon$ have to be eliminated, only $S \to \varepsilon$ can remain, if S doesn't occur in the right-hand side of productions. For this define the following sets:

$U_0 = \{A \in N \mid (A \to \varepsilon) \in P\}$

$U_i = U_{i-1} \cup \{A \in N \mid (A \to w) \in P,\ w \in U_{i-1}^+\}$.

Since for $i \geq 1$ we have $U_{i-1} \subseteq U_i$, $U_i \subseteq N$ and N is a finite set, there must exists such a k for which $U_{k-1} = U_k$. Let us denote this set as U. It is easy to see that a nonterminal A is in U if and only if $A \overset{*}{\Longrightarrow} \varepsilon$. (In addition $\varepsilon \in L(G_{ext})$ if and only if $S \in U$.)

We define the productions of G starting from the productions of G_{ext} in the following way. For each production $A \to \alpha$ with $\alpha \neq \varepsilon$ of G_{ext} add to the set of productions of G this one and all productions which can be obtained from it by eliminating from α one or more nonterminals which are in U, but only in the case when the right-hand side does not become ε.

It in not difficult to see that this grammar G generates the same language as G_{ext} does, except the empty word ε. So, if $\varepsilon \notin L(G_{ext})$ then the proof is finished. But if $\varepsilon \in L(G_{ext})$, then there are two cases. If the start symbol S does not occur in any right-hand side of productions, then by introducing the production $S \to \varepsilon$, grammar G will generate also the empty word. If S occurs in a production in the right-hand side, then we introduce a new start symbol S' and the new productions $S' \to S$ and $S' \to \varepsilon$. Now the empty word ε can also be generated by grammar G.

Type 3. First we use for G_{ext} the procedure defined for grammars of type 2 to eliminate productions of the form $A \to \varepsilon$. From the obtained grammar we eliminate the unit productions using the algorithm ELIMINATE-UNIT-PRODUCTIONS (see page 67).

In the obtained grammar for each production $A \to a_1a_2\ldots a_nB$, where $B \in N \cup \{\varepsilon\}$, add to the productions of G also the followings

$$\begin{aligned} A &\to a_1A_1,\\ A_1 &\to a_2A_2,\\ &\dots\\ A_{n-1} &\to a_nB, \end{aligned}$$

where $A_1, A_2, \dots, A_{n-1}$ are new nonterminals. It is easy to prove that grammar G built in this way is equivalent to G_{ext}. ■

Example 2.6 Let $G_{ext} = (N, T, P, S)$ be an extended grammar of type 1, where $N = \{S, B, C\}$, $T = \{a, b, c\}$ and P contains the following productions:

$$\begin{aligned} S &\to aSBC \mid aBC & CB &\to BC\\ aB &\to ab & bB &\to bb\\ bC &\to bc & cC &\to cc\ . \end{aligned}$$

The only production which is not context-sensitive is $CB \to BC$. Using the method given in the proof, we introduce the productions:

$$\begin{aligned} CB &\to AB\\ AB &\to AD\\ AD &\to BD\\ BD &\to BC \end{aligned}$$

Now the grammar $G = (\{S, A, B, C, D\}, \{a, b, c\}, P', S)$ is context-sensitive, where the elements of P' are

$$\begin{aligned} S &\to aSBC \mid aBC\\ CB &\to AB & aB &\to ab\\ AB &\to AD & bB &\to bb\\ AD &\to BD & bC &\to bc\\ BD &\to BC & cC &\to cc. \end{aligned}$$

It can be proved that $L(G_{ext}) = L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

Example 2.7 Let $G_{ext} = (\{S, B, C\}, \{a, b, c\}, P, S)$ be an extended grammar of type 2, where P contains:

$$\begin{aligned} S &\to aSc \mid B\\ B &\to bB \mid C\\ C &\to Cc \mid \varepsilon. \end{aligned}$$

Then $U_0 = \{C\}$, $U_1 = \{B, C\}$, $U_3 = \{S, B, C\} = U$. The productions of the new grammar are:

$$\begin{aligned} S &\to aSc \mid ac \mid B\\ B &\to bB \mid b \mid C\\ C &\to Cc \mid c. \end{aligned}$$

The original grammar generates also the empty word and because S occurs in the right-hand side of a production, a new start symbol and two new productions will be defined: $S' \to S, S' \to \varepsilon$. The context-free grammar equivalent to the original grammar is $G = (\{S', S, B, C\}, \{a, b, c\}, P', S')$ with the productions:

$$\begin{aligned} S' &\to S \mid \varepsilon\\ S &\to aSc \mid ac \mid B\\ B &\to bB \mid b \mid C\\ C &\to Cc \mid c. \end{aligned}$$

Both of these grammars generate language $\{a^m b^n c^p \mid p \geq m \geq 0, n \geq 0\}$.

Example 2.8 Let $G_{ext} = (\{S, A, B\}, \{a, b\}, P, S)$ be the extended grammar of type 3 under examination, where P:

$$\begin{aligned} S &\to abA\\ A &\to bB\\ B &\to S \mid \varepsilon. \end{aligned}$$

First, we eliminate production $B \to \varepsilon$. Since $U_0 = U = \{B\}$, the productions will be

$S \to abA$
$A \to bB \mid b$
$B \to S$.

The latter production (which a unit production) can also be eliminated, by replacing it with $B \to abA$. Productions $S \to abA$ and $B \to abA$ have to be transformed. Since, both productions have the same right-hand side, it is enough to introduce only one new nonterminal and to use the productions $S \to aC$ and $C \to bA$ instead of $S \to abA$. Production $B \to abA$ will be replaced by $B \to aC$. The new grammar is $G = (\{S, A, B, C\}, \{a, b\}, P', S)$, where P':

$S \to aC$
$A \to bB \mid b$
$B \to aC$
$C \to bA$.

Can be proved that $L(G_{ext}) = L(G) = \{(abb)^n \mid n \geq 1\}$.

2.1.5. Closure properties in the Chomsky-classes

We will prove the following theorem, by which the Chomsky-classes of languages are closed under the regular operations, that is, the union and product of two languages of type i is also of type i, the iteration of a language of type i is also of type i ($i = 0, 1, 2, 3$).

Theorem 2.8 *The class $\mathcal{L}_i$ ($i = 0, 1, 2, 3$) of languages is closed under the regular operations.*

Proof. For the proof we will use extended grammars. Consider the extended grammars $G_1 = (N_1, T_1, P_1, S_1)$ and $G_2 = (N_2, T_2, P_2, S_2)$ of type i each. We can suppose that $N_1 \cap N_2 = \emptyset$.

Union. Let $G_\cup = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$.

We will show that $L(G_\cup) = L(G_1) \cup L(G_2)$. If $i = 0, 2, 3$ then from the assumption that G_1 and G_2 are of type i follows by definition that $G_\cup$ also is of type i. If $i = 1$ and one of the grammars generates the empty word, then we eliminate from $G_\cup$ the corresponding production (possibly the both) $S_k \to \varepsilon$ ($k = 1, 2$) and replace it by production $S \to \varepsilon$.

Product. Let $G_\times = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$.

We will show that $L(G_\times) = L(G_1)L(G_2)$. By definition, if $i = 0, 2$ then $G_\times$ will be of the same type. If $i = 1$ and there is production $S_1 \to \varepsilon$ in P_1 but there is no production $S_2 \to \varepsilon$ in P_2 then production $S_1 \to \varepsilon$ will be replaced by $S \to S_2$. We will proceed the same way in the symmetrical case. If there is in P_1 production $S_1 \to \varepsilon$ and in P_2 production $S_2 \to \varepsilon$ then they will be replaced by $S \to \varepsilon$.

In the case of regular grammars ($i = 3$), because $S \to S_1 S_2$ is not a regular production, we need to use another grammar $G_\times = (N_1 \cup N_2, T_1 \cup T_2, P_1' \cup P_2, S_1)$, where the difference between P_1' and P_1 lies in that instead of productions in the form $A \to u, u \in T^*$ in P_1' will exist production of the form $A \to uS_2$.

Iteration. Let $G_* = (N_1 \cup \{S\}, T_1, P, S)$.

In the case of grammars of type 2 let $P = P_1 \cup \{S \to S_1 S,\ S \to \varepsilon\}$. Then G_* also is of type 2.

In the case of grammars of type 3, as in the case of product, we will change the productions, that is $P = P_1' \cup \{S \to S_1,\ S \to \varepsilon\}$, where the difference between P_1' and P_1 lies in that for each $A \to u$ ($u \in T^*$) will be replaced by $A \to uS$, and the others will be not changed. Then G_* also will be of type 3.

The productions given in the case of type 2 are not valid for $i = 0, 1$, because when applying production $S \to S_1 S$ we can get the derivations of type $S \overset{*}{\Longrightarrow} S_1 S_1$, $S_1 \overset{*}{\Longrightarrow} \alpha_1\beta_1$, $S_1 \overset{*}{\Longrightarrow} \alpha_2\beta_2$, where $\beta_1\alpha_2$ can be a left-hand side of a production. In this case, replacing $\beta_1\alpha_2$ by its right-hand side in derivation $S \overset{*}{\Longrightarrow} \alpha_1\beta_1\alpha_2\beta_2$, we can generate a word which is not in the iterated language. To avoid such situations, first let us assume that the language is in normal form, i.e. the left-hand side of productions does not contain terminals (*see* page 68), second we introduce a new nonterminal S', so the set of nonterminals now is $N_1 \cup \{S, S'\}$, and the productions are the following:

$$P = P_1 \cup \{S \to \varepsilon,\ S \to S_1 S'\} \cup \{aS' \to aS \mid a \in T_1\}\ .$$

Now we can avoid situations in which the left-hand side of a production can extend over the limits of words in a derivation because of the iteration. The above derivations can be used only by beginning with $S \Longrightarrow S_1 S'$ and getting derivation $S \overset{*}{\Longrightarrow} \alpha_1\beta_1 S'$. Here we can not replace S' unless the last symbol in β_1 is a terminal symbol, and only after using a production of the form $aS' \to aS$.

It is easy to show that $L(G_*) = L(G_1)^*$ for each type. ■

Exercises

2.1-1 Give a grammar which generates language $L = \{uu^{-1} \mid u \in \{a, b\}^*\}$ and determine its type.

2.1-2 Let $G = (N, T, P, S)$ be an extended context-free grammar, where

$N = \{S, A, C, D\}$, $T = \{a, b, c, d, e\}$,

$P = \{S \to abCADe,\ C \to cC,\ C \to \varepsilon,\ D \to dD,\ D \to \varepsilon,\ A \to \varepsilon,\ A \to dDcCA\}$.

Give an equivalent context-free grammar.

2.1-3 Show that Σ^* and Σ^+ are regular languages over arbitrary alphabet Σ.

2.1-4 Give a grammar to generate language $L = \{u \in \{0, 1\}^* \mid n_0(u) = n_1(u)\}$, where $n_0(u)$ represents the number of 0's in word u and $n_1(u)$ the number of 1's.

2.1-5 Give a grammar to generate all natural numbers.

2.1-6 Give a grammar to generate the following languages, respectively:

$L_1 = \{a^n b^m c^p \mid n \geq 1, m \geq 1, p \geq 1\}$,

$L_2 = \{a^{2n} \mid n \geq 1\}$,

$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\}$,

$L_4 = \{a^n b^m \mid n \geq m \geq 1\}$.

2.1-7 Let $G = (N, T, P, S)$ be an extended grammar, where $N = \{S, A, B, C\}$, $T = \{a\}$ and P contains the productions:

$$S \to BAB,\ BA \to BC,\ CA \to AAC,\ CB \to AAB,\ A \to a,\ B \to \varepsilon\ .$$

Determine the type of this grammar. Give an equivalent, not extended grammar with the same type. What language it generates?

2.2. Finite automata and regular languages

Finite automata are computing models with input tape and a finite set of states (Fig. 2.1). Among the states some are called initial and some final. At the beginning the automaton read the first letter of the input word written on the input tape. Beginning with an initial state, the automaton read the letters of the input word one after another while change its

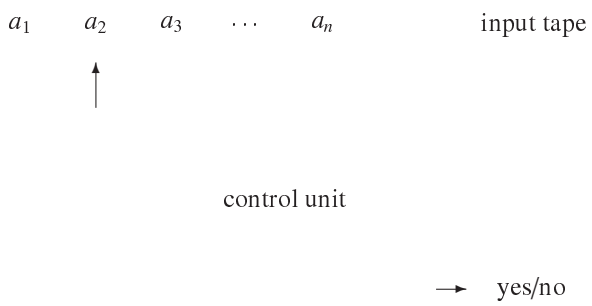


Figure 2.1. Finite automaton.

states, and when after reading the last input letter the current state is a final one, we say that the automaton accepts the given word. The set of words accepted by such an automaton is called the language accepted (recognized) by the automaton.

Definition 2.9 *A **nondeterministic finite automaton** (NFA) is a system* $\mathrm{A} = (Q, \Sigma, E, I, F)$, *where*

- Q *is a finite, nonempty set of **states**,*
- Σ *is the **input alphabet**,*
- E *is the set of **transitions** (or of edges), where* $E \subseteq Q \times \Sigma \times Q$,
- $I \subseteq Q$ *is the set of **initial states**,*
- $F \subseteq Q$ *is the set of **final states**.*

An NFA is in fact a directed, labelled graph, whose vertices are the states and there is a (directed) edge labelled with a from vertex p to vertex q if $(p, a, q) \in E$. Among vertices some are initial and some final states. Initial states are marked by a small arrow entering the corresponding vertex, while the final states are marked with double circles. If two vertices are joined by two edges with the same direction then these can be replaced by only one edge labelled with two letters. This graph can be called a transition graph.

Example 2.9 Let $\mathrm{A} = (Q, \Sigma, E, I, F)$, where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1, 2\}$,
$E = \{(q_0, 0, q_0), (q_0, 1, q_1), (q_0, 2, q_2),$
$(q_1, 0, q_1), (q_1, 1, q_2), (q_1, 2, q_0),$
$(q_2, 0, q_2), (q_2, 1, q_0), (q_2, 2, q_1)\}$
$I = \{q_0\}$, $F = \{q_0\}$.
The automaton can be seen in Fig. 2.2.

In the case of an edge (p, a, q) vertex p is the start-vertex, q the end-vertex and a the label. Now define the notion of the ***walk*** as in the case of graphs. A sequence

$$(q_0, a_1, q_1), (q_1, a_2, q_2), \ldots, (q_{n-2}, a_{n-1}, q_{n-1}), (q_{n-1}, a_n, q_n)$$

of edges of a NFA is a walk with the label $a_1 a_2 \ldots a_n$. If $n = 0$ then $q_0 = q_n$ and $a_1 a_2 \ldots a_n = \varepsilon$. Such a walk is called an ***empty walk***. For a walk the notation

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n \ ,$$

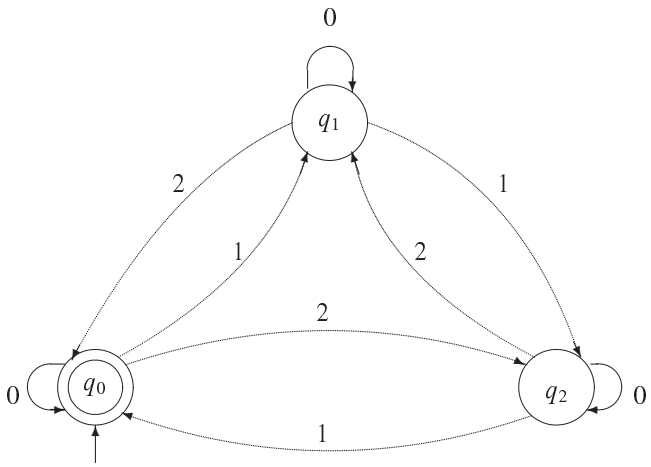


Figure 2.2. The finite automaton of Example 2.9..

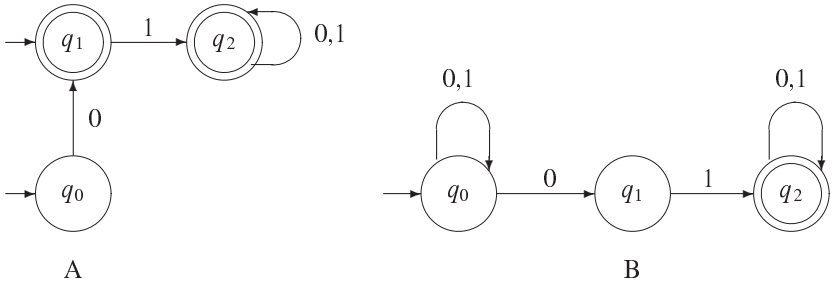


Figure 2.3. Nondeterministic finite automata.

will be used, or if $w = a_1 a_2 \dots a_n$ then we write shortly $q_0 \xrightarrow{w} q_n$. Here q_0 is the start-vertex and q_n the end-vertex of the walk. The states in a walk are not necessary distinct.

A walk is ***productive*** if its start-vertex is an initial state and its end-vertex is a final state. We say that an NFA ***accepts*** or ***recognizes*** a word if this word is the label of a productive walk. The empty word ε is accepted by an NFA if there is an empty productive walk, i.e. there is an initial state which is also a final state.

The set of words accepted by an NFA will be called the language accepted by this NFA. The ***language accepted*** or ***recognized*** by NFA A is

$$L(\mathrm{A}) = \left\{ w \in \Sigma^* \mid \exists p \in I,\ \exists q \in F,\ \exists p \xrightarrow{w} q \right\}.$$

The NFA A_1 and A_2 are ***equivalent*** if $L(\mathrm{A}_1) = L(\mathrm{A}_2)$.

Sometimes it is useful the following ***transition function***:

$$\delta : Q \times \Sigma \to P(Q), \quad \delta(p, a) = \{ q \in Q \mid (p, a, q) \in E \}.$$

δ	0	1
q_0	$\{q_1\}$	$\emptyset$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$

A

δ	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$

B

Figure 2.4. Transition tables of the NFA in Fig. 2.3.

This function associate to a state p and input letter a the set of states in which the automaton can go if its current state is p and the head is on input letter a.

Denote by $|H|$ the cardinal (the number of elements) of H.[2] An NFA is a ***deterministic finite automaton*** (DFA) if

$$|I| = 1 \text{ and } |\delta(q,a)| \leq 1,\ \forall q \in Q,\ \forall a \in \Sigma\ .$$

In Fig. 2.2 a DFA can be seen.

Condition $|\delta(q,a)| \leq 1$ can be replaced by

$$(p,a,q) \in E,\ (p,a,r) \in E \implies q = r\ , \forall p,q,r \in Q, \forall a \in \Sigma\ .$$

If for a DFA $|\delta(q,a)| = 1$ for each state $q \in Q$ and for each letter $a \in \Sigma$ then it is called a ***complete DFA***.

Every DFA can be transformed in a complete DFA by introducing a new state, which can be called a snare state. Let $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ be a DFA. An equivalent and complete DFA will be $\mathrm{A}' = (Q \cup \{s\}, \Sigma, E', \{q_0\}, F)$, where s is the new state and $E' = E \cup \{(p,a,s) \mid \delta(p,a) = \emptyset, p \in Q, a \in \Sigma\} \cup \{(s,a,s) \mid a \in \Sigma\}$. It is easy to see that $L(\mathrm{A}) = L(\mathrm{A}')$.

Using the transition function we can easily define the transition table. The rows of this table are indexed by the elements of Q, its columns by the elements of Σ. At the intersection of row $q \in Q$ and column $a \in \Sigma$ we put $\delta(q,a)$. In the case of Fig. 2.2, the transition table is:

δ	0	1	2
q_0	$\{q_0\}$	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_1\}$	$\{q_2\}$	$\{q_0\}$
q_2	$\{q_2\}$	$\{q_0\}$	$\{q_1\}$

The NFA in 2.3 are not deterministic: the first (automaton A) has two initial states, the second (automaton B) has two transitions with 0 from state q_0 (to states q_0 and q_1). The transition table of these two automata are in Fig. 2.4. $L(\mathrm{A})$ is set of words over $\Sigma = \{0,1\}$ which do not begin with two zeroes (of course ε is in language), $L(\mathrm{B})$ is the set of words which contain 01 as a subword.

Eliminating inaccessible states

Let $\mathrm{A} = (Q, \Sigma, E, I, F)$ be a finite automaton. A state is *accessible* if it is on a walk which

[2]The same notation is used for the cardinal of a set and length of a word, but this is no matter of confusion because for word we use lowercase letters and for set capital letters. The only exception is $\delta(q,a)$, but this could not be confused with a word.

starts by an initial state. The following algorithm determines the inaccessible states building a sequence $U_0, U_1, U_2, \ldots$ of sets, where U_0 is the set of initial states, and for any $i \geq 1$ U_i is the set of accessible states, which are at distance at most i from an initial state.

INACCESSIBLE-STATES(A, U)
$U_0 \leftarrow I$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  for all $q \in U_{i-1}$ **do**
    for all $a \in \Sigma$ **do**
      $U_i \leftarrow U_{i-1} \cup \delta(q, a)$
    endfor
  endfor
until $U_i = U_{i-1}$
$U \leftarrow Q \setminus U_i$
return U

The inaccessible states of the automaton can be eliminated without changing the accepted language.

If $|Q| = n$ and $|\Sigma| = m$ then the running time of the algorithm (the number of steps) in the worst case is $O(n^2m)$, because the number of steps in the two embedded loops is at most nm and in the loop **repeat** at most n.

Set U has the property that $L(\mathrm{A}) \neq \emptyset$ if and only if $U \cap F \neq \emptyset$. The above algorithm can be extended by inserting the $U \cap F \neq \emptyset$ condition to decide if language $L(\mathrm{A})$ is or not empty.

Eliminating nonproductive states

Let $\mathrm{A} = (Q, \Sigma, E, I, F)$ be a finite automaton. A state is *productive* if it is on a walk which ends in a terminal state. For finding the productive states the following algorithm uses the function δ^{-1}:

$$\delta^{-1} : Q \times \Sigma \to \mathcal{P}(Q), \quad \delta^{-1}(p, a) = \{q \mid (q, a, p) \in E\}.$$

This function for a state p and a letter a gives the set of all states from which using this letter a the automaton can go into the state p.

NONPRODUCTIVE-STATESA, V)
$V_0 \leftarrow F$
$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  for all $p \in V_{i-1}$ **do**
    for all $a \in \Sigma$ **do**
      $V_i \leftarrow V_{i-1} \cup \delta^{-1}(p, a)$
    endfor
  endfor
until $V_i = V_{i-1}$
$V \leftarrow Q \setminus V_i$

return V

The nonproductive states of the automaton can be eliminated without changing the accepted language.

If n is the number of states, m the number of letters in the alphabet, then the running time of the algorithm is also $O(n^2m)$ as in the case of the algorithm INACCESSIBLE-STATES.

The set V given by the algorithm has the property that $L(\mathrm{A}) \neq \emptyset$ if and only if $V \cap I \neq \emptyset$. So, by a little modification it can be used to decide if language $L(\mathrm{A})$ is or not empty.

2.2.1. Transforming nondeterministic finite automata in deterministic finite automata

As follows we will show that any NFA can be transformed in an equivalent DFA.

Theorem 2.10 *For any NFA one may construct an equivalent DFA.*

Proof. Let $\mathrm{A} = (Q, \Sigma, E, I, F)$ be an NFA. Define a DFA $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$, where

- $\overline{Q} = \mathcal{P}(Q) \setminus \emptyset$,
- edges of $\overline{E}$ are those triplets (S, a, R) for which $R, S \in \overline{Q}$ are not empty, $a \in \Sigma$ and $R = \bigcup_{p \in S} \delta(p, a)$,
- $\overline{I} = \{I\}$,
- $\overline{F} = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$.

We prove that $L(\mathrm{A}) = L(\overline{\mathrm{A}})$.

a) First prove that $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$. Let $w = a_1 a_2 \ldots a_k \in L(\mathrm{A})$. Then there exists a walk

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \quad q_0 \in I,\ q_k \in F.$$

Using the transition function $\overline{\delta}$ of NFA $\overline{\mathrm{A}}$ we construct the sets $S_0 = \{q_0\}$, $\overline{\delta}(S_0, a_1) = S_1$, $\ldots \overline{\delta}(S_{k-1}, a_k) = S_k$. Then $q_1 \in S_1, \ldots, q_k \in S_k$ and since $q_k \in F$ we get $S_k \cap F \neq \emptyset$, so $S_k \in \overline{F}$. Thus, there exists a walk

$$S_0 \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S_{k-1} \xrightarrow{a_k} S_k, \quad S_0 \subseteq I,\ S_k \in \overline{F}.$$

There are sets $S'_0, \ldots, S'_k$ for which $S'_0 = I$, and for $i = 0, 1, \ldots, k$ we have $S_i \subseteq S'_i$, and

$$S'_0 \xrightarrow{a_1} S'_1 \xrightarrow{a_2} S'_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} S'_{k-1} \xrightarrow{a_k} S'_k$$

is a productive walk. Therefore $w \in L(\overline{\mathrm{A}})$. That is $L(\mathrm{A}) \subseteq L(\overline{\mathrm{A}})$.

b) Now we show that $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. Let $w = a_1 a_2 \ldots a_k \in L(\overline{\mathrm{A}})$. Then there is a walk

$$\overline{q_0} \xrightarrow{a_1} \overline{q_1} \xrightarrow{a_2} \overline{q_2} \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} \overline{q}_{k-1} \xrightarrow{a_k} \overline{q}_k, \quad \overline{q}_0 \in \overline{I},\ \overline{q}_k \in \overline{F}.$$

Using the definition of $\overline{F}$ we have $\overline{q}_k \cap F \neq \emptyset$, i.e. there exists $q_k \in \overline{q}_k \cap F$, that is by the definitions of $q_k \in F$ and $\overline{q}_k$ there is q_{k-1} such that $(q_{k-1}, a_k, q_k) \in E$. Similarly, there are the states $q_{k-2}, \ldots, q_1, q_0$ such that $(q_{k-2}, a_k, q_{k-1}) \in E, \ldots, (q_0, a_1, q_1) \in E$, where $q_0 \in \overline{q}_0 = I$, thus, there is a walk

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{k-1}} q_{k-1} \xrightarrow{a_k} q_k, \quad q_0 \in I,\ q_k \in F,$$

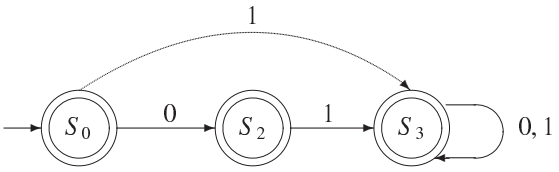


Figure 2.5. The equivalent DFA with NFA A in Fig. 2.3.

so $L(\overline{\mathrm{A}}) \subseteq L(\mathrm{A})$. ■

In constructing DFA we can use the corresponding transition function $\overline{\delta}$:

$$\overline{\delta}(\overline{q}, a) = \left\{ \bigcup_{q \in \overline{q}} \delta(q, a) \right\}, \ \forall \overline{q} \in \overline{Q}, \forall a \in \Sigma.$$

The empty set was excluded from the states, so we used here $\emptyset$ instead of $\{\emptyset\}$.

Example 2.10 Apply Theorem 2.10 to transform NFA A in Fig. 2.3. Introduce the following notation for the states of the DFA:

$S_0 := \{q_0, q_1\}$, $S_1 := \{q_0\}$, $S_2 := \{q_1\}$, $S_3 := \{q_2\}$,
$S_4 := \{q_0, q_2\}$, $S_5 := \{q_1, q_2\}$, $S_6 := \{q_0, q_1, q_2\}$,

where S_0 is the initial state. Using the transition function we get the transition table:

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_1	$\{S_2\}$	$\emptyset$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$
S_4	$\{S_5\}$	$\{S_3\}$
S_5	$\{S_3\}$	$\{S_3\}$
S_6	$\{S_5\}$	$\{S_3\}$

This automaton contains many inaccessible states. By algorithm Inaccessible-states we determine the accessible states of DFA:

$U_0 = \{S_0\}$, $U_1 = \{S_0, S_2, S_3\}$, $U_2 = \{S_0, S_2, S_3\} = U_1 = U$.

Initial state S_0 is also a final state. States S_2 and S_3 are final states. States S_1, S_4, S_5, S_6 are inaccessible and can be removed from the DFA. The transition table of the resulted DFA is

$\overline{\delta}$	0	1
S_0	$\{S_2\}$	$\{S_3\}$
S_2	$\emptyset$	$\{S_3\}$
S_3	$\{S_3\}$	$\{S_3\}$

The corresponding transition graph is in Fig. 2.5.

The algorithm given in Theorem 2.10 can be simplified. It is not necessary to consider all subset of the set of states of NFA. The states of DFA $\overline{\mathrm{A}}$ can be obtained successively.

Begin with the state $\overline{q}_0 = I$ and determine the states $\overline{\delta}(\overline{q}_0, a)$ for all $a \in \Sigma$. For the newly obtained states we determine the states accessible from them. This can be continued until no new states arise.

In our previous example $\overline{q}_0 := \{q_0, q_1\}$ is the initial state. From this we get

$\overline{\delta}(\overline{q}_0, 0) = \{\overline{q_1}\}$, where $\overline{q}_1 := \{q_1\}$, $\quad \overline{\delta}(\overline{q}_0, 1) = \{\overline{q}_2\}$, where $\overline{q}_2 := \{q_2\}$,
$\overline{\delta}(\overline{q}_1, 0) = \emptyset$, $\quad \overline{\delta}(\overline{q}_1, 1) = \{\overline{q}_2\}$,
$\overline{\delta}(\overline{q}_2, 0) = \{\overline{q}_2\}$, $\quad \overline{\delta}(\overline{q}_2, 1) = \{\overline{q}_2\}$.

The transition table is

$\overline{\delta}$	0	1
$\overline{q}_0$	$\{\overline{q}_1\}$	$\{\overline{q}_2\}$
$\overline{q}_1$	$\emptyset$	$\{\overline{q}_2\}$
$\overline{q}_2$	$\{\overline{q}_2\}$	$\{\overline{q}_2\}$

which is the same (excepted the notation) as before.

The next algorithm will construct for an NFA $\mathrm{A} = (Q, \Sigma, E, I, F)$ the transition table M of the equivalent DFA $\overline{\mathrm{A}} = (\overline{Q}, \Sigma, \overline{E}, \overline{I}, \overline{F})$, but without to determine the final states (which can easily be included). Value of $\textsc{IsIn}(\overline{q}, \overline{Q})$ in the algorithm is true if state $\overline{q}$ is already in $\overline{Q}$ and is false otherwise. Let $a_1, a_2, \ldots, a_m$ be an ordered list of the letters of Σ.

$\textsc{Nfa-dfa}(\mathrm{A}, \overline{\mathrm{A}})$

```
 1  q̄_0 ← I
 2  Q̄ ← {q̄_0}
 3  i ← 0                                          ▷ i counts the rows.
 4  k ← 0                                          ▷ k counts the states.
 5  repeat
 6        for j = 1, 2, ..., m                     ▷ j counts the columns.
 7            do q̄ ← ⋃_{p∈q̄_i} δ(p, a_j)
 8               if q̄ ≠ ∅
 9                  then if IsIn(q̄, Q̄)
10                             then M[i, j] ← {q̄}
11                             else k ← k + 1
12                                  q̄_k ← q̄
13                                  M[i, j] ← {q̄_k}
14                                  Q̄ ← Q̄ ∪ {q̄_k}
15                  else M[i, j] ← ∅
16        i ← i + 1
17  until i = k + 1
18  return transition table M of Ā
```

Since loop **repeat** is executed as many times as the number of states of new automaton, in worst case the running time can be exponential, because, if the number of states in NFA is n, then DFA can have even $2^n - 1$ states. (The number of subsets of a set of n elements is 2^n, including the empty set.)

Theorem 2.10 will have it that to any NFA one may construct an equivalent DFA. Conversely, any DFA is also an NFA by definition. So, the nondeterministic finite automata

accepts the same class of languages as the deterministic finite automata.

2.2.2. Equivalence of deterministic finite automata

In this subsection we will use complete deterministic finite automata only. In this case $\delta(q, a)$ has a single element. In formulae, sometimes, instead of set $\delta(q, a)$ we will use its single element. We introduce for a set $A = \{a\}$ the function $elem(A)$ which give us the single element of set A, so $elem(A) = a$. Using walks which begin with the initial state and have the same label in two DFA's we can determine the equivalence of these DFA's. If only one of these walks ends in a final state, then they could not be equivalent.

Consider two DFA's over the same alphabet $A = (Q, \Sigma, E, \{q_0\}, F)$ and $A' = (Q', \Sigma, E', \{q_0'\}, F')$. We are interested to determine if they are or not equivalent. We construct a table with elements of form (q, q'), where $q \in Q$ and $q' \in Q'$. Beginning with the second column of the table, we associate a column to each letter of the alphabet Σ. If the first element of the ith row is (q, q') then at the cross of ith row and the column associated to letter a will be the pair $\Big(elem(\delta(q, a)), elem(\delta'(q', a))\Big)$.

	$\dots$ a $\dots$
$\dots$	$\dots$
(q, q')	$\Big(elem(\delta(q, a)), elem(\delta'(q', a))\Big)$
$\dots$	$\dots$

In the first column of the first row we put (q_0, q_0') and complete the first row using the above method. If in the first row in any column there occur a pair of states from which one is a final state and the other not then the algorithm ends, the two automata are ***not equivalent***. If there is no such a pair of states, every new pair is written in the first column. The algorithm continues with the next unfilled row. If no new pair of states occurs in the table and for each pair both of states are final or both are not, then the algorithm ends and the two DFA are ***equivalent***.

DFA-EQUIVALENCE(A, A′)

```
1  write in the first column of the first row the pair (q_0, q'_0)
2  i ← 0
3  repeat
4        i ← i + 1
5        let (q, q′) be the pair in the first column of the ith row
6        for all a ∈ Σ
7            do write in the column associated to a in the ith row
                the pair (elem(δ(q, a)), elem(δ′(q′, a)))
8               if one state in (elem(δ(q, a)), elem(δ′(q′, a))) is final and the other not
9                  then return NO
10                 else write pair (elem(δ(q, a)), elem(δ′(q′, a))) in the next empty row
                        of the first column, if not occurred already in the first column
11 until the first element of (i + 1)th row becomes empty
12 return YES
```

If $|Q| = n$, $|Q'| = n'$ and $|\Sigma| = m$ then taking into account that in worst case loop **repeat** is executed nn' times, loop **for** m times, the running time of the algorithm in worst case will be $O(nn'm)$, or if $n = n'$ then $O(n^2m)$.

Our algorithm was described to determine the equivalence of two complete DFA's. If we have to determine the equivalence of two NFA's, first we transform them into complete DFA's and after this we can apply the above algorithm.

Example 2.11 Determine if the two DFA's in Fig. 2.6 are equivalent or not. The algorithm gives the table

	a	b
(q_0, p_0)	(q_2, p_3)	(q_1, p_1)
(q_2, p_3)	(q_1, p_2)	(q_2, p_3)
(q_1, p_1)	(q_2, p_3)	(q_0, p_0)
(q_1, p_2)	(q_2, p_3)	(q_0, p_0)

The two DFA's are equivalent because all possible pairs of states are considered and in every pair both

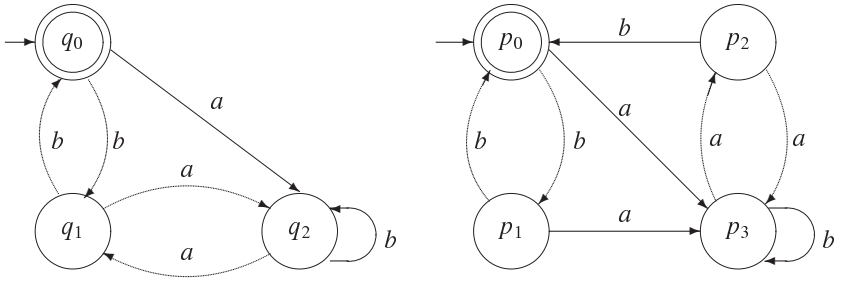


Figure 2.6. Equivalent DFA's (Example 2.11.).

states are final or both are not final.

Example 2.12 The table of the two DFA's in Fig. 2.7 is:

	a	b
(q_0, p_0)	(q_1, p_3)	(q_2, p_1)
(q_1, p_3)	(q_2, p_2)	(q_0, p_3)
(q_2, p_1)		
(q_2, p_2)		

These two DFA's are not equivalent, because in the last column of the second row in the pair (q_0, p_3) the first state is final and the second not.

2.2.3. Equivalence of finite automata and regular languages

We have seen that NFA's accept the same class of languages as DFA's. The following theorem states that this class is that of regular languages.

Theorem 2.11 *If L is a language accepted by a DFA, then one may construct a regular grammar which generates language L.*

Proof. Let $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ be the DFA accepting language L, that is $L = L(\mathrm{A})$. Define the regular grammar $G = (Q, \Sigma, P, q_0)$ with the productions:

- If $(p, a, q) \in E$ for $p, q \in Q$ and $a \in \Sigma$, then put production $p \to aq$ in P.
- If $(p, a, q) \in E$ and $q \in F$, then put also production $p \to a$ in P.

Prove that $L(G) = L(\mathrm{A}) \setminus \{\varepsilon\}$.

Let $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$ and $u \neq \varepsilon$. Thus, since A accepts word u, there is a walk

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n, \quad q_n \in F.$$

Then there are in P the productions

$$q_0 \to a_1 q_1, \quad q_1 \to a_2 q_2, \quad \dots, \quad q_{n-2} \to a_{n-1} q_{n-1}, \quad q_{n-1} \to a_n$$

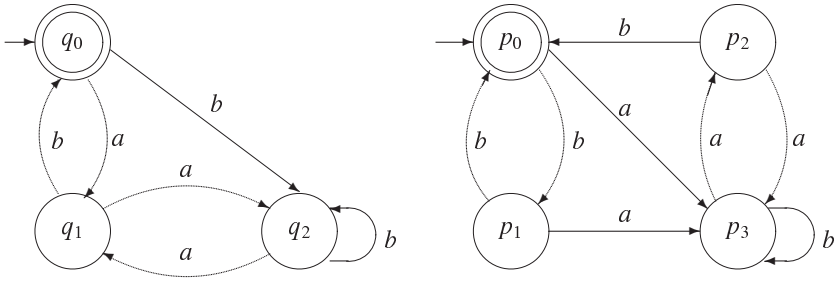


Figure 2.7. Non equivalent DFA's (Example 2.12.).

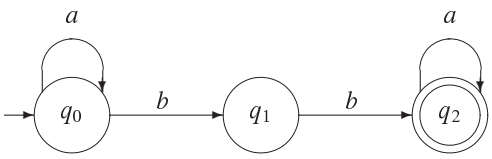


Figure 2.8. DFA of the Example 2.13..

(in the right-hand side of the last production q_n does not occur, because $q_n \in F$), so there is the derivation

$$q_0 \Longrightarrow a_1q_1 \Longrightarrow a_1a_2q_2 \Longrightarrow \ldots \Longrightarrow a_1a_2\ldots a_{n-1}q_{n-1} \Longrightarrow a_1a_2\ldots a_n.$$

Therefore, $u \in L(G)$.

Conversely, let $u = a_1a_2\ldots a_n \in L(G)$ and $u \neq \varepsilon$. Then there exists a derivation

$$q_0 \Longrightarrow a_1q_1 \Longrightarrow a_1a_2q_2 \Longrightarrow \ldots \Longrightarrow a_1a_2\ldots a_{n-1}q_{n-1} \Longrightarrow a_1a_2\ldots a_n,$$

in which productions

$$q_0 \to a_1q_1, \quad q_1 \to a_2q_2, \quad \ldots, \quad q_{n-2} \to a_{n-1}q_{n-1}, \quad q_{n-1} \to a_n$$

were used, which by definition means that in DFA A there is a walk

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n,$$

and since q_n is a final state, $u \in L(\mathrm{A}) \setminus \{\varepsilon\}$.

If the DFA accepts also the empty word ε, then in the above grammar we introduce a new start symbol q_0' instead of q_0, consider the new production $q_0' \to \varepsilon$ and for each production $q_0 \to \alpha$ introduce also $q_0' \to \alpha$. ■

Example 2.13 Let $\mathrm{A} = (\{q_0, q_1, q_2\}, \{a, b\}, E, \{q_0\}, \{q_2\})$ be a DFA, where $E = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_2, a, q_2)\}$. The corresponding transition table is

δ	a	b
q_0	$\{q_0\}$	$\{q_1\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\{q_2\}$	$\emptyset$

The transition graph of A is in Fig. 2.8. By Theorem 2.11 we define regular grammar $G = (\{q_0, q_1, q_2\}, \{a, b\}, P, q_0)$ with the productions in P

$$q_0 \to aq_0 \mid bq_1, \qquad q_1 \to bq_2 \mid b, \qquad q_2 \to aq_2 \mid a.$$

One may prove that $L(\mathrm{A}) = \{a^m bba^n \mid m \geq 0, n \geq 0\}$.

The method described in the proof of Theorem 2.11 easily can be given as an algorithm. The productions of regular grammar $G = (Q, \Sigma, P, q_0)$ obtained from the DFA $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ can be determined by the following algorithm.

REGULAR-GRAMMAR-FROM-DFA(A, G)

```
1  P ← ∅
2  for all p ∈ Q
3      do for all a ∈ Σ
4             do for all q ∈ Q
5                    do if (p, a, q) ∈ E
6                          then P ← P ∪ {p → aq}
7                               if q ∈ F
8                                  then P ← P ∪ {p → a}
9  if q_0 ∈ F
10    then P ← P ∪ {q_0 → ε}
```

It is easy to see that the running time of the algorithm is $\Theta(n^2m)$, if the number of states is n and the number of letter in alphabet is m. In lines 2–4 we can consider only one loop, if we use the elements of E. Then the worst case running time is $\Theta(p)$, where p is the number of transitions of DFA. This is also $O(n^2m)$, since all transitions are possible. This algorithm is:

REGULAR-GRAMMAR-FROM-DFA'(A, G)

```
1  P ← ∅
2  for all (p, a, q) ∈ E
3      do P ← P ∪ {p → aq}
4         if q ∈ F
5            then P ← P ∪ {p → a}
6  if q_0 ∈ F
7      then P ← P ∪ {q_0 → ε}
```

Theorem 2.12 *If $L = L(G)$ is a regular language, then one may construct an NFA that accepts language L.*

Proof. Let $G = (N, T, P, S)$ be the grammar which generates language L. Define NFA $\mathrm{A} = (Q, T, E, \{S\}, F)$:

- $Q = N \cup \{Z\}$, where $Z \notin N \cup T$ (i.e. Z is a new symbol),
- For every production $A \to aB$, define transition (A, a, B) in E.
- For every production $A \to a$, define transition (A, a, Z) in E.
- $F = \begin{cases} \{Z\} & \text{if production } S \to \varepsilon \text{ does not occur in } G, \\ \{Z, S\} & \text{if production } S \to \varepsilon \text{ occurs in } G. \end{cases}$

Prove that $L(G) = L(A)$.
Let $u = a_1a_2 \ldots a_n \in L(G)$, $u \neq \varepsilon$. Then there is in G a derivation of word u:

$$S \Longrightarrow a_1A_1 \Longrightarrow a_1a_2A_2 \Longrightarrow \ldots \Longrightarrow a_1a_2 \ldots a_{n-1}A_{n-1} \Longrightarrow a_1a_2 \ldots a_n.$$

This derivation is based on productions

$$S \to a_1A_1, \quad A_1 \to a_2A_2, \quad \ldots, \quad A_{n-2} \to a_{n-1}A_{n-1}, \quad A_{n-1} \to a_n.$$

Then, by the definition of the transitions of NFA A there exists a walk

$$S \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} Z, \quad Z \in F.$$

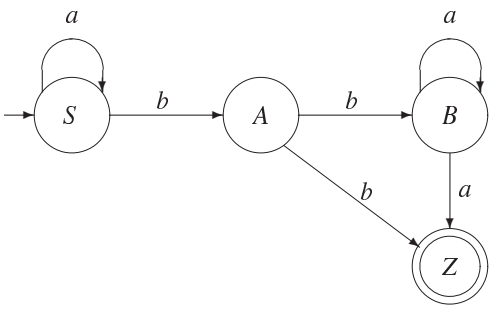


Figure 2.9. NFA associated to grammar in Example 2.14..

Thus, $u \in L(\mathrm{A})$. If $\varepsilon \in L(G)$, there is production $S \to \varepsilon$, but in this case the initial state is also a final one, so $\varepsilon \in L(A)$. Therefore, $L(G) \subseteq L(A)$.

Let now $u = a_1 a_2 \dots a_n \in L(\mathrm{A})$. Then there exists a walk

$$S \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} Z, \quad Z \in F.$$

If u is the empty word, then instead of Z we have in the above formula S, which also is a final state. In other cases only Z can be as last symbol. Thus, in G there exist the productions

$$S \to a_1 A_1, \quad A_1 \to a_2 A_2, \quad \dots, \quad A_{n-2} \to a_{n-1} A_{n-1}, \quad A_{n-1} \to a_n \ ,$$

and there is the derivation

$$S \Longrightarrow a_1 A_1 \Longrightarrow a_1 a_2 A_2 \Longrightarrow \dots \Longrightarrow a_1 a_2 \dots a_{n-1} A_{n-1} \Longrightarrow a_1 a_2 \dots a_n,$$

thus, $u \in L(G)$ and therefore $L(A) \subseteq L(G)$. ■

Example 2.14 Let $G = (\{S, A, B\}, \{a, b\}, \{S \to aS, S \to bA, A \to bB, A \to b, B \to aB, B \to a\}, S)$ be a regular grammar. The NFA associated is $A = (\{S, A, B, Z\}, \{a, b\}, E, S, \{Z\})$, where $E = \{(S, a, S), (S, b, A), (A, b, B), (A, b, Z), (B, a, B), (B, a, Z)\}$. The corresponding transition table is

δ	a	b
S	$\{S\}$	$\{A\}$
A	$\emptyset$	$\{B, Z\}$
B	$\{B, Z\}$	$\emptyset$
E	$\emptyset$	$\emptyset$

The transition graph is in Fig. 2.9. This NFA can be simplified, states B and Z can be contracted in one final state.

Using the above theorem we define an algorithm which associate an NFA $\mathrm{A} = (Q, T, E, \{S\}, F)$ to a regular grammar $G = (N, T, P, S)$

NFA-FROM-REGULAR-GRAMMAR(G, A)

```
1  E ← ∅
2  Q ← N ∪ {Z}
3  for all A ∈ N
4      do for all a ∈ T
5             do if (A → a) ∈ P
6                    then E ← E ∪ {(A, a, Z)}
7                for all B ∈ N
8                    do if (A → aB) ∈ P
9                           then E ← E ∪ {(A, a, B)}
10 if (S → ε) ∉ P
11    then F ← {Z}
12    else F ← {Z, S}
```

As in the case of algorithm REGULAR-GRAMMAR-FROM-DFA, the running time is $\Theta(n^2m)$, where n is number of nonterminals and m the number of terminals. Loops in lines 3, 4 and 7 can be replaced by only one, which uses productions. The running time in this case is better and is equal to $\Theta(p)$, if p is the number of productions. This algorithm is:

NFA-FROM-REGULAR-GRAMMAR'(G, A)

```
1  E ← ∅
2  Q ← N ∪ {Z}
3  for all (A → u) ∈ P
4      do if u = a
5             then E ← E ∪ {(A, a, Z)}
6         if u = aB
7             then E ← E ∪ {(A, a, B)}
8  if (S → ε) ∉ P
9     then F ← {Z}
10    else F ← {Z, S}
```

From theorems 2.10, 2.11 and 2.12 results that the class of regular languages coincides with the class of languages accepted by NFA's and also with class of languages accepted by DFA's. The result of these three theorems is illustrated in Fig. 2.10 and can be summarised also in the following theorem.

Theorem 2.13 *The following three class of languages are the same:*

- *the class of regular languages,*
- *the class of languages accepted by DFA's,*
- *the class of languages accepted by NFA's.*

Operation on regular languages

It is known (see Theorem 2.8) that the set $\mathcal{L}_3$ of regular languages is closed under the regular operations, that is if L_1, L_2 are regular languages, then languages $L_1 \cup L_2$, L_1L_2 and L_1^* are also regular. For regular languages are true also the following statements.

The complement of a regular language is also regular. This is easy to prove using au-

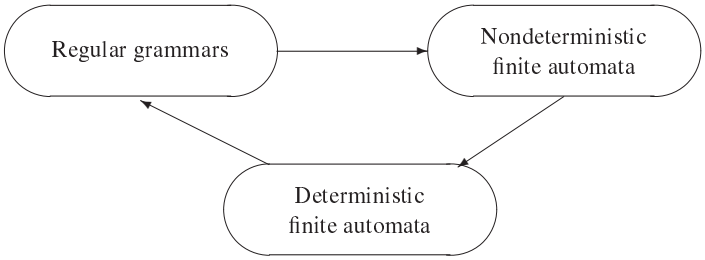


Figure 2.10. Relations between regular grammars and finite automata. To any regular grammar one may construct an NFA which accepts the language generated by that grammar. Any NFA can be transformed in an equivalent DFA. To any DFA one may construct a regular grammar which generates the language accepted by that DFA.

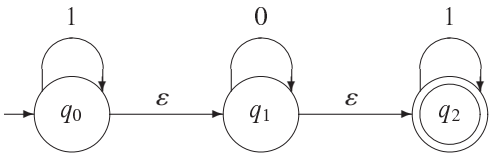


Figure 2.11. Finite automata ε-moves.

tomata. Let L be a regular language and let $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ be a DFA which accepts language L. It is easy to see that the DFA $\overline{\mathrm{A}} = (Q, \Sigma, E, \{q_0\}, Q \setminus F)$ accepts language $\overline{L}$. So, $\overline{L}$ is also regular.

The intersection of two regular languages is also regular. Since $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, the intersection is also regular.

The difference of two regular languages is also regular. Since $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$, the difference is also regular.

2.2.4. Finite automata with ε-moves

A finite automaton with ε-moves (FA with ε-moves) extends NFA in such way that it may have transitions on the empty input ε, i.e. it may change a state without reading any input symbol. In the case of a FA with ε-moves $\mathrm{A} = (Q, \Sigma, E, I, F)$ for the set of transitions it is true that $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

The transition function of a FA with ε-moves is:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q), \quad \delta(p, a) = \{q \in Q \mid (p, a, q) \in E\} .$$

The FA with ε-moves in Fig. 2.11 accepts words of form uvw, where $u \in \{1\}^*$, $v \in \{0\}^*$ and $w \in \{1\}^*$.

Theorem 2.14 *To any FA with ε-moves one may construct an equivalent NFA (without ε-moves).*

Let $\mathrm{A} = (Q, \Sigma, E, I, F)$ be an FA with ε-moves and we construct an equivalent NFA $\overline{\mathrm{A}} = (Q, \Sigma, \overline{E}, I, \overline{F})$. The following algorithm determines sets $\overline{F}$ and $\overline{E}$.

For a state q denote by $\Lambda(q)$ the set of states (including even q) in which one may go

from q using ε-moves only. This may be extended also to sets

$$\Lambda(S) = \bigcup_{q \in S} \Lambda(q), \quad \forall S \subseteq Q \ .$$

Clearly, for all $q \in Q$ and $S \subseteq Q$ both $\Lambda(q)$ and $\Lambda(S)$ may be computed. Suppose in the sequel that these are given.

The following algorithm determine the transitions using the transition function $\overline{\delta}$, which is defined in line 5.

If $|Q| = n$ and $|\Sigma| = m$,, then lines 2–6 show that the running time in worst case is $O(n^2 m)$.

ELIMINATE-EPSILON-MOVES$(\mathrm{A}, \overline{\mathrm{A}})$

1 $\overline{F} \leftarrow F \cup \{q \in I \mid \Lambda(q) \cap F \neq \emptyset\}$
2 **for** all $q \in Q$
3 **do for** all $a \in \Sigma$
4 **do** $\Delta \leftarrow \bigcup_{p \in \Lambda(q)} \delta(p, a)$
5 $\overline{\delta}(q, a) \leftarrow \Delta \cup \left(\bigcup_{p \in \Delta} \Lambda(p) \right)$
6 $\overline{E} \leftarrow \{(p, a, q), \mid p, q \in Q, \ a \in \Sigma, q \in \overline{\delta}(p, a)\}$

Example 2.15 Consider the FA with ε-moves in Fig. 2.11. The corresponding transition table is:

δ	0	1	ε
q_0	$\emptyset$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$	$\emptyset$

Apply algorithm ELIMINATE-EPSILON-MOVES.
$\Lambda(q_0) = \{q_0, q_1, q_2\}$, $\Lambda(q_1) = \{q_1, q_2\}$, $\Lambda(q_2) = \{q_2\}$
$\Lambda(I) = \Lambda(q_0)$, and its intersection with F is not empty, thus $\overline{F} = F \cup \{q_0\} = \{q_0, q_2\}$.
$(q_0, 0)$:
$\Delta = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\}$, $\{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_0, 0) = \{q_1, q_2\}$.
$(q_0, 1)$:
$\Delta = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_0, q_2\}$, $\{q_0, q_2\} \cup (\Lambda(q_0) \cup \Lambda(q_2)) = \{q_0, q_1, q_2\}$
$\overline{\delta}(q_0, 1) = \{q_0, q_1, q_2\}$
$(q_1, 0)$:
$\Delta = \delta(q_1, 0) \cup \delta(q_2, 0) = \{q_1\}$, $\{q_1\} \cup \Lambda(q_1) = \{q_1, q_2\}$
$\overline{\delta}(q_1, 0) = \{q_1, q_2\}$
$(q_1, 1)$:
$\Delta = \delta(q_1, 1) \cup \delta(q_2, 1) = \{q_2\}$, $\{q_2\} \cup \Lambda(q_2) = \{q_2\}$
$\overline{\delta}(q_1, 1) = \{q_2\}$
$(q_1, 1) : \Delta = \delta(q_2, 0) = \emptyset$
$\overline{\delta}(q_2, 0) = \emptyset$

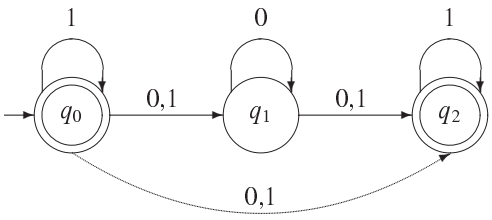


Figure 2.12. NFA equivalent to FA with ε-moves given in Fig. 2.11.

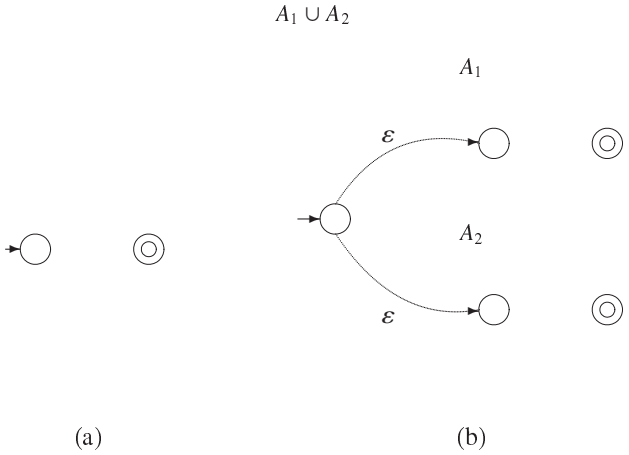


Figure 2.13. **(a)** Representation of an NFA. Initial states are represented by a circle with an arrow, final states by a double circle. **(b)** Union of two NFA's.

$(q_2, 1)$:

$\Delta = \delta(q_2, 1) = \{q_2\}, \quad \{q_2\} \cup \Lambda(q_2) = \{q_2\}$

$\overline{\delta}(q_2, 1) = \{q_2\}.$

The transition table of NFA $\overline{\mathrm{A}}$ is:

$\overline{\delta}$	0	1
q_0	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$
q_1	$\{q_1, q_2\}$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$

and the transition graph is in Fig. 2.12.

Define regular operations on NFA: union, product and iteration. The result will be an FA with ε-moves.

Operation will be given also by diagrams. An NFA is given as in Fig. 2.13(a). Initial states are represented by a circle with an arrow, final states by a double circle.

Let $\mathrm{A}_1 = (Q_1, \Sigma_1, E_1, I_1, F_1)$ and $\mathrm{A}_2 = (Q_2, \Sigma_2, E_2, I_2, F_2)$ be NFA. The result of any operation is a FA with ε-moves $\mathrm{A} = (Q, \Sigma, E, I, F)$. Suppose that $Q_1 \cap Q_2 = \emptyset$ always. If

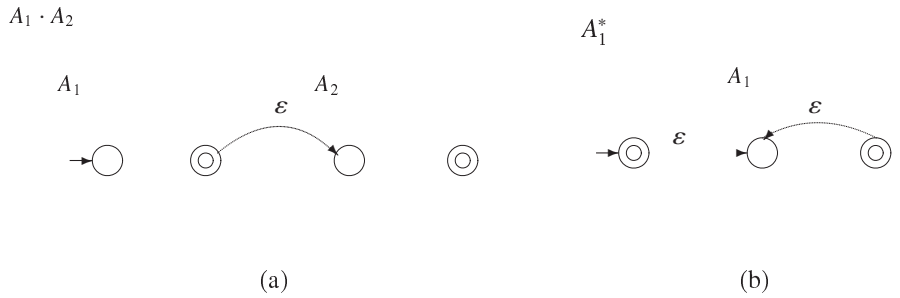


Figure 2.14. **(a)** Product of two FA. **(b)** Iteration of an FA.

not, we can rename the elements of any set of states.

Union. $\mathrm{A} = \mathrm{A}_1 \cup \mathrm{A}_2$, where

$$\begin{aligned} Q &= Q_1 \cup Q_2 \cup \{q_0\}, \\ \Sigma &= \Sigma_1 \cup \Sigma_2, \\ I &= \{q_0\}, \\ F &= F_1 \cup F_2, \\ E &= E_1 \cup E_2 \cup \bigcup_{q \in I_1 \cup I_2} \{(q_0, \varepsilon, q)\}. \end{aligned}$$

For the result of the union see Fig. 2.13(b). The result is the same if instead of a single initial state we choose as set of initial states the union $I_1 \cup I_2$. In this case the result automaton will be without ε-moves. By the definition it is easy to see that $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

Product. $\mathrm{A} = \mathrm{A}_1 \cdot \mathrm{A}_2$, where

$$\begin{aligned} Q &= Q_1 \cup Q_2, \\ \Sigma &= \Sigma_1 \cup \Sigma_2, \\ F &= F_2, \\ I &= I_1, \\ E &= E_1 \cup E_2 \cup \bigcup_{\substack{p \in F_1 \\ q \in I_2}} \{(p, \varepsilon, q)\} \end{aligned}$$

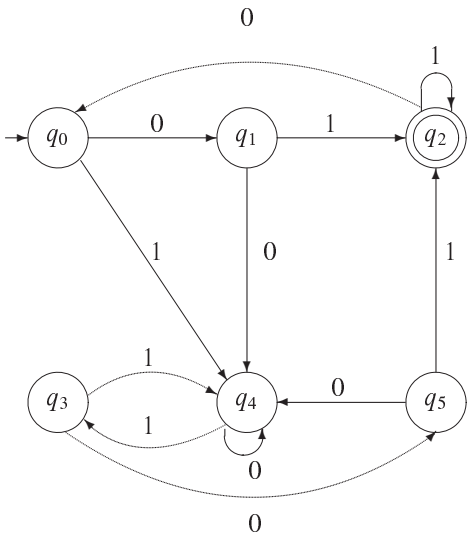
For the result automaton see Fig. 2.14(a). Here also $L(A_1 \cdot A_2) = L(A_1)L(A_2)$.

Iteration. $\mathrm{A} = \mathrm{A}_1{}^*$, where

$$\begin{aligned} Q &= Q_1 \cup \{q_0\}, \\ \Sigma &= \Sigma_1, \\ F &= F_1 \cup \{q_0\}, \\ I &= \{q_0\} \\ E &= E_1 \cup \bigcup_{p \in I_1} \{(q_0, \varepsilon, p)\} \cup \bigcup_{\substack{q \in F_1 \\ p \in I_1}} \{(q, \varepsilon, p)\}\ . \end{aligned}$$

The iteration of an FA can be seen in Fig. 2.14(b). For this operation it is also true that $L(A_1^*) = (L(A_1))^*$.

The definition of these tree operations proves again that regular languages are closed under the regular operations.



				q_5
			q_4	*
		q_3	*	*
	q_2	*	*	*
q_1	*	*	*	
q_0	*	*	*	*

Figure 2.15. Minimization of DFA.

2.2.5. Minimization of finite automata

A DFA $A = (Q, \Sigma, E, \{q_0\}, F)$ is called ***minimum state automaton*** if for any equivalent complete DFA $A' = (Q', \Sigma, E', \{q_0'\}, F')$ it is true that $|Q| \leq |Q'|$. We give an algorithm which builds for any complete DFA an equivalent minimum state automaton.

States p and q of an DFA $A = (Q, \Sigma, E, \{q_0\}, F)$ are ***equivalent*** if for arbitrary word u we reach from both either final or nonfinal states, that is

$$p \equiv q \text{ if for any word } u \in \Sigma^* \begin{cases} p \xrightarrow{u} r,\ r \in F \text{ and } q \xrightarrow{u} s,\ s \in F \text{ or} \\ p \xrightarrow{u} r,\ r \notin F \text{ and } q \xrightarrow{u} s,\ s \notin F\ . \end{cases}$$

If two states are not equivalent, then they are distinguishable. In the following algorithm the distinguishable states will be marked by a star, and equivalent states will be merged. The algorithm will associate list of pair of states with some pair of states expecting a later marking by a star, that is if we mark a pair of states by a star, then all pairs on the associated list will be also marked by a star. The algorithm is given for DFA without inaccessible states. The used DFA is complete, so $\delta(p, a)$ contains exact one element, function *elem* defined on page 81, which gives the unique element of the set, will be also used here.

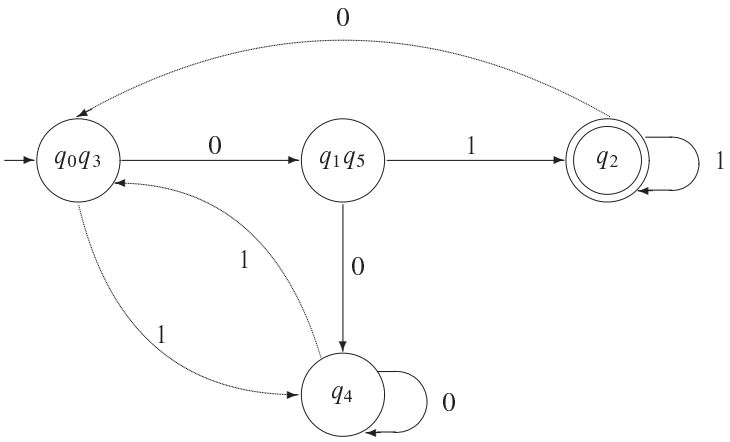


Figure 2.16. Minimum automaton equivalent with DFA in Fig. 2.15.

AUTOMATON-MINIMIZATION(A)

1 mark with a star all pairs of states $\{p, q\}$ for which
$p \in F$ and $q \notin F$ or $p \notin F$ and $q \in F$,
2 associate an empty list with each unmarked pair $\{p, q\}$,
3 **for** all unmarked pair of states $\{p, q\}$ and for all symbol $a \in \Sigma$
examine pairs of states $\{elem(\delta(p, a)), elem(\delta(q, a))\}$,
if any of these pairs is marked,
then mark also pair $\{p, q\}$ with all the elements on the list before
associated with pair $\{p, q\}$,
else, **if** all the above pairs are unmarked,
then put pair $\{p, q\}$ on each list associated with pairs
$\{elem(\delta(p, a)), elem(\delta(q, a))\}$, unless $\delta(p, a) = \delta(q, a)$,
4 merge all unmarked (equivalent) pairs.

After finishing the algorithm, if a cell of the table does not contain a star, then the states corresponding to its row and column index, are equivalent and may be merged. Merging states is continued until it is possible. We can say that the equivalence relation decomposes the set of states in equivalence classes, and the states in such a class may be all merged.

Remark. The above algorithm can be used also in the case of an DFA which is not complete, that is there are states for which does not exist transition. Then a pair $\{\emptyset, \{q\}\}$ may occur, and if q is a final state, consider this pair marked.

Example 2.16 Let be the DFA in Fig. 2.15. We will use a table for marking pairs with a star. Marking pair $\{p, q\}$ means putting a star in the cell corresponding to row p and column q (or row q and column p).

First we mark pairs $\{q_2, q_0\}$, $\{q_2, q_1\}$, $\{q_2, q_3\}$, $\{q_2, q_4\}$ and $\{q_2, q_5\}$ (because q_2 is the single final state). Then consider all unmarked pairs and examine them as the algorithm requires. Let us begin with pair $\{q_0, q_1\}$. Associate with it pairs $\{elem(\delta(q_0, 0)), elem(\delta(q_1, 0))\}, \{elem(\delta(q_0, 1)), elem(\delta(q_1, 1))\}$,

that is $\{q_1, q_4\}$, $\{q_4, q_2\}$. Because pair $\{q_4, q_2\}$ is already marked, mark also pair $\{q_0, q_1\}$.

In the case of pair $\{q_0, q_3\}$ the new pairs are $\{q_1, q_5\}$ and $\{q_4, q_4\}$. With pair $\{q_1, q_5\}$ associate pair $\{q_0, q_3\}$ on a list, that is

$$\{q_1, q_5\} \longrightarrow \{q_0, q_3\} .$$

Now continuing with $\{q_1, q_5\}$ one obtain pairs $\{q_4, q_4\}$ and $\{q_2, q_2\}$, with which nothing are associated by algorithm.

Continue with pair $\{q_0, q_4\}$. The associated pairs are $\{q_1, q_4\}$ and $\{q_4, q_3\}$. None of them are marked, so associate with them on a list pair $\{q_0, q_4\}$, that is

$$\{q_1, q_4\} \longrightarrow \{q_0, q_4\}, \qquad \{q_4, q_3\} \longrightarrow \{q_0, q_4\} .$$

Now continuing with $\{q_1, q_4\}$ we get the pairs $\{q_4, q_4\}$ and $\{q_2, q_3\}$, and because this latter is marked we mark pair $\{q_1, q_4\}$ and also pair $\{q_0, q_4\}$ associated to it on a list. Continuing we will get the table in Fig. 2.15, that is we get that $q_0 \equiv q_3$ and $q_1 \equiv q_5$. After merging them we get an equivalent minimum state automaton (see Fig. 2.16).

2.2.6. Pumping lemma for regular languages

The following theorem, called *pumping lemma* for historical reasons, may be efficiently used to prove that a language is not regular. It is a sufficient condition for a regular language.

Theorem 2.15 (pumping lemma). *For any regular language L there exists a natural number $n \geq 1$ (depending only on L), such that any word u of L with length at least n may be written as $u = xyz$ such that*

(1) $|xy| \leq n$,

(2) $|y| \geq 1$,

(3) $xy^i z \in L$ *for all* $i = 0, 1, 2, \ldots$.

Proof. If L is a regular language, then there is such an DFA which accepts L (by theorems 2.12 and 2.10). Let $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ be this DFA, so $L = L(\mathrm{A})$. Let n be the number of its states, that is $|Q| = n$. Let $u = a_1 a_2 \ldots a_m \in L$ and $m \geq n$. Then, because the automaton accepts word u, there are states $q_0, q_1, \ldots, q_m$ and walk

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{m-1}} q_{m-1} \xrightarrow{a_m} q_m, \ q_m \in F.$$

Because the number of states is n and $m \geq n$, by the pigeonhole principle[3] states $q_0, q_1, \ldots, q_m$ can not all be distinct (see Fig. 2.17), there are at least two of them which are equal. Let $q_j = q_k$, where $j < k$ and k is the least such index. Then $j < k \leq n$. Decompose word u as:

$x = a_1 a_2 \ldots a_j$

$y = a_{j+1} a_{j+2} \ldots a_k$

$z = a_{k+1} a_{k+2} \ldots a_m$.

This decomposition immediately yields to $|xy| \leq n$ and $|y| \geq 1$. We will prove that $xy^i z \in L$ for any i.

Because $u = xyz \in L$, there exists an walk

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_k \xrightarrow{z} q_m, \ q_m \in F,$$

[3] *Pigeonhole principle:* If we have to put more than k objects into k boxes, then at least two boxes will contain at least two objects.

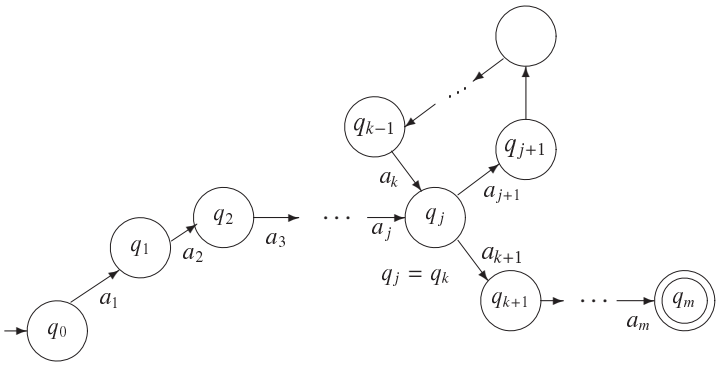


Figure 2.17. Sketch of DFA used in the proof of the pumping lemma.

and because of $q_j = q_k$, this may be written also as

$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ .$$

From this walk $q_j \xrightarrow{y} q_j$ can be omitted or can be inserted many times. So, there are the following walks:

$$q_0 \xrightarrow{x} q_j \xrightarrow{z} q_m,\ q_m \in F\ ,$$
$$q_0 \xrightarrow{x} q_j \xrightarrow{y} q_j \xrightarrow{y} \ldots \xrightarrow{y} q_j \xrightarrow{z} q_m,\ q_m \in F\ .$$

Therefore $xy^iz \in L$ for all i, and this proves the theorem. ■

Example 2.17 We use the pumping lemma to show that $L_1 = \{a^kb^k \mid k \geq 1\}$ is not regular. Assume that L_1 is regular, and let n be the corresponding natural number given by the pumping lemma. Because the length of the word $u = a^nb^n$ is $2n$, this word can be written as in the lemma. We prove that this leads to a contradiction. Let $u = xyz$ be the decomposition as in the lemma. Then $|xy| \leq n$, so x and y can contain no other letters than a, and because we must have $|y| \geq 1$, word y contains at least one a. Then xy^iz for $i \neq 1$ will contain a different number of a's and b's, therefore $xy^iz \notin L_1$ for any $i \neq 1$. This is a contradiction with the third assertion of the lemma, this is why that assumption that L_1 is regular, is false. Therefore $L_1 \notin \mathcal{L}_3$.

Because the context-free grammar $G_1 = (\{S\}, \{a, b\}, \{S \to ab, S \to aSb\}, S)$ generates language L_1, we have $L_1 \in \mathcal{L}_2$. From these two follow that $\mathcal{L}_3 \subset \mathcal{L}_2$.

Example 2.18 We show that $L_2 = \{u \in \{0, 1\}^* \mid n_0(u) = n_1(u)\}$ is not regular. ($n_0(u)$ is the number of 0's in u, while $n_1(u)$ the number of 1's).

We proceed as in the previous example using here word $u = 0^n1^n$, where n is the natural number associated by lemma to language L_2.

Example 2.19 We prove, using the pumping lemma, that $L_3 = \{uu \mid u \in \{a, b\}^*\}$ is not a regular language. Let $w = a^nba^nb = xyz$ be, where n here is also the natural number associated to L_3 by the pumping lemma. From $|xy| \leq n$ we have that y contains no other letters than a, but it contains at least one. By lemma we have $xz \in L_3$, that is not possible. Therefore L_3 is not regular.

Pumping lemma has several interesting consequences.

Corollary 2.16 *Regular language L is not empty if and only if there exists a word $u \in L$, $|u| < n$, where n is the natural number associated to L by the pumping lemma.*

Proof. The assertion in a direction is obvious: if there exists a word shorter than n in L, then $L \neq \emptyset$. Conversely, let $L \neq \emptyset$ and let u be the shortest word in L. We show that $|u| < n$. If $|u| \geq n$, then we apply the pumping lemma, and give the decomposition $u = xyz$, $|y| > 1$ and $xz \in L$. This is a contradiction, because $|xz| < |u|$ and u is the shortest word in L. Therefore $|u| < n$. ■

Corollary 2.17 *There exists an algorithm that can decide if a regular language is or not empty.*

Proof. Assume that $L = L(\mathrm{A})$, where $\mathrm{A} = (Q, \Sigma, E, \{q_0\}, F)$ is a DFA. By consequence 2.16 and theorem 2.15 language L is not empty if and only if it contains a word shorter than n, where n is the number of states of automaton A. By this it is enough to decide that there is a word shorter than n which is accepted by automaton A. Because the number of words shorter than n is finite, the problem can be decided. ■

When we had given an algorithm for inaccessible states of a DFA, we remarked that the procedure can be used also to decide if the language accepted by that automaton is or not empty. Because finite automata accept regular languages, we can consider to have already two procedures to decide if a regular languages is or not empty. Moreover, we have a third procedure, if we take into account that the algorithm for finding productive states also can be used to decide on a regular language when it is empty.

Corollary 2.18 *A regular language L is infinite if and only if there exists a word $u \in L$ such that $n \leq |u| < 2n$, where n is the natural number associated to language L, given by the pumping lemma.*

Proof. If L is infinite, then it contains words longer than $2n$, and let u be the shortest word longer than $2n$ in L. Because L is regular we can use the pumping lemma, so $u = xyz$, where $|xy| \leq n$, thus $|y| \leq n$ is also true. By the lemma $u' = xz \in L$. But because $|u'| < |u|$ and the shortest word in L longer than $2n$ is u, we get $|u'| < 2n$. From $|y| \leq n$ we get also $|u'| \geq n$.

Conversely, if there exists a word $u \in L$ such that $n \leq |u| < 2n$, then using the pumping lemma, we obtain that $u = xyz$, $|y| \geq 1$ and $xy^i z \in L$ for any i, therefore L is infinite. ■

Now, the question is: how can we apply the pumping lemma for a finite regular language, since by pumping words we get an infinite number of words? The number of states of a DFA accepting language L is greater than the length of the longest word in L. So, in L there is no word with length at least n, when n is the natural number associated to L by the pumping lemma. Therefore, no word in L can be decomposed in the form xyz, where $|xyz| \geq n$, $|xy| \leq n$, $|y| \geq 1$, and this is why we can not obtain an infinite number of words in L.

2.2.7. Regular expressions

In this subsection we introduce for any alphabet Σ the notion of regular expressions over Σ and the corresponding representing languages. A regular expression is a formula, and the

$$
\begin{array}{rcl}
x+y & \equiv & y+x \\
(x+y)+z & \equiv & x+(y+z) \\
(xy)z & \equiv & x(yz) \\
(x+y)z & \equiv & xz+yz \\
x(y+z) & \equiv & xy+xz \\
(x+y)^* \equiv (x^*+y)^* & \equiv & (x+y^*)^* \equiv (x^*+y^*)^* \\
(x+y)^* \equiv (xy^*)^* & \equiv & (x^*y)^* \equiv (x^*y^*)^* \\
(x^*)^* & \equiv & x^* \\
x^*x & \equiv & xx^* \\
xx^*+\varepsilon & \equiv & x^*
\end{array}
$$

2.1. Table. Properties of regular expressions.

corresponding language is a language over Σ. For example, if $\Sigma = \{a, b\}$, then a^*, b^*, $a^* + b^*$ are regular expressions over Σ which represent respectively languages $\{a\}^*$, $\{b\}^*$, $\{a\}^* \cup \{b\}^*$. The exact definition is the following.

Definition 2.19 *Define recursively a regular expression over Σ and the language it represent.*

- *$\emptyset$ is a regular expression representing the empty language.*
- *ε is a regular expression representing language $\{\varepsilon\}$.*
- *If $a \in \Sigma$, then a is a regular expression representing language $\{a\}$.*
- *If x, y are regular expressions representing languages X and Y respectively, then $(x + y)$, (xy), (x^*) are regular expressions representing languages $X \cup Y$, XY and X^* respectively.*

Regular expression over Σ can be obtained only by using the above rules a finite number of times.

Some brackets can be omitted in the regular expressions if taking into account the priority of operations (iteration, product, union) the corresponding languages are not affected. For example instead of $((x^*)(x + y))$ we can consider $x^*(x + y)$.

Two regular expressions are ***equivalent*** if they represent the same language, that is $x \equiv y$ if $X = Y$, where X and Y are the languages represented by regular expressions x and y respectively. Table 2.1 shows some equivalent expressions.

We show that to any finite language L can be associated a regular expression x which represent language L. If $L = \emptyset$, then $x = \emptyset$. If $L = \{w_1, w_2, \ldots, w_n\}$, then $x = x_1 + x_2 + \ldots + x_n$, where for any $i = 1, 2, \ldots, n$ expression x_i is a regular expression representing language $\{w_i\}$. This latter can be done by the following rule. If $w_i = \varepsilon$, then $x_i = \varepsilon$, else if $w_i = a_1 a_2 \ldots a_m$, where $m \geq 1$ depends on i, then $x_i = a_1 a_2 \ldots a_m$, where the brackets are omitted.

We prove the theorem of Kleene which refers to the relationship between regular languages and regular expression.

Theorem 2.20 (Kleene's theorem). *Language $L \subseteq \Sigma^*$ is regular if and only if there exists a regular expression over Σ representing language L.*

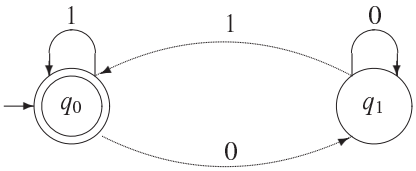


Figure 2.18. DFA from Example 2.20., to which regular expression is associated by Method 1.

Proof. First we prove that if x is a regular expression, then language L which represents x is also regular. The proof will be done by induction on the construction of expression.

If $x = \emptyset$, $x = \varepsilon$, $x = a, \forall a \in \Sigma$, then $L = \emptyset$, $L = \{\varepsilon\}$, $L = \{a\}$ respectively. Since L is finite in all three cases, it is also regular.

If $x = (x_1 + x_2)$, then $L = L_1 \cup L_2$, where L_1 and L_2 are the languages which represent the regular expressions x_1 and x_2 respectively. By the induction hypothesis languages L_1 and L_2 are regular, so L is also regular because regular languages are closed on union. Cases $x = (x_1 x_2)$ and $x = (x_1^*)$ can be proved by similar way.

Conversely, we prove that if L is a regular language, then a regular expression x can be associated to it, which represent exactly the language L. If L is regular, then there exists a DFA $A = (Q, \Sigma, E, \{q_0\}, F)$ for which $L = L(A)$. Let $q_0, q_1, \ldots, q_n$ the states of the automaton A. Define languages R_{ij}^k for all $-1 \leq k \leq n$ and $0 \leq i, j \leq n$. R_{ij}^k is the set of words, for which automaton A goes from state q_i to state q_j without using any state with index greater than k. Using transition graph we can say: a word is in R_{ij}^k, if from state q_i we arrive to state q_j following the edges of the graph, and concatenating the corresponding labels on edges we get exactly that word, not using any state $q_{k+1}, \ldots q_n$. Sets R_{ij}^k can be done also formally:

$R_{ij}^{-1} = \{a \in \Sigma \mid (q_i, a, q_j) \in E\}$, if $i \neq j$,

$R_{ii}^{-1} = \{a \in \Sigma \mid (q_i, a, q_i) \in E\} \cup \{\varepsilon\}$,

$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1} \left(R_{kk}^{k-1}\right)^* R_{kj}^{k-1}$ for all $i, j, k \in \{0, 1, \ldots, n\}$.

We can prove by induction that sets R_{ij}^k can be described by regular expressions. Indeed, if $k = -1$, then for all i and j languages R_{ij}^k are finite, so they can be expressed by regular expressions representing exactly these languages. Moreover, if for all i and j language R_{ij}^{k-1} can be expressed by regular expression, then language R_{ij}^k can be expressed also by regular expression, which can be corresponding constructed from regular expressions representing languages R_{ij}^{k-1}, R_{ik}^{k-1}, R_{kk}^{k-1} and R_{kj}^{k-1} respectively, using the above formula for R_{ij}^k.

Finally, if $F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ is the set of final states of the DFA A, then $L = L(A) = R_{0i_1}^n \cup R_{0i_2}^n \cup \ldots \cup R_{0i_p}^n$ can be expressed by a regular expression obtained from expressions representing languages $R_{0i_1}^n, R_{0i_2}^n, \ldots, R_{0i_p}^n$ using operation +. ■

Further on we give some procedures which associate DFA to regular expressions and conversely regular expression to DFA.

Associating regular expressions to finite automata

We present here three methods, each of which associate to a DFA the corresponding regular expression.

Method 1. Using the result of the theorem of Kleene, we will construct the sets R_{ij}^k, and write a regular expression which represent the language $L = R_{0i_1}^n \cup R_{0i_2}^n \cup \ldots \cup R_{0i_p}^n$, where

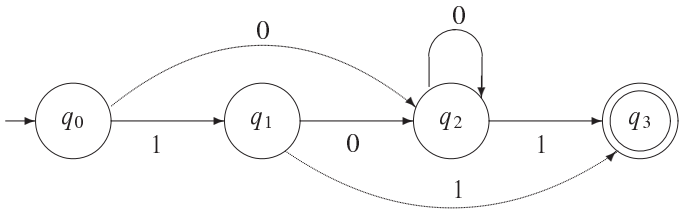


Figure 2.19. DFA in example 2.21. to which a regular expression is associated by Method 1. The computation are in the Table 2.2.

$F = \{q_{i_1}, q_{i_2}, \ldots, q_{i_p}\}$ is the set of final states of the automaton.

Example 2.20 Consider the DFA in Fig. 2.18.

$L(\mathrm{A}) = R^1_{00} = R^0_{00} \cup R^0_{01}\left(R^0_{11}\right)^* R^0_{10}$

R^0_{00} : $1^* + \varepsilon \equiv 1^*$

R^0_{01} : 1^*0

R^0_{11} : $11^*0 + \varepsilon + 0 \equiv (11^* + \varepsilon)0 + \varepsilon \equiv 1^*0 + \varepsilon$

R^0_{10} : 11^*

Then the regular expression corresponding to $L(\mathrm{A})$ is $1^* + 1^*0(1^*0 + \varepsilon)^*11^* \equiv 1^* + 1^*0(1^*0)^*11^*$.

Example 2.21 Find a regular expression associated to DFA in Fig. 2.19. The computations are in Table 2.2. The regular expression corresponding to R^3_{03} is $11 + (0 + 10)0^*1$.

Method 2. Now we generalize the notion of finite automaton, considering words instead of letters as labels of edges. In such an automaton each walk determine a regular expression, which determine a regular language. The regular language accepted by a generalized finite automaton is the union of regular languages determined by the productive walks. It is easy to see that the generalized finite automata accept regular languages.

The advantage of generalized finite automata is that the number of its edges can be diminuted by equivalent transformations, which do not change the accepted language, and leads to a graph with only one edge which label is exactly the accepted language.

The possible equivalent transformations can be seen in Fig. 2.20. If some of the vertices 1, 2, 4, 5 on the figure coincide, in the result they are merged, and a loop will arrive.

First, the automaton is transformed by corresponding ε-moves to have only one initial and one final state. Then, applying the equivalent transformations until the graph will have only one edge, we will obtain as the label of this edge the regular expression associated to the automaton.

Example 2.22 In the case of Fig. 2.18 the result is obtained by steps illustrated in Fig. 2.21. This result is $(1 + 00^*1)^*$, which represents the same language as obtained by Method 1 (See example 2.20.).

Example 2.23 In the case of Fig. 2.19 is not necessary to introduce new initial and final state. The

	$k=-1$	$k=0$	$k=1$	$k=2$	$k=3$
R_{00}^k	ε	ε	ε	ε	
R_{01}^k	1	1	1	1	
R_{02}^k	0	0	$0+10$	$(0+10)0^*$	
R_{03}^k	$\emptyset$	$\emptyset$	11	$11+(0+10)0^*1$	$11+(0+10)0^*1$
R_{11}^k	ε	ε	ε	ε	
R_{12}^k	0	0	0	00^*	
R_{13}^k	1	1	1	$1+00^*1$	
R_{22}^k	$0+\varepsilon$	$0+\varepsilon$	$0+\varepsilon$	0^*	
R_{23}^k	1	1	1	0^*1	
R_{33}^k	ε	ε	ε	ε	

2.2. Table. Determining a regular expression associated to DFA in Fig. 2.19 using sets R_{ij}^k.

steps of transformations can be seen in Fig. 2.22. The resulted regular expression can be written also as $(0+10)0^*1+11$, which is the same as obtained by the previous method.

Method 3. The third method for writing regular expressions associated to finite automata uses formal equations. A variable X is associated to each state of the automaton (to different states different variables). Associate to each state an equation which left side contains X, its right side contains sum of terms of form Ya or ε, where Y is a variable associated to a state, and a is its corresponding input symbol. If there is no incoming edge in the state corresponding to X then the right side of the equation with left side X contains ε, otherwise is the sum of all terms of the form Ya for which there is a transition labelled with letter a from state corresponding to Y to the state corresponding to X. If the state corresponding to

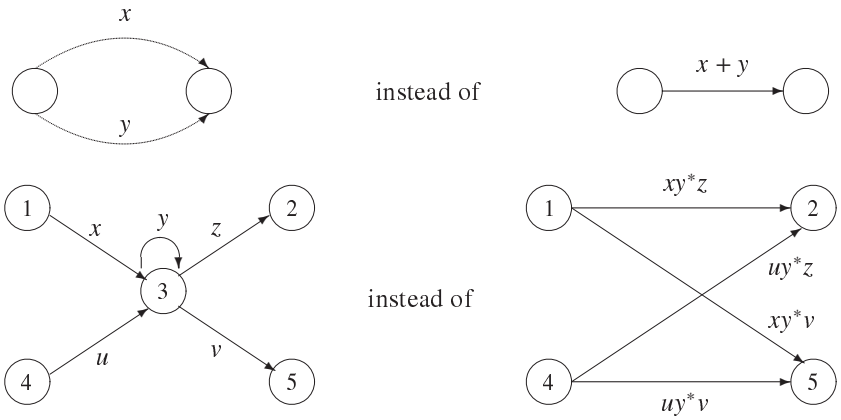


Figure 2.20. Possible equivalent transformations for finding regular expression associated to an automaton.

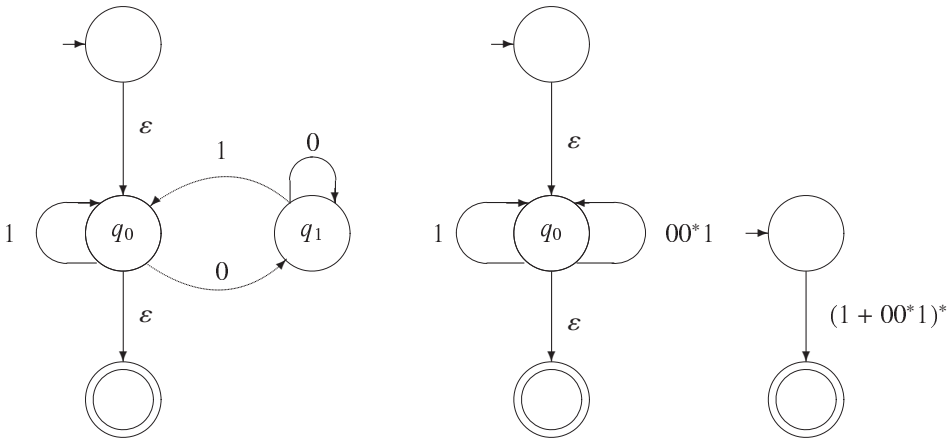


Figure 2.21. Transformation of the finite automaton in Fig. 2.18.

X is also an initial and a final state, then on right side of the equation with the left side X will be also a term equal to ε. For example in the case of Fig. 2.19 let these variable X, Y, Z, U corresponding to the states q_0, q_1, q_2, q_3. The corresponding equation are

$X = \varepsilon$

$Y = X1$

$Z = X0 + Y0 + Z0$

$U = Y1 + Z1.$

If an equation is of the form $X = X\alpha + \beta$, where α, β are arbitrary words not containing X, then it is easy to see by a simple substitution that $X = \beta\alpha^*$ is a solution of the equation.

Because these equations are linear, all of them can be written in the form $X = X\alpha + \beta$ or $X = X\alpha$, where α do not contain any variable. Substituting this in the other equations the number of remaining equations will be diminuted by one. In such a way the system of equation can be solved for each variable.

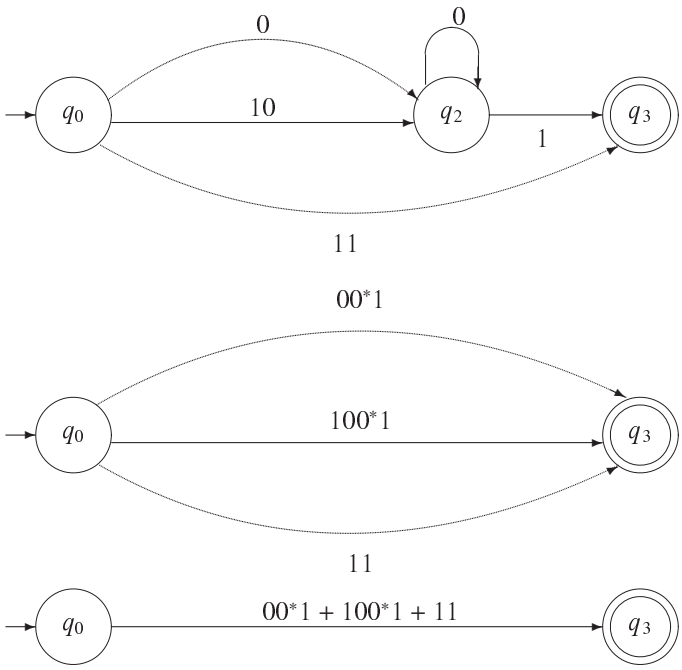


Figure 2.22. Steps of example 2.23..

The solution will be given by variables corresponding to final states summing the corresponding regular expressions.

In our example from the first equation we get $Y = 1$. From here $Z = 0 + 10 + Z0$, or $Z = Z0 + (0 + 10)$, and solving this we get $Z = (0 + 10)0^*$. Variable U can be obtained immediately and we obtain $U = 11 + (0 + 10)0^*1$.

Using this method in the case of Fig. 2.18, the following equations will be obtained

$X = \varepsilon + X1 + Y1$

$Y = X0 + Y0$

Therefore

$X = \varepsilon + (X + Y)1$

$Y = (X + Y)0$.

Adding the two equations we will obtain

$X + Y = \varepsilon + (X + Y)(0 + 1)$, from where (considering ε as β and $(0 + 1)$ as α) we get the result

$X + Y = (0 + 1)^*$.

From here the value of X after the substitution is

$X = \varepsilon + (0 + 1)^*1$,

which is equivalent to the expression obtained using the other methods.

Associating finite automata to regular expressions

Associate to the regular expression r a generalized finite automaton:

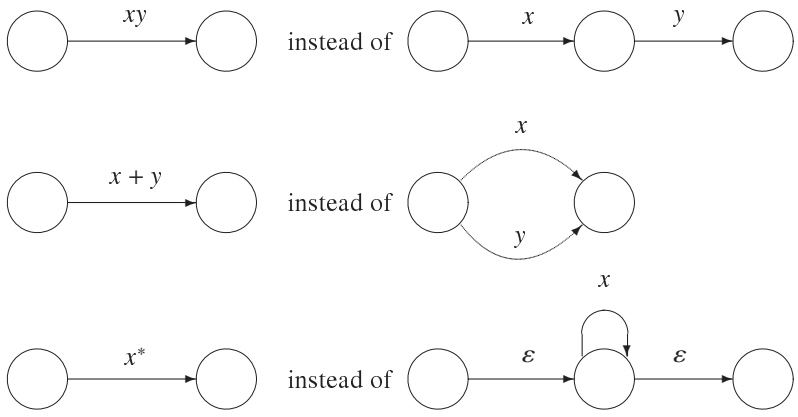
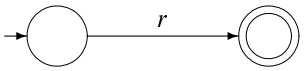


Figure 2.23. Possible transformations to obtain finite automaton associated to a regular expression.



After this, use the transformations in Fig. 2.23 step by step, until an automaton with labels equal to letters from Σ or ε will be obtained.

Example 2.24 Get started from regular expression $\varepsilon+(0+1)^*1$. The steps of transformations are in Fig. 2.24(a)-(e). The last finite automaton (see Fig. 2.24(e)) can be done in a simpler form as can be seen in Fig. 2.24(f). After eliminating the ε-moves and transforming in a deterministic finite automaton the DFA in Fig. 2.25 will be obtained, which is equivalent to DFA in Fig. 2.18.

Exercises

2.2-1 Give a DFA which accepts natural numbers divisible by 9.

2.2-2 Give a DFA which accepts the language containing all words formed by

a. an even number of 0's and an even number of 1's,

b. an even number of 0's and an odd number of 1's,

c. an odd number of 0's and an even number of 1's,

d. an odd number of 0's and an odd number of 1's.

2.2-3 Give a DFA to accept respectively the following languages:

$L_1 = \{a^n b^m \mid n \geq 1, m \geq 0\}$, $L_2 = \{a^n b^m \mid n \geq 1, m \geq 1\}$,

$L_3 = \{a^n b^m \mid n \geq 0, m \geq 0\}$, $L_4 = \{a^n b^m \mid n \geq 0, m \geq 1\}$.

2.2-4 Give an NFA which accepts words containing at least two 0's and any number of 1's. Give an equivalent DFA.

2.2-5 Minimize the DFA's in Fig. 2.26.

2.2-6 Show that the DFA in 2.27.(a) is a minimum state automaton.

2.2-7 Transform NFA in Fig. 2.27.(b) in a DFA, and after this minimize it.

2.2-8 Define finite automaton A_1 which accepts all words of the form $0(10)^n$ ($n \geq 0$), and finite automaton A_2 which accepts all words of the form $1(01)^n$ ($n \geq 0$). Define the union

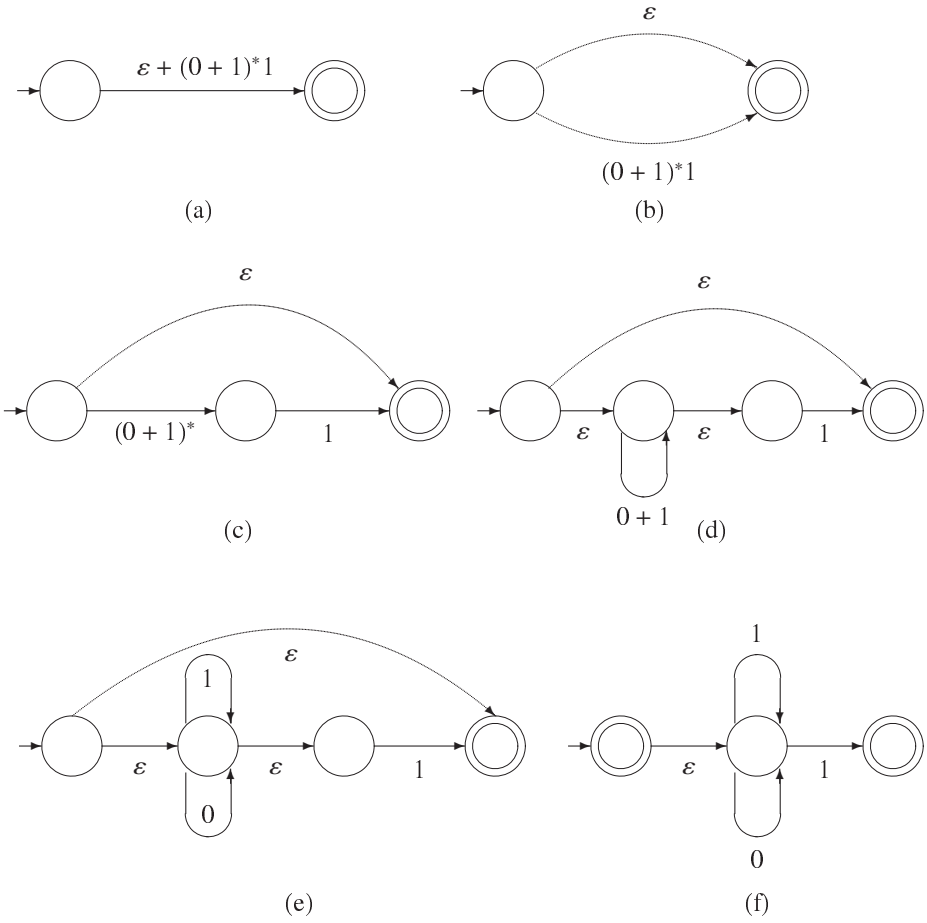


Figure 2.24. Associating finite automaton to regular expression $\varepsilon + (0 + 1)^*1$.

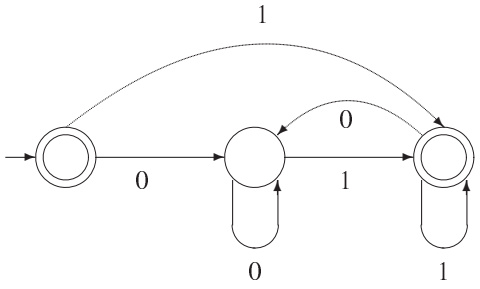


Figure 2.25. Finite automaton associated to regular expression $\varepsilon + (0 + 1)^*1$.

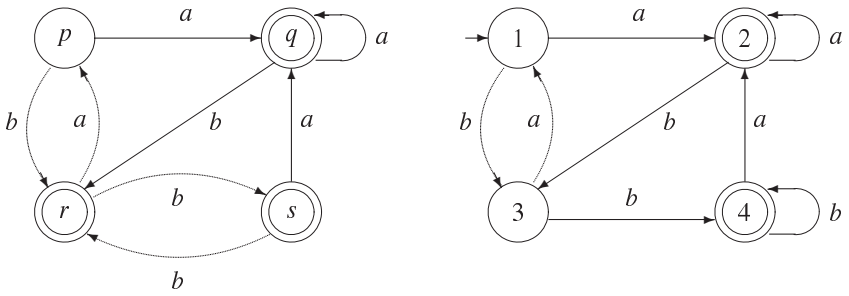


Figure 2.26. DFA's to minimize for exercise 2.2-5..

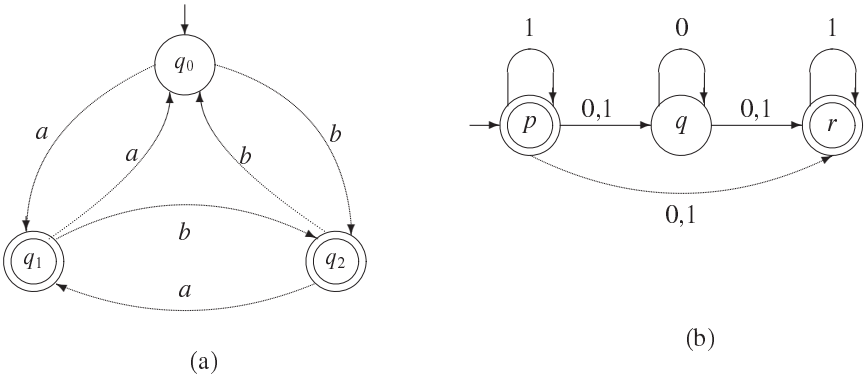


Figure 2.27. Finite automata for exercises 2.2-6. and 2.2-7..

automaton $A_1 \cup A_2$, and then eliminate the ε-moves.

2.2-9 Associate to DFA in Fig. 2.28 a regular expression.

2.2-10 Associate to regular expression $ab^*ba^* + b + ba^*a$ a DFA.

2.2-11 Prove, using the pumping lemma, that none of the following languages are regular:

$L_1 = \{a^n c b^n \mid n \geq 0\}$, $L_2 = \{a^n b^n a^n \mid n \geq 0\}$, $L_3 = \{a^p \mid p \text{ prím}\}$.

2.2-12 Prove that if L is a regular language, then $\{u^{-1} \mid u \in L\}$ is also regular.

2.2-13 Prove that if $L \subseteq \Sigma^*$ is a regular language, then the following languages are also regular.

$\text{pre}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, wu \in L\}$, $\text{suf}(L) = \{w \in \Sigma^* \mid \exists u \in \Sigma^*, uw \in L\}$.

2.2-14 Show that the following languages are all regular.

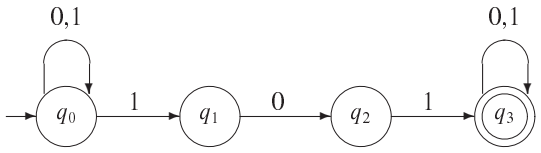


Figure 2.28. DFA for exercise 2.2-9..

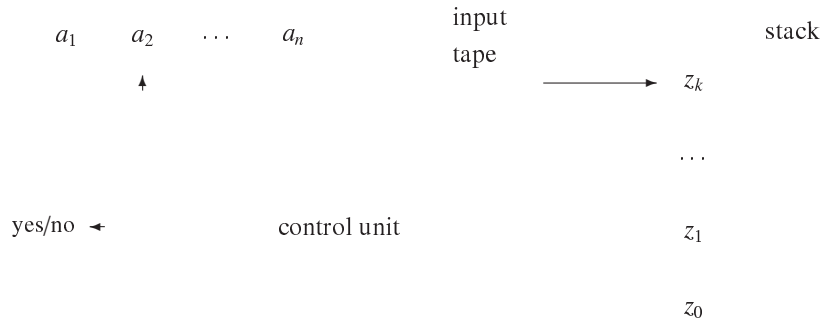


Figure 2.29. Pushdown automaton.

$L_1 = \{ab^n cd^m \mid n > 0, m > 0\}$,
$L_2 = \{(ab)^n \mid n \geq 0\}$,
$L_3 = \{a^{kn} \mid n \geq 0, \quad k \text{ constant}\}$.

2.3. Pushdown automata and context-free languages

In this section we deal with the pushdown automata and the class of languages — the context-free languages — accepted by them.

As we have been seen in the section 2.1, a context-free grammar $G = (N, T, P, S)$ is one with the productions of the form $A \to \beta$, $A \in N, \beta \in (N \cup T)^+$. The production $S \to \varepsilon$ is also permitted if S does not appear in right hand side of any productions. Language $L(G) = \{u \in T \mid S \overset{*}{\underset{G}{\Longrightarrow}} u\}$ is the context-free language generated by grammar G.

2.3.1. Pushdown automata

We have been seen that finite automata accept the class of regular languages. Now we get to know a new kind of automata, the so-called ***pushdown automata***, which accept context-free languages. The pushdown automata differ from finite automata mainly in that to have the possibility to change states without reading any input symbol (i.e. to read the empty symbol) and possess a stack memory, which uses the so-called stack symbols (See Fig. 2.29).

The pushdown automaton get a word as input, start to function from an initial state having in the stack a special symbol, the initial stack symbol. While working, the pushdown automaton change its state based on current state, next input symbol (or empty word) and stack top symbol and replace the top symbol in the stack with a (possibly empty) word.

There are two type of acceptances. The pushdown automaton accepts a word by final state when after reading it the automaton enter a final state. The pushdown automaton accepts a word by empty stack when after reading it the automaton empties its stack. We show that these two acceptances are equivalent.

Definition 2.21 *A **nondeterministic pushdown automaton** is a system*

$$V = (Q, \Sigma, W, E, q_0, z_0, F),$$

where

- Q *is the finite, non-empty set of* ***states***
- Σ *is the* ***input alphabet,***
- W *is the* ***stack alphabet,***
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times W \times W^* \times Q$ *is the set of* ***transitions*** *or* ***edges***,
- $q_0 \in Q$ *is the* ***initial state,***
- $z_0 \in W$ *is the* ***start symbol of stack,***
- $F \subseteq Q$ *is the set of* ***final states***.

A transition (p, a, z, w, q) means that if pushdown automaton V is in state p, reads from the input tape letter a (instead of input letter we can also consider the empty word ε), and the top symbol in the stack is z, then the pushdown automaton enters state q and replaces in the stack z by word w. Writing word w in the stack is made by natural order (letters of word w will be put in the stack letter by letter from left to right). Instead of writing transition (p, a, z, w, q) we will use a more suggestive notation $(p, (a, z/w), q)$.

Here, as in the case of finite automata, we can define a transition function

$$\delta \ : \ Q \times (\Sigma \cup \{\varepsilon\}) \times W \to \mathcal{P}(W^* \times Q) \ ,$$

which associate to current state, input letter and top letter in stack pairs of the form (w, q), where $w \in W^*$ is the word written in stack and $q \in Q$ the new state.

Because the pushdown automaton is nondeterministic, we will have for the transition function

$\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (if the pushdown automaton reads an input letter and moves to right), or

$\delta(q, \varepsilon, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$ (without move on the input tape).

A pushdown automaton is ***deterministic,*** if for any $q \in Q$ and $z \in W$ we have

- $|\delta(q, a, z)| \leq 1$, $\forall a \in \Sigma \cup \{\varepsilon\}$ and
- if $\delta(q, \varepsilon, z) \neq \emptyset$, then $\delta(q, a, z) = \emptyset$, $\forall a \in \Sigma$.

We can associate to any pushdown automaton a transition table, exactly as in the case of finite automata. The rows of this table are indexed by elements of Q, the columns by elements from $\Sigma \cup \{\varepsilon\}$ and W (to each $a \in \Sigma \cup \{\varepsilon\}$ and $z \in W$ will correspond a column). At intersection of row corresponding to state $q \in Q$ and column corresponding to $a \in \Sigma \cup \{\varepsilon\}$ and $z \in W$ we will have pairs $(w_1, p_1), \ldots, (w_k, p_k)$ if $\delta(q, a, z) = \{(w_1, p_1), \ldots, (w_k, p_k)\}$.

The transition graph, in which the label of edge (p, q) will be $(a, z/w)$ corresponding to transition $(p, (a, z/w), q)$, can be also defined.

Example 2.25 $V_1 = (\{q_0, q_1, q_2\}, \{a, b\}, \{z_0, z_1\}, E, q_0, z_0, \{q_0\})$. Elements of E are:

$(q_0, (a, z_0/z_0z_1), q_1)$
$(q_1, (a, z_1/z_1z_1), q_1)$ $\qquad$ $(q_1, (b, z_1/\varepsilon), q_2)$
$(q_2, (b, z_1/\varepsilon), q_2)$ $\qquad$ $(q_2, (\varepsilon, z_0/\varepsilon), q_0)$.

The transition function:

$\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$
$\delta(q_1, a, z_1) = \{(z_1z_1, q_1)\}$ $\qquad$ $\delta(q_1, b, z_1) = \{(\varepsilon, q_2)\}$
$\delta(q_2, b, z_1) = \{(\varepsilon, q_2)\}$ $\qquad$ $\delta(q_2, \varepsilon, z_0) = \{(\varepsilon, q_0)\}$.

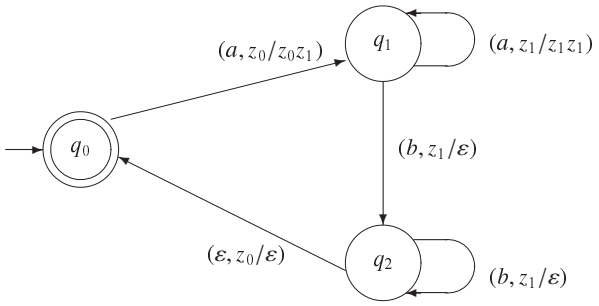


Figure 2.30. Example of pushdown automaton.

The transition table:

$\Sigma \cup \{\varepsilon\}$	a		b	ε
W	z_0	z_1	z_1	z_0
q_0	(z_0z_1, q_1)			
q_1		(z_1z_1, q_1)	(ε, q_2)	
q_2			(ε, q_2)	(ε, q_0)

Because for the transition function every set which is not empty contains only one element (e.g. $\delta(q_0, a, z_0) = \{(z_0z_1, q_1)\}$), in the above table each cell contains only one element, And the set notation is not used. Generally, if a set has more than one element, then its elements are written one under other. The transition graph of this pushdown automaton is in Fig. 2.30.

The current state, the unread part of the input word and the content of stack constitutes a ***configuration*** of the pushdown automaton, i.e. for each $q \in Q$, $u \in \Sigma^*$ and $v \in W^*$ the triplet (q, u, v) can be a configuration.

If $u = a_1a_2 \dots a_k$ and $v = x_1x_2 \dots x_m$, then the pushdown automaton can change its configuration in two ways:

- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_{m-1}x_m) \Longrightarrow (p,\ a_2a_3 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$,
  if $(q, (a_1, x_m/w), p) \in E$
- $(q,\ a_1a_2 \dots a_k,\ x_1x_2 \dots x_m) \Longrightarrow (p,\ a_1a_2 \dots a_k,\ x_1, x_2 \dots x_{m-1}w)$,
  if $(q, (\varepsilon, x_m/w), p) \in E$.

The reflexive and transitive closure of the relation $\Longrightarrow$ will be denoted by $\overset{*}{\Longrightarrow}$. Instead of using $\Longrightarrow$, sometimes $\vdash$ is considered.

How does work such a pushdown automaton? Getting started with the initial configuration $(q_0, a_1a_2 \dots a_n, z_0)$ we will consider all possible next configurations, and after this the next configurations to these next configurations, and so on, until it is possible.

Definition 2.22 *Pushdown automaton* V *accepts (recognizes) word* u ***by final state*** *if there exist a sequence of configurations of* V *for which the following are true:*

- *the first element of the sequence is* (q_0, u, z_0),
- *there is a going from each element of the sequence to the next element, excepting the case when the sequence has only one element,*
- *the last element of the sequence is* (p, ε, w), *where* $p \in F$ *and* $w \in W^*$.

Therefore pushdown automaton V accepts word u by final state, if and only if $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, w)$ for some $w \in W^*$ and $p \in F$. The set of words accepted by final state by pushdown automaton V will be called the language accepted by V by final state and will be denoted by $L(\mathrm{V})$.

Definition 2.23 *Pushdown automaton* V *accepts (recognizes) word* u ***by empty stack*** *if there exist a sequence of configurations of* V *for which the following are true:*

- *the first element of the sequence is* (q_0, u, z_0),
- *there is a going from each element of the sequence to the next element,*
- *the last element of the sequence is* $(p, \varepsilon, \varepsilon)$ *and* p *is an arbitrary state.*

Therefore pushdown automaton V accepts a word u by empty stack if $(q_0, u, z_0) \overset{*}{\Longrightarrow} (p, \varepsilon, \varepsilon)$ for some $p \in Q$. The set of words accepted by empty stack by pushdown automaton V will be called the language accepted by empty stack by V and will be denoted by $L_\varepsilon(\mathrm{V})$.

Example 2.26 Pushdown automaton V_1 of Example 2.25. accepts the language $\{a^n b^n \mid n \geq 0\}$ by final state. Consider the derivation for words *aaabbb* and *abab*.

Word a^3b^3 is accepted by the considered pushdown automaton because

$(q_0, aaabbb, z_0) \Longrightarrow (q_1, aabbb, z_0z_1) \Longrightarrow (q_1, abbb, z_0z_1z_1) \Longrightarrow (q_1, bbb, z_0z_1z_1z_1)$

$\Longrightarrow (q_2, bb, z_0z_1z_1) \Longrightarrow (q_2, b, z_0z_1) \Longrightarrow (q_2, \varepsilon, z_0) \Longrightarrow (q_0, \varepsilon, \varepsilon)$ and because q_0 is a final state the pushdown automaton accepts this word. But the stack being empty, it accepts this word also by empty stack.

Because the initial state is also a final state, the empty word is accepted by final state, but not by empty stack.

To show that word *abab* is not accepted, we need to study all possibilities. It is easy to see that in our case there is only a single possibility:

$(q_0, abab, z_0) \Longrightarrow (q_1, bab, z_0z_1) \Longrightarrow (q_2, ab, z_0) \Longrightarrow (q_0, ab, \varepsilon)$, but there is no further going, so word *abab* is not accepted.

Example 2.27 The transition table of the pushdown automaton $\mathrm{V}_2 = (\{q_0, q_1\}, \{0, 1\}, \{z_0, z_1, z_2\}, E, q_0, z_0, \emptyset)$ is:

$\Sigma \cup \{\varepsilon\}$	0			1			ε
W	z_0	z_1	z_2	z_0	z_1	z_2	z_0
q_0	(z_0z_1, q_0)	(z_1z_1, q_0) (ε, q_1)	(z_2z_1, q_0)	(z_0z_2, q_0)	(z_1z_2, q_0)	(z_2z_2, q_0) (ε, q_1)	(ε, q_1)
q_1		(ε, q_1)				(ε, q_1)	(ε, q_1)

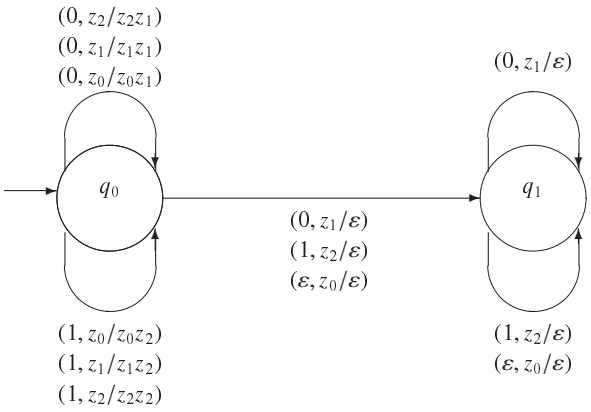


Figure 2.31. Transition graph of the Example 2.27..

The corresponding transition graph can be seen in Fig. 2.31. Pushdown automaton V_2 accepts the language $\{uu^{-1} \mid u \in \{0,1\}^*\}$. Because V_2 is nemdeterministic, all the configurations obtained from the initial configuration (q_0, u, z_0) can be illustrated by a ***computation tree***. For example the computation tree associated to the initial configuration $(q_0, 1001, z_0)$ can be seen in Fig. 2.32. From this computation tree we can observe that, because $(q_1, \varepsilon, \varepsilon)$ is a leaf of the tree, pushdown automaton V_2 accepts word 1001 by empty stack. The computation tree in Fig. 2.33 shows that pushdown automaton V_2 does not accept word 101, because the configurations in leaves can not be continued and none of them has the form $(q, \varepsilon, \varepsilon)$.

Theorem 2.24 *A language L is accepted by a nondeterministic pushdown automaton V_1 by empty stack if and only if it can be accepted by a nondeterministic pushdown automaton V_2 by final state.*

Proof. *a)* Let $V_1 = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ be the pushdown automaton which accepts by empty stack language L. Define pushdown automaton $V_2 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \{p\})$, where $p, p_0 \notin Q$, , $x \notin W$ and

$$E' = E \cup \left\{(p_0, (\varepsilon, x/xz_0), q_0)\right\} \cup \left\{(q, (\varepsilon, x/\varepsilon), p) \mid q \in Q\right\}$$

Working of V_2: Pushdown automaton V_2 with an ε-move first goes in the initial state of V_1, writing z_0 (the initial stack symbol of V_1) in the stack (beside x). After this it is working as V_1. If V_1 for a given word empties its stack, then V_2 still has x in the stack, which can be deleted by V_2 using an ε-move, while a final state will be reached. V_2 can reach a final state only if V_1 has emptied the stack.

b) Let $V_2 = (Q, \Sigma, W, E, q_0, z_0, F)$ be a pushdown automaton, which accepts language L by final state. Define pushdown automaton $V_1 = (Q \cup \{p_0, p\}, \Sigma, W \cup \{x\}, E', p_0, x, \emptyset)$, where $p_0, p \notin Q$, $x \notin W$ and

$$E' = E \cup \left\{(p_0, (\varepsilon, x/xz_0), q_0)\right\} \cup \left\{(q, (\varepsilon, z/\varepsilon), p) \mid q \in F, p \in Q, z \in W\right\}$$
$$\cup \left\{(p, (\varepsilon, z/\varepsilon), p) \mid p \in Q, z \in W \cup \{x\}\right\}$$

Working V_1: Pushdown automaton V_1 with an ε-move writes in the stack beside x the initial

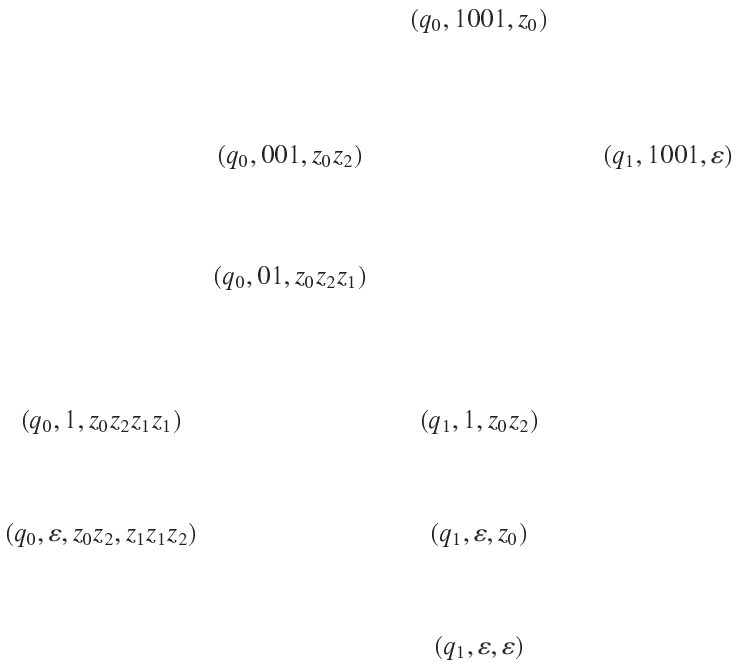


Figure 2.32. Computation tree to show acceptance of the word 1001 (see Example 2.27.).

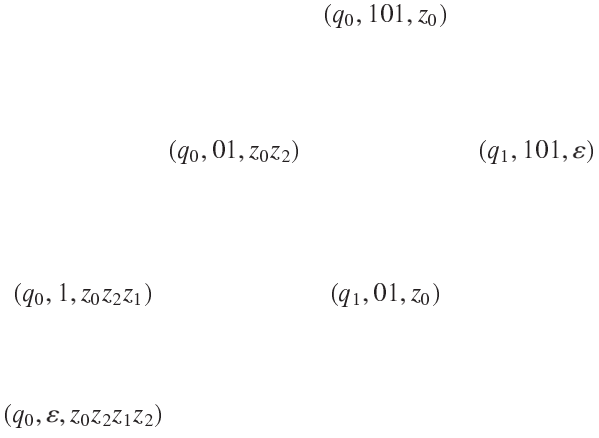


Figure 2.33. Computation tree to show that the pushdown automaton in Example 2.27. does not accept word 101.

stack symbol z_0 of V_2, then works as V_2, i.e reaches a final state for each accepted word. After this V_1 empties the stack by an ε-move. V_1 can empty the stack only if V_2 goes in a final state. ■

The next two theorems prove that the class of languages accepted by nondeterministic pushdown automata is just the set of context-free languages.

Theorem 2.25 *If G is a context-free grammar, then there exists such a nondeterministic pushdown automaton* V *which accepts $L(G)$ by empty stack, i.e. $L_\varepsilon(\mathrm{V}) = L(G)$.*

We outline the proof only. Let $G = (N, T, P, S)$ be a context-free grammar. Define pushdown automaton $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$, where $q \notin N \cup T$, and the set E of transitions is:

- If there is in the set of productions of G a production of type $A \to \alpha$, then let put in E the transition $(q, (\varepsilon, A/\alpha^{-1}), q)$,
- For any letter $a \in T$ let put in E the transition $(q, (a, a/\varepsilon), q)$.

If there is a production $S \to \alpha$ in G, the pushdown automaton put in the stack the mirror of α with an ε-move. If the input letter coincides with that in the top of the stack, then the automaton deletes it from the stack. If in the top of the stack there is a nonterminal A, then the mirror of right-hand side of a production which has A in its left-hand side will be put in the stack. If after reading all letters of the input word, the stack will be empty, then the pushdown automaton recognized the input word.

The following algorithm builds for a context-free grammar $G = (N, T, P, S)$ the pushdown automaton $\mathrm{V} = (\{q\}, T, N \cup T, E, q, S, \emptyset)$, which accepts by empty stack the language generated by G.

FROM-CFG-TO-PUSHDOWN-AUTOMATON(G, V)

1 **for** all production $A \to \alpha$
2 **do** put in E the transition $(q, (\varepsilon, A/\alpha^{-1}), q)$
3 **for** all terminal $a \in T$
4 **do** put in E the transition $(q, (a, a/\varepsilon), q)$

If G has n productions and m terminals, then the number of step of the algorithm is $\Theta(n + m)$.

Example 2.28 Let $G = (\{S, A\}, \{a, b\}, \{S \to \varepsilon,\ S \to ab,\ S \to aAb,\ A \to aAb,\ A \to ab\}, S)$. Then $\mathrm{V} = (\{q\}, \{a, b\}, \{a, b, A, S\}, E, q, S, \emptyset)$, with the following transition table.

$\Sigma \cup \{\varepsilon\}$	a	b	ε	
W	a	b	S	A
q	(ε, q)	(ε, q)	(ε, q) (ba, q) (bAa, q)	(bAa, q) (ba, q)

Let us see how pushdown automaton V accepts word *aabb*, which in grammar G can be derived in the following way:

$$S \Longrightarrow aAb \Longrightarrow aabb,$$

where productions $S \to aAb$ and $A \to ab$ were used. Word is accepted by empty stack (see Fig. 2.34).

Theorem 2.26 *For a nondeterministic pushdown automaton* V *there exists always a*

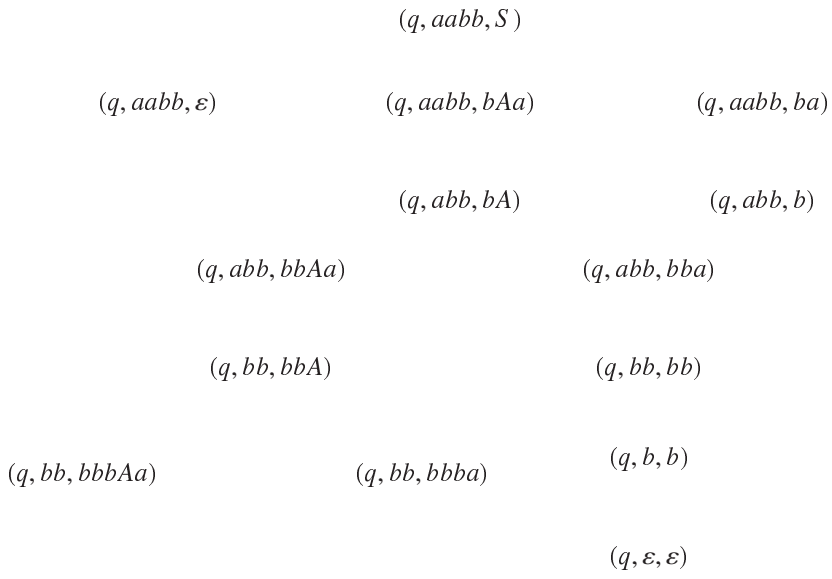


Figure 2.34. Recognising a word by empty stack (see Example 2.28.).

context-free grammar G such that V *accepts language $L(G)$ by empty stack, i.e.* $L_\varepsilon(\mathrm{V}) = L(G)$.

Instead of a proof we will give a method to obtain grammar G. Let $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ be the nondeterministic pushdown automaton in question.

Then $G = (N, T, P, S)$, where

$N = \{S\} \cup \{S_{p,z,q} \mid p, q \in Q, z \in W\}$ and $T = \Sigma$.

Productions in P will be obtained as follows.

• For all state q put in P production $S \to S_{q_0,z_0,q}$.

• If $(q, (a, z/z_k \dots z_2 z_1), p) \in E$, where $q \in Q$, $z, z_1, z_2, \dots z_k \in W$ $(k \geq 1)$ and $a \in \Sigma \cup \{\varepsilon\}$, put in P for all possible states $p_1, p_2, \dots, p_k$ productions

$S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$.

• If $(q, (a, z/\varepsilon), p) \in E$, where $p, q \in Q, z \in W$, and $a \in \Sigma \cup \{\varepsilon\}$, put in P production $S_{q,z,p} \to a$.

The context-free grammar defined by this is an extended one, to which an equivalent context-free language can be associated. The proof of the theorem is based on the fact that to every sequence of configurations, by which the pushdown automaton V accepts a word, we can associate a derivation in grammar G. This derivation generates just the word in question, because of productions of the form $S_{q,z,p_k} \to a S_{p,z_1,p_1} S_{p_1,z_2,p_2} \dots S_{p_{k-1},z_k,p_k}$, which were defined for all possible states $p_1, p_2, \dots, p_k$. In Example 2.27. we show how can be associated a derivation to a sequence of configurations. The pushdown automaton defined in the example recognizes word 00 by the sequence of configurations

$(q_0, 00, z_0) \Longrightarrow (q_0, 0, z_0 z_1) \Longrightarrow (q_1, \varepsilon, z_0) \Longrightarrow (q_1, \varepsilon, \varepsilon)$,

which sequence is based on the transitions

$(q_0, (0, z_0/z_0 z_1), q_0)$,

$(q_0, (0, z_1/\varepsilon), q_1)$,

$(q_1, (\varepsilon, z_1/\varepsilon), q_1)$.

To these transitions, by the definition of grammar G, the following productions can be associated

(1) $S_{q_0,z_0,p_2} \longrightarrow 0S_{q_0,z_1,p_1}S_{p_1,z_0,p_2}$ for all states $p_1, p_2 \in Q$,
(2) $S_{q_0,z_1,q_1} \longrightarrow 0$,
(3) $S_{q_1,z_0,q_1} \longrightarrow \varepsilon$.

Furthermore, for each state q productions $S \longrightarrow S_{q_0,z_0,q}$ were defined.

By the existence of production $S \longrightarrow S_{q_0,z_0,q}$ there exists the derivation $S \Longrightarrow S_{q_0,z_0,q}$, where q can be chosen arbitrarily. Let choose in above production (1) state q to be equal to p_2. Then there exists also the derivation

$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0S_{q_0,z_1,p_1}S_{p_1,z_0,q}$,

where $p_1 \in Q$ can be chosen arbitrarily. If $p_1 = q_1$, then the derivation

$S \Longrightarrow S_{q_0,z_0,q} \Longrightarrow 0S_{q_0,z_1,q_1}S_{q_1,z_0,q} \Longrightarrow 00S_{q_1,z_0,q}$

will result. Now let q equal to q_1, then

$S \Longrightarrow S_{q_0,z_0,q_1} \Longrightarrow 0S_{q_0,z_1,q_1}S_{q_1,z_0,q_1} \Longrightarrow 00S_{q_1,z_0,q_1} \Longrightarrow 00$,

which proves that word 00 can be derived used the above grammar.

The next algorithm builds for a pushdown automaton $\mathrm{V} = (Q, \Sigma, W, E, q_0, z_0, \emptyset)$ a context-free grammar $G = (N, T, P, S)$, which generates the language accepted by pushdown automaton V by empty stack.

FROM-PUSHDOWN-AUTOMATON-TO-CF-GRAMMAR(V, G)

1 **for** all $q \in Q$
2 **do** put in P production $S \to S_{q_0,z_0,q}$
3 **for** all $(q, (a, z/z_k \ldots z_2 z_1), p) \in E$ $\triangleright$ $q \in Q$, $z, z_1, z_2, \ldots z_k \in W$ $(k \geq 1)$, $a \in \Sigma \cup \{\varepsilon\}$
4 **do for** all states $p_1, p_2, \ldots, p_k$
5 **do** put in P productions $S_{q,z,p_k} \to aS_{p,z_1,p_1}S_{p_1,z_2,p_2} \ldots S_{p_{k-1},z_k,p_k}$
6 **for** All $(q(a, z/\varepsilon), p) \in E$ $\triangleright$ $p, q \in Q, z \in W$, $a \in \Sigma \cup \{\varepsilon\}$
7 **do** put in P production $S_{q,z,p} \to a$

If the automaton has n states and m productions, then the above algorithm executes at most $n + mn + m$ steps, so in worst case the number of steps is $O(nm)$.

Finally, without proof, we mention that the class of languages accepted by deterministic pushdown automata is a proper subset of the class of languages accepted by nondeterministic pushdown automata. This points to the fact that pushdown automata behave differently as finite automata.

Example 2.29 As an example, consider pushdown automaton V from the Example 2.28.: $\mathrm{V} = (\{q\}, \{a, b\}, \{a, b, A, S\}, E, q, S, \emptyset)$. Grammar G is:

$$G = (\{S, S_a, S_b, S_S, S_A, \}, \{a, b\}, P, S) ,$$

where for all $z \in \{a, b, S, A\}$ instead of $S_{q,z,q}$ we shortly used S_z. The transitions:

$(q, (a, a/\varepsilon), q)$, $(q, (b, b/\varepsilon), q)$,
$(q, (\varepsilon, S/\varepsilon), q)$, $(q, (\varepsilon, S/ba), q)$, $(q, (\varepsilon, S/bAa), q)$,
$(q, (\varepsilon, A/ba), q)$, $(q, (\varepsilon, A/bAa), q)$.

Based on these, the following productions are defined:

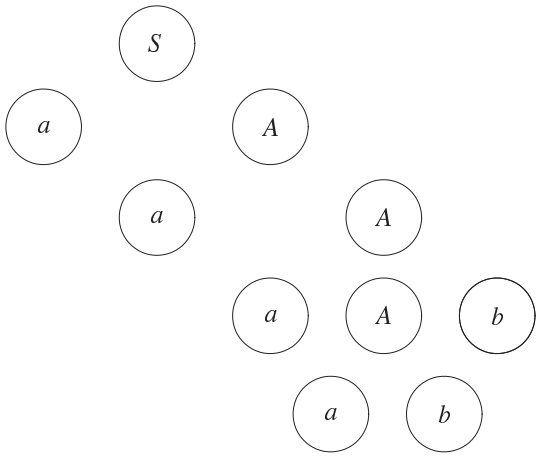


Figure 2.35. Derivation (or syntax) tree of word *aaaabb*.

$S \to S_S$
$S_a \to a$
$S_b \to b$
$S_S \to \varepsilon \mid S_a S_b \mid S_a S_A S_b$
$S_A \to S_a S_A S_b \mid S_a S_b$.

It is easy to see that S_S can be eliminated, and the productions will be:

$S \to \varepsilon \mid S_a S_b \mid S_a S_A S_b$,
$S_A \to S_a S_A S_b \mid S_a S_b$,
$S_a \to a,\ S_b \to b$,

and these productions can be replaced:

$S \to \varepsilon \mid ab \mid aAb$,
$A \to aAb \mid ab$.

2.3.2. Context-free languages

Consider context-free grammar $G = (N, T, P, S)$. A ***derivation tree*** of G is a finite, ordered, labelled tree, which root is labelled by the the start symbol S, every interior vertex is labelled by a nonterminal and every leaf by a terminal. If an interior vertex labelled by a nonterminal A has k descendents, then in P there exists a production $A \to a_1 a_2 \dots a_k$ such that the descendents are labelled by letters $a_1, a_2, \dots a_k$. The ***result*** of a derivation tree is a word over T, which can be obtained by reading the labels of the leaves from left to right. Derivation tree is also called ***syntax tree***.

Consider the context-free grammar $G = (\{S, A\}, \{a, b\}, \{S \to aA,\ S \to a,\ S \to \varepsilon,\ A \to aA,\ A \to aAb,\ A \to ab,\ A \to b\}, S)$. It generates language $L(G) = \{a^n b^m \mid n \geq m \geq 0\}$. Derivation of word $a^4 b^2 \in L(G)$ is:

$$S \Longrightarrow aA \Longrightarrow aaA \Longrightarrow aaaAb \Longrightarrow aaaabb.$$

In Fig. 2.35 this derivation can be seen, which result is *aaaabb*.

To every derivation we can associate a syntax tree. Conversely, to any syntax tree more

than one derivation can be associated. For example to syntax tree in Fig. 2.35 the derivation

$$S \Longrightarrow aA \Longrightarrow aaAb \Longrightarrow aaaAb \Longrightarrow aaaabb$$

also can be associated.

Definition 2.27 *Derivation* $\alpha_0 \Longrightarrow \alpha_1 \Longrightarrow \ldots \Longrightarrow \alpha_n$ *is a* ***leftmost derivation****, if for all* $i = 1, 2, \ldots, n-1$ *there exist words* $u_i \in T^*$, $\beta_i \in (N \cup T)^*$ *and productions* $(A_i \to \gamma_i) \in P$, *for which we have*

$$\alpha_i = u_i A_i \beta_i \quad \text{and} \quad \alpha_{i+1} = u_i \gamma_i \beta_i.$$

Consider grammar:

$G = (\{S, A\}, \{a, b, c\}, \{S \to bA,\ S \to bAS,\ S \to a,\ A \to cS,\ A \to a\}, S)$.

In this grammar word *bcbaa* has two different leftmost derivations:

$S \Longrightarrow bA \Longrightarrow bcS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$,

$S \Longrightarrow bAS \Longrightarrow bcSS \Longrightarrow bcbAS \Longrightarrow bcbaS \Longrightarrow bcbaa$.

Definition 2.28 *A context-free grammar G is* ***ambiguous*** *if in L(G) there exists a word with more than one leftmost derivation. Otherwise G is* ***unambiguous****.*

The above grammar G is ambiguous, because word *bcbaa* has two different leftmost derivations. A language can be generated by more than one grammar, and between them can exist ambiguous and unambiguous too. A context-free language is ***inherently ambiguous,*** if there is no unambiguous grammar which generates it.

Example 2.30 Examine the following two grammars.

Grammar $G_1 = (\{S\}, \{a, +, *\}, \{S \to S + S,\ S \to S * S,\ S \to a\}, S)$ is ambiguous because

$S \Longrightarrow S + S \Longrightarrow a + S \Longrightarrow a + S * S \Longrightarrow a + a * S \Longrightarrow a + a * S + S \Longrightarrow a + a * a + S$
$\Longrightarrow a + a * a + a$ and

$S \Longrightarrow S * S \Longrightarrow S + S * S \Longrightarrow a + S * S \Longrightarrow a + a * S \Longrightarrow a + a * S + S \Longrightarrow a + a * a + S$
$\Longrightarrow a + a * a + a$.

Grammar $G_2 = (\{S, A\}, \{a, *, +\}, \{S \to A + S \mid A,\ A \to A * A \mid a\}, S)$ is unambiguous.

Can be proved that $L(G_1) = L(G_2)$.

2.3.3. Pumping lemma for context-free languages

Like for regular languages there exists a pumping lemma also for context-free languages.

Theorem 2.29 (pumping lemma). *For any context-free language L there exists a natural number n (which depends only on L), such that every word z of the language longer than n can be written in the form uvwxy and the following are true:*

(1) $|w| \geq 1$,

(2) $|vx| \geq 1$,

(3) $|vwx| \leq n$,

(4) $uv^i wx^i y$ *is also in L for all* $i \geq 0$.

Proof. Let $G = (N, T, P, S)$ be a grammar without unit productions, which generates language L. Let $m = |N|$ be the number of nonterminals, and let ℓ be the maximum of lengths of right-hand sides of productions, i.e. $\ell = \max\{|\alpha| \mid \exists A \in N : (A \to \alpha) \in P\}$. Let $n = \ell^{m+1}$ and

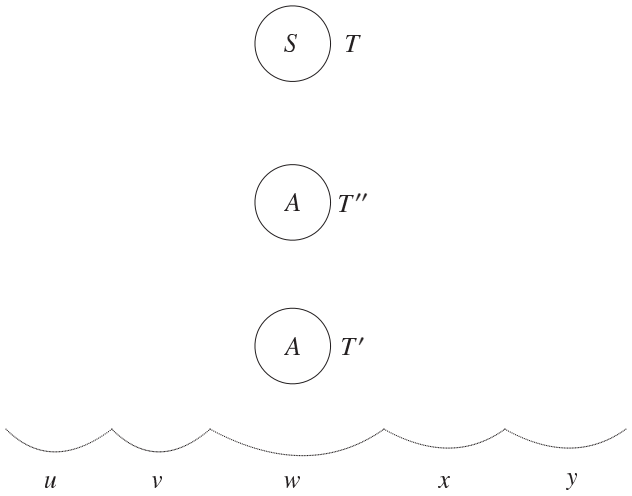


Figure 2.36. Decomposition of tree in the proof of pumping lemma.

$z \in L(G)$, such that $|z| > n$. Then there exists a derivation tree T with the result z. Let h be the height of T (the maximum of path lengths from root to leaves). Because in T all interior vertices have at most ℓ descendents, T has at most ℓ^h leaves, i.e. $|z| \leq \ell^h$. On the other hand, because of $|z| > \ell^{m+1}$, we get that $h > m+1$. From this follows that in derivation tree T there is a path from root to a leave in which there are more than $(m+1)$ vertices. Consider such a path. Because in G the number of nonterminals is m and on this path vertices different from the leaf are labelled with nonterminals, by the pigeonhole principle, it must be a nonterminal on this path which occurs at least twice.

Let us denote by A the nonterminal being the first on this path from root to the leaf which firstly repeat. Denote by T' the subtree, which root is this occurrence of A. Similarly, denote by T'' the subtree, which root is the second occurrence of A on this path. Let w be the result of the tree T'. Then the result of T'' is in form vwx, while of T in $uvwxy$. Derivation tree T with this decomposition of z can be seen in Fig. 2.36. We show that this decomposition of z satisfies conditions (1)–(4) of lemma.

Because in P there are no ε-productions (except maybe the case $S \to \varepsilon$), we have $|w| \geq 1$. Furthermore, because each interior vertex of the derivation tree has at least two descendents (namely there are no unit productions), also the root of T'' has, hence $|vx| \geq 1$. Because A is the first repeated nonterminal on this path, the height of T'' is at most $m+1$, and from this $|vwx| \leq \ell^{m+1} = n$ results.

After eliminating from T all vertices of T'' excepting the root, the result of obtained tree is uAy, i.e. $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$.

Similarly, after eliminating T' we get $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$, and finally because of the definition of T' we get $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Then $S \overset{*}{\underset{G}{\Longrightarrow}} uAy$, $A \overset{*}{\underset{G}{\Longrightarrow}} vAx$ and $A \overset{*}{\underset{G}{\Longrightarrow}} w$. Therefore $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uwy$ and $S \overset{*}{\underset{G}{\Longrightarrow}} uAy \overset{*}{\underset{G}{\Longrightarrow}} uvAxy \overset{*}{\underset{G}{\Longrightarrow}} \ldots \overset{*}{\underset{G}{\Longrightarrow}} uv^iAx^iy \overset{*}{\underset{G}{\Longrightarrow}} uv^iwx^iy$ for all

$i \geq 1$. Therefore, for all $i \geq 0$ we have $S \overset{*}{\Longrightarrow} uv^i wx^i y$, i.e. for all $i \geq 0$ $uv^i wx^i y \in L(G)$. ■

Now we present two consequences of the lemma.

Corollary 2.30 $\mathcal{L}_2 \subset \mathcal{L}_1$.

Proof. This consequence states that there exists a context-sensitive language which is not context-free. To prove this it is sufficient to find a context-sensitive language for which the lemma is not true. Let this language be $L = \{a^m b^m c^m \mid m \geq 1\}$.

To show that this language is context-sensitive it is enough to give a convenient grammar. In Example 2.2. both grammars are extended context-sensitive, and we know that to each extended grammar of type i an equivalent grammar of the same type can be associated.

Let n be the natural number associated to L by lemma, and consider the word $z = a^n b^n c^n$. Because of $|z| = 3n > n$, if L is context-free z can be decomposed in $z = uvwxy$ such that conditions (1)–(4) are true. We show that this leads us to a contradiction.

Firstly, we will show that word v and x can contain only one type of letters. Indeed if either v or x contain more than one type of letters, then in word $uvvwxxy$ the order of the letters will be not the order a, b, c, so $uvvwxxy \notin L(G)$, which contradicts condition (4) of lemma.

If both v and x contain at most one type of letters, then in word uwy the number of different letters will be not the same, so $uwy \notin L(G)$. This also contradicts condition (4) in lemma. Therefore L is not context-free. ■

Corollary 2.31 *The class of context-free languages is not closed under the intersection.*

Proof. We give two context-free languages which intersection is not context-free. Let $N = \{S, A, B\}$, $T = \{a, b, c\}$ and

$G_1 = (N, T, P_1, S)$ where P_1 :
$S \to AB$,
$A \to aAb \mid ab$,
$B \to cB \mid c$,

and $G_2 = (N, T, P_2, S)$, where P_2 :
$S \to AB$,
$A \to Aa \mid a$,
$B \to bBc \mid bc$.

Languages $L(G_1) = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$ and $L(G_2) = \{a^n b^m c^m \mid n \geq 1, m \geq 1\}$ are context-free. But

$$L(G_1) \cap L(G_2) = \{a^n b^n c^n \mid n \geq 1\}$$

is not context-free (see the proof of the Consequence 2.30). ■

2.3.4. Normal forms of the context-free languages

In the case of arbitrary grammars the normal form was defined (see page 68) as grammars with no terminals in the left-hand side of productions. The normal form in the case of the context-free languages will contains some restrictions on the right-hand sides of productions. Two normal forms (Chomsky and Greibach) will be discussed.

Chomsky normal form

Definition 2.32 *A context-free grammar $G = (N, T, P, S)$ is in Chomsky normal form, if all productions have form $A \to a$ or $A \to BC$, where $A, B, C \in N$, $a \in T$.*

Example 2.31 Grammar $G = (\{S, A, B, C\}, \{a, b\}, \{S \to AB,\ S \to CB,\ C \to AS,\ A \to a,\ B \to b\}, S)$ is in Chomsky normal form and $L(G) = \{a^n b^n \mid n \geq 1\}$.

To each ε-free context-free language can be associated an equivalent grammar is Chomsky normal form. The next algorithm transforms an ε-free context-free grammar $G = (N, T, P, S)$ in grammar $G' = (N', T, P', S)$ which is in Chomsky normal form.

CHOMSKY-NORMAL-FORM(G,G')

1 $N' \leftarrow N$
2 eliminate unit productions, and let P' the new set of productions
(see algorithm ELIMINATE-UNIT-PRODUCTIONS on page 67)
3 in P' replace in each production with at least two letters in right-hand side
all terminals a by a new nonterminal A, and add this nonterminal to N'
and add production $A \to a$ to P'
4 replace all productions $B \to A_1 A_2 \ldots A_k$, where $k \geq 3$ and $A_1, A_2, \ldots, A_k \in N$,
by the following:

$$\begin{aligned} B &\to A_1 C_1, \\ C_1 &\to A_2 C_2, \\ &\ldots \\ C_{k-3} &\to A_{k-2} C_{k-2}, \\ C_{k-2} &\to A_{k-1} A_k, \end{aligned}$$

where $C_1, C_2, \ldots, C_{k-2}$ are new nonterminals, and add them to N'.

Example 2.32 Let $G = (\{S, D\}, \{a, b, c\}, \{S \to aSc,\ S \to D,\ D \to bD,\ D \to b\}, S)$. It is easy to see that $L(G) = \{a^n b^m c^n \mid n \geq 0, m \geq 1\}$. Steps of transformation to Chomsky normal form are the following:

Step 1: $N' = \{S, D\}$
Step 2: After eliminating the unit production $S \to D$ the productions are:

$$\begin{aligned} S &\to aSc \mid bD \mid b, \\ D &\to bD \mid b. \end{aligned}$$

Step 3: We introduce three new nonterminals because of the three terminals in productions. Let these be A, B, C. Then the production are:

$$\begin{aligned} S &\to ASC \mid BD \mid b, \\ D &\to BD \mid b, \\ A &\to a, \\ B &\to b, \\ C &\to c. \end{aligned}$$

Step 4: Only one new nonterminal (let this E) must be introduced because of a single production with three letters in the right-hand side. Therefore $N' = \{S, A, B, C, D, E\}$, and the productions in P' are:

$$\begin{aligned} S &\to AE \mid BD \mid b, \\ D &\to BD \mid b, \end{aligned}$$

$A \to a$,
$B \to b$,
$C \to c$,
$E \to SC$.
All these productions are in required form.

Greibach normal form

Definition 2.33 *A context-free grammar $G = (N, T, P, S)$ is in **Greibach normal form** if all production are in the form $A \to aw$, where $A \in N$, $a \in T$, $w \in N^*$.*

Example 2.33 Grammar $G = (\{S, B\}, \{a, b\}, \{S \to aB,\ S \to aSB,\ B \to b\}, S)$ is in Greibach normal form and $L(G) = \{a^n b^n \mid n \geq 1\}$.

To each ε-free context-free grammar an equivalent grammar in Greibach normal form can be given. We give and algorithm which transforms a context-free grammar $G = (N, T, P, S)$ in Chomsky normal form in a grammar $G' = (N', T, P', S)$ in Greibach normal form.

First, we give an order of the nonterminals: $A_1, A_2, \ldots, A_n$, where A_1 is the start symbol. The algorithm will use the notations $x \in N'^+$, $\alpha \in TN'^* \cup N'^+$.

GREIBACH-NORMAL-FORM(G,G')

1 $N' \leftarrow N$
2 $P' \leftarrow P$
3 **for** $i \leftarrow 2$ **to** n ▷ Case $A_i \to A_j x,\ j < i$
4 **do for** $j \leftarrow 1$ **to** $i - 1$
5 **do** for all productions $A_i \to A_j x$ and $A_j \to \alpha$ (where α has no A_j as first letter)
in P' productions $A_i \to \alpha x$,
delete from P' productions $A_i \to A_j x$
6 **if** there is a production $A_i \to A_i x$ ▷ Case $A_i \to A_i x$
7 **then** put in N' the new nonterminal B_i,
for all productions $A_i \to A_i x$ put in P' productions $B_i \to xB_i$ and $B_i \to x$,
delete from P' production $A_i \to A_i x$,
for all production $A_i \to \alpha$ (where A_i is not the first letter of α)
put in P' production $A_i \to \alpha B_i$
8 **for** $i \leftarrow n - 1$ **downto** 1 ▷ Case $A_i \to A_j x,\ j > i$
9 **do for** $j \leftarrow i + 1$ **to** n
10 **do** for all productions $A_i \to A_j x$ and $A_j \to \alpha$
put in P' production $A_i \to \alpha x$ and
delete from P' productions $A_i \to A_j x$,
11 **for** $i \leftarrow 1$ **to** n ▷ Case $B_i \to A_j x$
12 **do for** $j \leftarrow 1$ **to** n
13 **do** for all productions $B_i \to A_j x$ and $A_j \to \alpha$
put in P' production $B_i \to \alpha x$ and
delete from P' productions $B_i \to A_j x$

The algorithm first transform productions of the form $A_i \to A_j x,\ \ j < i$ such that $A_i \to A_j x,\ \ j \geq i$ or $A_i \to \alpha$, where this latter is in Greibach normal form. After this, introducing a new nonterminal, eliminate productions $A_i \to A_i x$, and using substitutions all production of the form $A_i \to A_j x,\ \ j > i$ and $B_i \to A_j x$ will be transformed in Greibach normal form.

Example 2.34 Transform productions in Chomsky normal form

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to A_2A_3 \mid a$
$A_3 \to A_2A_4 \mid b$
$A_4 \to c$

in Greibach normal form.

Steps of the algorithm:

3–5: Production $A_3 \to A_2A_4$ must be transformed. For this production $A_2 \to a$ is appropriate. Put $A_3 \to aA_4$ in the set of productions and eliminate $A_3 \to A_2A_4$.

The productions will be:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to A_2A_3 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$

6-7: Elimination of production $A_2 \to A_2A_3$ will be made using productions:

$B_2 \to A_3B_2$
$B_2 \to A_3$
$A_2 \to aB_2$

Then, after steps 6–7. the productions will be:

$A_1 \to A_2A_3 \mid A_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to A_3B_2 \mid A_3$

8–10: We make substitutions in productions with A_1 in left-hand side. The results is:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$

11–13: Similarly with productions with B_2 in left-hand side:

$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$

After the elimination in steps 8–13 of productions in which substitutions were made, the following productions, which are now in Greibach normal form, result:

$A_1 \to aA_3 \mid aB_2A_3 \mid aA_4 \mid aB_2A_4$
$A_2 \to aB_2 \mid a$
$A_3 \to aA_4 \mid b$
$A_4 \to c$
$B_2 \to aA_4B_2 \mid aA_3A_4B_2 \mid aA_4 \mid aA_3A_4$

Example 2.35 Language

$$L = \{a^n b^k c^{n+k} \mid n \geq 0, k \geq 0, n + k > 0\}$$

can be generated by grammar

$$G = \{\{S, R\}, \{a, b, c\}, \{S \to aSc,\ S \to ac,\ S \to R, R \to bRc,\ R \to bc\}, S\}$$

First, will eliminate the single unit production, and after this we will give an equivalent grammar in Chomsky normal form, which will be transformed in Greibach normal form.

Productions after the elimination of production $S \to R$:

$S \to aSc \mid ac \mid bRc \mid bc$

$R \to bRc \mid bc.$

We introduce productions $A \to a, B \to b, C \to c$, and replace terminals by the corresponding nonterminals:

$S \to ASC \mid AC \mid BRC \mid BC,$

$R \to BRC \mid BC,$

$A \to a,\ B \to b,\ C \to c.$

After introducing two new nonterminals (D, E):

$S \to AD \mid AC \mid BE \mid BC,$

$D \to SC,$

$E \to RC,$

$R \to BE \mid BC,$

$A \to a,\ B \to b,\ C \to c.$

This is now in Chomsky normal form. Replace the nonterminals to be letters A_i as in the algorithm. Then, after applying the replacements

S replaced by A_1, A replaced by A_2, B replaced by A_3, C replaced by A_4, D replaced by A_5, E replaced by A_6, R replaced by A_7,

our grammar will have the productions:

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4,$

$A_2 \to a,\ A_3 \to b,\ A_4 \to c,$

$A_5 \to A_1A_4,$

$A_6 \to A_7A_4,$

$A_7 \to A_3A_6 \mid A_3A_4.$

In steps 3–5 of the algorithm the new productions will occur:

$A_5 \to A_2A_5A_4 \mid A_2A_4A_4 \mid A_3A_6A_4 \mid A_3A_4A_4$ then

$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$

$A_7 \to A_3A_6 \mid A_3A_4$, then

$A_7 \to bA_6 \mid bA_4.$

Therefore

$A_1 \to A_2A_5 \mid A_2A_4 \mid A_3A_6 \mid A_3A_4,$

$A_2 \to a,\ A_3 \to b,\ A_4 \to c,$

$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$

$A_6 \to A_7A_4,$

$A_7 \to bA_6 \mid bA_4.$

Steps 6–7 will be skipped, because we have no left-recursive productions. In steps 8–10 after the appropriate substitutions we have:

$A_1 \to aA_5 \mid aA_4 \mid bA_6 \mid bA_4,$

$A_2 \to a,$

$A_3 \to b,$

$A_4 \to c,$

$A_5 \to aA_5A_4 \mid aA_4A_4 \mid bA_6A_4 \mid bA_4A_4$

$A_6 \to bA_6A_4 \mid bA_4A_4,$

$A_7 \to bA_6 \mid bA_4.$

Exercises

2.3-1 Give pushdown automata to accept the following languages:

$L_1 = \{a^n c b^n \mid n \geq 0\},$

$L_2 = \{a^n b^{2n} \mid n \geq 1\},$

$L_3 = \{a^{2n}b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$,

2.3-2 Give a context-free grammar to generate language $L = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$, and transform it in Chomsky and Greibach normal forms. Give a pushdown automaton which accepts L.

2.3-3 What languages are generated by the following context-free grammars?

$G_1 = (\{S\}, \{a, b\}, \{S \to SSa, \to b\}, S)$, $G_2 = (\{S\}, \{a, b\}, \{S \to SaS, \to b\}, S)$

2.3-4 Give a context-free grammar to generate words with an equal number of letters a and b.

2.3-5 Prove, using the pumping lemma, that a language which words contains an equal number of letters a, b and c can not be context-free.

2.3-6 Let the grammar $G = (V, T, P, S)$, where

$V = \{S\}$,

$T = \{$*if, then, else, a, c*$\}$,

$P = \{S \to$ *if a then* S, $S \to$ *if a then* S *else* S, $S \to c\}$,

Show that word *if a then if a then c else c* has two different leftmost derivations.

2.3-7 Prove that if L is context-free, then $L^{-1} = \{u^{-1} \mid u \in L\}$ is also context-free.

Problems

2-1. Linear grammars

A grammar $G = (N, T, P, S)$ which has productions only in the form $A \to u_1 B u_2$ or $A \to u$, where $A, B \in N$, $u, u_1, u_2 \in T^*$, is called a ***linear grammar***. If in a linear grammar all production are of the form $A \to Bu$ or $A \to v$, then it is called a left-linear grammar. Prove that the language generated by a left-linear grammar is regular.

2-2. Operator grammars

An ε-free context-free grammar is called ***operator grammar*** if in the right-hand side of productions there are no two successive nonterminals. Show that, for all ε-free context-free grammar an equivalent operator grammar can be built.

2-3. Complement of context-free languages

Prove that the class of context-free languages is not closed on complement.

Chapter notes

In the definition of finite automata instead of transition function we have used the transition graph, which in many cases help us to give simpler proofs.

There exist a lot of classical books on automata and formal languages. We mention from these the following: two books of Aho and Ullman [1, 2] in 1972 and 1973, book of Gécseg and Peák [17] in 1972, two books of Salomaa [53, 54] in 1969 and 1973, a book of Hopcroft and Ullman [26] in 1979, a book of Harrison [24] in 1978, a book of Manna [41], which in 1981 was published also in Hungarian. We notice also a book of Sipser [60] in 1997 and a monograph of Rozenberg and Salomaa [52]. In a book of Lothaire (common name of French authors) [38] on combinatorics of words we can read on other types of automata. Paper of Giammarresi and Montalbano [21] generalise the notion of finite automata. A new

monograph is of Hopcroft, Motwani and Ullman [25]. In German we recommend the student book of Asteroth and Baier [5]. The concise description of the transformation in Greibach normal form is based on this book.

Other books in English: : [7, 9, 12, 28, 31, 34, 37, 42, 44, 58, 59, 65, 66].

At the end of the chapter on compilers another books on the subject are mentioned.

3. Complexity

4. Cryptography

5. Recurrences

The recursive definition of the Fibonacci numbers is well-known: if F_n is the n^{th} Fibonacci number, then

$$F_0 = 0, \quad F_1 = 1 \ ,$$

$$F_{n+2} = F_{n+1} + F_n, \qquad \text{if } \ n \geq 0 \ .$$

We are interested in an explicit form of the numbers F_n for all n natural numbers. Actually, the problem is to solve an equation where the unknown is given recursively, in which case the equation is called a ***recurrence equation***. The solution can be considered as a function over natural numbers, because F_n is defined for all n. Such recurrence equations are also known as ***difference equations***, but could be named as ***discrete differential equations*** for their similarities to differential equations.

Definition 5.1 *A* ***k*th order recurrence equation**, $(k \geq 1)$ *is an equation of the form*

$$f(x_n, x_{n+1}, \ldots, , x_{n+k}) = 0, \qquad n \geq 0, \tag{5.1}$$

where x_n must be given in an explicit form.

For a unique determination of x_n, k initial values must be given. Usually these values are $x_0, x_1, \ldots, x_{k-1}$. These can be considered as ***initial conditions***. In the case of the equation for Fibonacci-numbers, which is of second order, two initial values must be given.

The sequence $x_n = g(n)$ satisfying equation (5.1) and the corresponding initial conditions is called a ***particular solution***. If all particular solutions of equation (5.1) can be obtained from the sequence $x_n = h(n, C_1, C_2, \ldots, C_k)$, by adequately choosing of the constants $C_1, C_2, \ldots, C_k$, then this sequence x is a ***general solution***.

Solving recurrence equations is not an easy task. In the chapter we will discuss methods which can be used in special cases. For simplicity of writing we will use the notation x_n instead of $x(n)$ as it appears in several books (sequences can be considered as functions over natural numbers).

The chapter is divided into three sections. In section 5.1 we deal with solving linear recurrence equations, in section 5.2 with generating functions and their use in solving recurrence equations and in section 5.3 we focus our attention on numerical solution of recurrence equations.

5.1. Linear recurrence equations

If the recurrence equation is of the form

$$f_0(n)x_n + f_1(n)x_{n+1} + \cdots + f_k(n)x_{n+k} = f(n), \quad n \geq 0,$$

where $f, f_0, f_1, \ldots, f_k$ are functions defined over natural numbers, $f_0, f_k \neq 0$, and x_n must be given explicitly, then the recurrence equation is ***linear.*** If f is the zero function, then the equation is ***homogeneous,*** otherwise ***nonhomogeneous.*** If all the functions $f_0, f_1, \ldots, f_k$ are constant, the equation is called a ***linear recurrence equation with constant coefficients.***

5.1.1. Linear homogeneous equations with constant coefficients

Let the equation be

$$a_0x_n + a_1x_{n+1} + \cdots + a_kx_{n+k} = 0, \qquad n \geq k\ , \tag{5.2}$$

where $a_0, a_1, \ldots, a_k$ are real constants, $a_0, a_k \neq 0,\ k \geq 1$. If k initial conditions are given (usually $x_0, x_1, \ldots, x_{k-1}$), then the general solution of this equation can be uniquely given.

To solve the equation let us consider its ***characteristic equation***

$$a_0 + a_1r + \cdots + a_{k-1}r^{k-1} + a_kr^k = 0\ , \tag{5.3}$$

a polynomial equation with real coefficients. This equation has k roots in the field of complex numbers. It can easily be seen after a simple substitution that if r_0 is a real solution of the characteristic equation, then $C_0r_0^n$ is a solution of (5.2), for arbitrary C_0.

The general solution of equation (5.2) is

$$x_n = C_1x_n^{(1)} + C_2x_n^{(2)} + \cdots + C_kx_n^{(k)}\ ,$$

where $x_n^{(i)}$ $(i = 1, 2, \ldots, k)$ are the linearly independent solutions of equation (5.2). The constants $C_1, C_2, \ldots, C_k$ can be determined from the initial conditions by solving a system of k equations.

The linearly independent solutions are supplied by the roots of the characteristic equation by the following way. A ***fundamental solution*** of equation (5.2) can be associated with each root of the characteristic equation. Let us consider the following cases.

Distinct real roots

Let $r_1, r_2, \ldots, r_p$ be distinct real roots of the characteristic equation. Then

$$r_1^n,\ r_2^n,\ \ldots,\ r_p^n$$

are solutions of equation (5.2), and

$$C_1r_1^n + C_2r_2^n + \cdots + C_pr_p^n \tag{5.4}$$

is also a solution, for arbitrary constants $C_1, C_2, \ldots, C_p$. If $p = k$, then (5.4) is the general solution of the recurrence equation.

Example 5.1 Solve the recurrence equation

$$x_{n+2} = x_{n+1} + x_n, \quad x_0 = 0, \ x_1 = 1.$$

The corresponding characteristic equation is

$$r^2 - r - 1 = 0 \, ,$$

with the solutions

$$r_1 = \frac{1 + \sqrt{5}}{2} \, , \quad r_2 = \frac{1 - \sqrt{5}}{2} \, .$$

These are distinct real solutions, so the general solution of the equation is

$$x_n = C_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + C_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n \, .$$

The constants C_1 and C_2 can be determined using the initial conditions. From $x_0 = 0$, $x_1 = 1$ the following system of equations can be obtained.

$$\begin{aligned} C_1 + C_2 &= 0 \, , \\ C_1 \frac{1 + \sqrt{5}}{2} + C_2 \frac{1 - \sqrt{5}}{2} &= 1 \, . \end{aligned}$$

The solution of this system of equations is $C_1 = 1/\sqrt{5}$, $C_2 = -1/\sqrt{5}$. Therefore the general solution is

$$x_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n ,$$

which is the nth Fibonacci number F_n.

Multiple real roots

Let r be a real root of the characteristic equation with multiplicity p. Then

$$r^n, \ nr^n, \ n^2 r^n, \ \ldots, \ n^{p-1} r^n$$

are solutions of equation (5.2) (fundamental solutions corresponding to r), and

$$(C_0 + C_1 n + C_2 n^2 + \cdots + C_{p-1} n^{p-1}) r^n \tag{5.5}$$

is also a solution, for any constants $C_0, C_1, \ldots, C_{p-1}$. If the characteristic equation has no other solutions, then (5.5) is a general solution of the recurrence equation.

Example 5.2 Solve the recurrence equation

$$x_{n+2} = 4x_{n+1} - 4x_n, \quad x_0 = 1, \ x_1 = 3.$$

The characteristic equation is

$$r^2 - 4r + 4 = 0 \, ,$$

with $r = 2$ a solution with multiplicity 2. Then

$$x_n = (C_0 + C_1 n) 2^n$$

is a general solution of the recurrence equation.

From the initial conditions we have

$$\begin{aligned} C_0 &= 1\,, \\ 2C_0 + 2C_1 &= 3\,. \end{aligned}$$

From this system of equations $C_0 = 1$, $C_1 = 1/2$, so the general solution is

$$x_n = \left(1 + \frac{1}{2}n\right)2^n \quad \text{or} \quad x_n = (n+2)2^{n-1}\,.$$

Distinct complex roots

If the complex number $a(\cos b + i \sin b)$, written in trigonometric form, is a root of the characteristic equation, then its conjugate $a(\cos b - i \sin b)$ is also a root, because the coefficients of the characteristic equation are real numbers. Then

$$a^n \cos bn \quad \text{and} \quad a^n \sin bn$$

are solutions of equation (5.2) and

$$C_1 a^n \cos bn + C_2 a^n \sin bn \tag{5.6}$$

is also a solution, for any constants C_1 and C_2. If these are the only solutions of the characteristic equation, then (5.6) is a general solution.

Example 5.3 Solve the recurrence equation

$$x_{n+2} = 2x_{n+1} - 2x_n, \quad x_0 = 0,\ x_1 = 1.$$

The corresponding characteristic equation is

$$r^2 - 2r + 2 = 0\,,$$

with roots $1 + i$ and $1 - i$. These can be written in trigonometric form as $\sqrt{2}(\cos(\pi/4) + i \sin(\pi/4))$ and $\sqrt{2}(\cos(\pi/4) - i \sin(\pi/4))$. Therefore

$$x_n = C_1(\sqrt{2})^n \cos\frac{n\pi}{4} + C_2(\sqrt{2})^n \sin\frac{n\pi}{4}$$

is a general solution of the recurrence equation. From the initial conditions

$$\begin{aligned} C_1 &= 0\,, \\ C_1\sqrt{2}\cos\frac{\pi}{4} + C_2\sqrt{2}\sin\frac{\pi}{4} &= 1. \end{aligned}$$

Therefore $C_1 = 0,\ C_2 = 1$. Hence the general solution is

$$x_n = (\sqrt{2})^n \sin\frac{n\pi}{4}\,.$$

Multiple complex roots

If the complex number written in trigonometric form as $a(\cos b + i \sin b)$ is a root of the characteristic equation with multiplicity p, then its conjugate $a(\cos b - i \sin b)$ is also a root with multiplicity p.

Then

$$a^n \cos bn,\ na^n \cos bn,\ \ldots,\ n^{p-1}a^n \cos bn$$

and

$$a^n \sin bn,\ na^n \sin bn,\ \ldots,\ n^{p-1}a^n \sin bn$$

are solutions of the recurrence equation (5.2). Then

$$(C_0 + C_1 n + \cdots + C_{p-1}n^{p-1})a^n \cos bn + (D_0 + D_1 n + \cdots + D_{p-1}n^{p-1})a^n \sin bn$$

is also a solution, where $C_0, C_1, \ldots, C_{p-1}, D_0, D_1, \ldots, D_{p-1}$ are arbitrary constants, which can be determined from the initial conditions. This solution is general if the characteristic equation has no other roots.

Example 5.4 Solve the recurrence equation

$$x_{n+4} + 2x_{n+2} + x_n = 0,\quad x_0 = 0,\ x_1 = 1,\ x_2 = 2,\ x_3 = 3.$$

The characteristic equation is

$$r^4 + 2r^2 + 1 = 0\ ,$$

which can be written as $(r^2 + 1)^2 = 0$. The complex numbers i and $-i$ are double roots. The trigonometric form of these are

$$i = \cos\frac{\pi}{2} + i \sin\frac{\pi}{2},\quad \text{and}\quad -i = \cos\frac{\pi}{2} - i \sin\frac{\pi}{2}$$

respectively. Therefore the general solution is

$$x_n = (C_0 + C_1 n)\cos\frac{n\pi}{2} + (D_0 + D_1 n)\sin\frac{n\pi}{2}\ .$$

From the initial conditions we obtain

$$\begin{aligned}
C_0 &= 0\ ,\\
(C_0 + C_1)\cos\frac{\pi}{2} + (D_0 + D_1)\sin\frac{\pi}{2} &= 1\ ,\\
(C_0 + 2C_1)\cos\pi + (D_0 + 2D_1)\sin\pi &= 2\ ,\\
(C_0 + 3C_1)\cos\frac{3\pi}{2} + (D_0 + 3D_1)\sin\frac{3\pi}{2} &= 3\ ,
\end{aligned}$$

that is

$$\begin{aligned}
C_0 &= 0\ ,\\
D_0 + D_1 &= 1\ ,\\
-2C_1 &= 2\ ,\\
-D_0 - 3D_1 &= 3\ .
\end{aligned}$$

Solving this system of equations $C_0 = 0$, $C_1 = -1$, $D_0 = 3$ and $D_1 = -2$. Thus the general solution is

$$x_n = (3 - 2n)\sin\frac{n\pi}{2} - n\cos\frac{n\pi}{2}\ .$$

Using these four cases all linear homogeneous equations with constant coefficients can be solved, if we can solve their characteristic equations.

Example 5.5 Solve the recurrence equation

$$x_{n+3} = 4x_{n+2} - 6x_{n+1} + 4x_n, \quad x_0 = 0,\ x_1 = 1,\ x_2 = 1.$$

The characteristic equation is

$$r^3 - 4r^2 + 6r - 4 = 0\ ,$$

with roots 2, $1 + i$ and $1 - i$. Therefore the general solution is

$$x_n = C_1 2^n + C_2(\sqrt{2})^n \cos\frac{n\pi}{4} + C_3(\sqrt{2})^n \sin\frac{n\pi}{4}\ .$$

After determining the constants we obtain

$$x_n = -2^{n-1} + \frac{(\sqrt{2})^n}{2}\left(\cos\frac{n\pi}{4} + 3\sin\frac{n\pi}{4}\right)\ .$$

The general solution

The characteristic equation of the kth order linear homogeneous equation (5.2) has k roots in th field of complex numbers, which are not necessarily distinct. Let these roots be the following:

r_1 real, with multiplicity p_1 $(p_1 \geq 1)$,
r_2 real, with multiplicity p_2 $(p_2 \geq 1)$,
...
r_t real, with multiplicity p_t $(p_t \geq 1)$,
$s_1 = a_1(\cos b_1 + i \sin b_1)$ complex, with multiplicity q_1 $(q_1 \geq 1)$,
$s_2 = a_2(\cos b_2 + i \sin b_2)$ complex, with multiplicity q_2 $(q_2 \geq 1)$,
...
$s_m = a_m(\cos b_m + i \sin b_m)$ complex, with multiplicity q_m $(q_m \geq 1)$.

Since the equation has k roots, $p_1 + p_2 + \cdots + p_t + 2(q_1 + q_2 + \cdots + q_m) = k$.
In this case the general solution of equation (5.2) is

$$\begin{aligned} x_n &= \sum_{j=1}^{t}\left(C_0^{(j)} + C_1^{(j)} n + \cdots + C_{p_j-1}^{(j)} n^{p_j-1}\right) r_j^n \\ &+ \sum_{j=1}^{m}\left(D_0^{(j)} + D_1^{(j)} n + \cdots + D_{q_j-1}^{(j)} n^{q_j-1}\right) a_j^n \cos b_j n \\ &+ \sum_{j=1}^{m}\left(E_0^{(j)} + E_1^{(j)} n + \cdots + E_{q_j-1}^{(j)} n^{q_j-1}\right) a_j^n \sin b_j n\ , \end{aligned} \tag{5.7}$$

where

$C_0^{(j)},\ C_1^{(j)},\ \ldots,\ C_{p_j-1}^{(j)},\ j = 1, 2, \ldots, t$,

$D_0^{(l)},\ E_0^{(l)},\ D_1^{(l)},\ E_1^{(l)},\ \ldots,\ D_{p_l-1}^{(l)},\ E_{p_l-1}^{(l)},\ l = 1, 2, \ldots, m$ are constants, which can be determined from the initial conditions.

The above statements can be summarised in the following theorem.

Theorem 5.1 *Let $k \geq 1$ be an integer and $a_0,\ a_1,\ \ldots, a_k$ real numbers with $a_0,\ a_k \neq 0$. The general solution of the linear recurrence equation (5.2) can be obtained as a linear combination of the terms $n^j r_i^n$, where r_i are the roots of the characteristic equation (5.3) with multiplicity p_i ($0 \leq j < p_i$) and the coefficients of the linear combination depend on the initial conditions.*

The proof of the theorem is left to the reader (see exercise 5.1-5.).

The algorithm for the general solution is the following.

LINEAR-HOMOGENEOUS

1 determine the characteristic equation of the recurrence equation
2 find all roots of the characteristic equation with their multiplicities
3 find the general solution (5.7) based on the roots
4 determine the constants of (5.7) using the initial conditions, if these exists.

5.1.2. Linear nonhomogeneous recurrence equations with constant coefficients

Consider the linear nonhomogeneous recurrence equation with constant coefficients

$$a_0 x_n + a_1 x_{n+1} + \cdots + a_k x_{n+k} = f(n)\ , \tag{5.8}$$

where $a_0, a_1, \ldots, a_k$ are real constants, $a_0, a_k \neq 0,\ k \geq 1$, and f is not the zero function.

The corresponding linear homogeneous equation (5.2) can be solved using Theorem 5.1. If a particular solution of equation (5.8) is known, then equation (5.8) can be solved.

Theorem 5.2 *Let $k \geq 1$ be an integer, $a_0,\ a_1,\ \ldots, a_k$ real numbers, $a_0,\ a_k \neq 0$. If $x_n^{(1)}$ is a particular solution of the linear nonhomogeneous equation (5.8) and $x_n^{(0)}$ is a general solution of the linear homogeneous equation (5.2), then*

$$x_n = x_n^{(0)} + x_n^{(1)}$$

is a general solution of the equation (5.8).

The proof of the theorem is left to the reader (see exercise 5.1-6.).

Example 5.6 Solve the recurrence equation

$$x_{n+2} + x_{n+1} - 2x_n = 2^n,\quad x_0 = 0,\ x_1 = 1.$$

First we solve the homogeneous equation

$$x_{n+2} + x_{n+1} - 2x_n = 0,$$

and obtain the general solution

$$x_n^{(0)} = C_1(-2)^n + C_2\ ,$$

since the roots of the characteristic equation are -2 and 1 . It is easy to see that

$$x_n = C_1(-2)^n + C_2 + 2^{n-2}$$

$f(n)$	$x_n^{(1)}$
$n^p a^n$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n$
$a^n n^p \sin bn$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n \sin bn + (D_0 + D_1 n + \cdots + D_p n^p)a^n \cos bn$
$a^n n^p \cos bn$	$(C_0 + C_1 n + \cdots + C_p n^p)a^n \sin bn + (D_0 + D_1 n + \cdots + D_p n^p)a^n \cos bn$

Figure 5.1. The form of particular solutions.

is a solution of the nonhomogeneous equation. Therefore the general solution is

$$x_n = -\frac{1}{4}(-2)^n + 2^{n-2} \quad \text{or} \quad x_n = \frac{2^n - (-2)^n}{4} ,$$

The constants C_1 and C_2 can be determined using the initial conditions. Thus, that is

$$x_n = \begin{cases} 0, & \text{if } n \text{ is even} , \\ 2^{n-1}, & \text{if } n \text{ is odd} . \end{cases}$$

A particular solution can be obtained using the *method of variation of constants*. However, there are cases when there is an easier way of finding a particular solution. In figure 5.1 we can see types of functions $f(n)$, for which a particular solution $x_n^{(1)}$ can be obtained in the given form in the table. The constants can be obtained by substitutions.

In the previous example $f(n) = 2^n$, so the first case can be used with $a = 2$ and $p = 0$. Therefore we try to find a particular solution of the form $C_0 2^n$. After substitution we obtain $C_0 = 1/4$, thus the particular solution is

$$x_n^{(1)} = 2^{n-2} .$$

Exercises

5.1-1 Solve the recurrence equation

$$H_n = 2H_{n-1} + 1, \quad \text{ha } n \geq 1, \text{ és} \quad H_0 = 0 .$$

(Here H_n is the optimal number of moves in the problem of Towers of Hanoi.)

5.1-2 Analyse the problem of Towers of Hanoi if n discs have to be moved from stick A to stick C in such a way that no disc can be moved *directly* from A to C and vice versa.

Hint. Show that if the optimal number of moves is denoted by M_n, and $n \geq 1$, then $M_n = 3M_{n-1} + 2$.

5.1-3 Solve the recurrence equation

$$(n+1)R_n = 2(2n-1)R_{n-1}, \quad \text{ha } n \geq 1, \text{ és} \quad R_0 = 1.$$

5.1-4 Solve the linear nonhomogeneous recurrence equation

$$x_n = 2^n - 2 + 2x_{n-1}, \quad \text{ha } n \geq 2, \quad \text{és} \quad x_1 = 0.$$

Hint. Try to find a particular solution of the form $C_1 n 2^n + C_2$.

5.1-5⋆ Prove Theorem 5.1.
5.1-6 Prove Theorem 5.2.

5.2. Generating functions and recurrence equations

Generating functions can be used, among others, to solve recurrence equations, count objects (e.g. binary trees), prove identities and solve partition problems. Counting the number of objects can be done by stating and solving recurrence equations. These equations are usually not linear, and generating functions can help us in solving them.

5.2.1. Definition and operations

Associate a series with the infinite sequence $(a_n)_{n\geq 0} = \langle a_0, a_1, a_2, \ldots, a_n, \ldots \rangle$ the following way

$$A(z) = a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n + \cdots = \sum_{n\geq 0} a_n z^n \ .$$

This is called the ***generating function*** of the sequence $(a_n)_{n\geq 0}$.

For example, in the case of the Fibonacci numbers this generating function is

$$F(z) = \sum_{n\geq 0} F_n z^n = z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + \cdots \ .$$

Multiplying both sides of the equation by z, then by z^2, we obtain

$$\begin{aligned} F(z) &= F_0 + F_1 z + F_2 z^2 + F_3 z^3 + \cdots + F_n z^n + \cdots \ , \\ zF(z) &= F_0 z + F_1 z^2 + F_2 z^3 + \cdots + F_{n-1} z^n + \cdots \ , \\ z^2 F(z) &= F_0 z^2 + F_1 z^3 + \cdots + F_{n-2} z^n + \cdots \ . \end{aligned}$$

If we subtract the second and the third equation from the first one term by term, then use the defining formula of the Fibonacci numbers, we get

$$F(z)(1 - z - z^2) = z \ ,$$

that is

$$F(z) = \frac{z}{1 - z - z^2} \ . \tag{5.9}$$

The correctness of these operations can be proved mathematically, but here we do not want to go into details. The formulae obtained using generating functions can usually also be proved using other methods.

Let us consider the following generating functions

$$A(z) = \sum_{n\geq 0} a_n z^n \text{ and } B(z) = \sum_{n\geq 0} b_n z^n.$$

The generating functions $A(z)$ and $B(z)$ are *equal*, if and only if $a_n = b_n$ for all n natural numbers.

Now we define the following operations with the generating functions: addition, multiplication by real number, shift, multiplication, derivation and integration.

Addition and multiplication by real number

$$\alpha A(z) + \beta B(z) = \sum_{n\geq 0} (\alpha a_n + \beta b_n) z^n \ .$$

Shift

The generating function

$$z^k A(z) = \sum_{n\geq 0} a_n z^{n+k} = \sum_{n\geq k} a_{n-k} z^n$$

represents the sequence $< \underbrace{0, 0, \ldots, 0}_{k}, a_0, a_1, \ldots >$, while the generating function

$$\frac{1}{z^k}(A(z) - a_0 - a_1 z - a_2 z^2 - \cdots - a_{k-1} z^{k-1}) = \sum_{n\geq k} a_n z^{n-k} = \sum_{n\geq 0} a_{k+n} z^n$$

represents the sequence $< a_k, a_{k+1}, a_{k+2}, \ldots >$.

Example 5.7 Let $A(z) = 1 + z + z^2 + \cdots$. Then

$$\frac{1}{z}\Big(A(z) - 1\Big) = A(z) \quad \text{and} \quad A(z) = \frac{1}{1-z} \ .$$

Multiplication

If $A(z)$ and $B(z)$ are generating functions, then

$$\begin{aligned} A(z)B(z) &= (a_0 + a_1 z + \cdots + a_n z^n + \cdots)(b_0 + b_1 z + \cdots + b_n z^n + \cdots) \\ &= a_0 b_0 + (a_0 b_1 + a_1 b_0) z + (a_0 b_2 + a_1 b_1 + a_2 b_0) z^2 + \cdots \\ &= \sum_{n\geq 0} s_n z^n, \end{aligned}$$

where $s_n = \sum_{k=0}^{n} a_k b_{n-k}$.

Special case. If $b_n = 1$ for all natural numbers n, then

$$A(z)\frac{1}{1-z} = \sum_{n\geq 0} \left(\sum_{k=0}^{n} a_k \right) z^n \ . \tag{5.10}$$

If, in addition, $a_n = 1$ for all n, then

$$\frac{1}{(1-z)^2} = \sum_{n\geq 0} (n+1)z^n \ . \tag{5.11}$$

Derivation

$$A'(z) = a_1 + 2a_2 z + 3a_3 z^2 + \cdots = \sum_{n\geq 0} (n+1)a_{n+1} z^n \ .$$

Example 5.8 After differentiating the both sides of the generating function

$$A(z) = \sum_{n\geq 0} z^n = \frac{1}{1-z},$$

we obtain

$$A'(z) = \sum_{n\geq 1} n z^{n-1} = \frac{1}{(1-z)^2} \ .$$

Integration

$$\int_0^z A(t)dt = a_0 z + \frac{1}{2}a_1 z^2 + \frac{1}{3}a_2 z^3 + \cdots = \sum_{n\geq 1} \frac{1}{n} a_{n-1} z^n \ .$$

Example 5.9 Let

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots$$

After integrating both sides we get

$$\ln \frac{1}{1-z} = z + \frac{1}{2}z^2 + \frac{1}{3}z^3 + \cdots = \sum_{n\geq 1} \frac{1}{n} z^n \ .$$

Multiplying the above generating functions we obtain

$$\frac{1}{1-z} \ln \frac{1}{1-z} = \sum_{n\geq 1} H_n z^n \ ,$$

where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$ $(H_0 = 0, \quad H_1 = 1)$ are the so-called *harmonic numbers*.

Changing the arguments

Let $A(z) = \sum_{n\geq 0} a_n z^n$ represent the sequence $< a_0, a_1, a_2, \ldots >$, then $A(cz) = \sum_{n\geq 0} c^n a_n z^n$ represents the sequence $< a_0, ca_1, c^2 a_2, \ldots c^n a_n, \ldots >$. The following statements holds

$$\frac{1}{2}\Big(A(z) + A(-z)\Big) = a_0 + a_2 z^2 + \cdots + a_{2n} z^{2n} + \cdots \ ,$$

$$\frac{1}{2}\Big(A(z) - A(-z)\Big) = a_1 z + a_3 z^3 + \cdots + a_{2n-1} z^{2n-1} + \cdots \ .$$

Example 5.10 Let $A(z) = 1 + z + z^2 + z^3 + \cdots = \dfrac{1}{1-z}$. Then

$$1 + z^2 + z^4 + \cdots = \frac{1}{2}(A(z) + A(-z)) = \frac{1}{2}\left(\frac{1}{1-z} + \frac{1}{1+z}\right) = \frac{1}{1-z^2} ,$$

which can also be obtained by substituting z with z^2 in $A(z)$. We can obtain the sum of the odd power terms the same way,

$$z + z^3 + z^5 + \cdots = \frac{1}{2}(A(z) - A(-z)) = \frac{1}{2}\left(\frac{1}{1-z} - \frac{1}{1+z}\right) = \frac{z}{1-z^2} .$$

Using generating functions we can obtain interesting formulae. For example, let $A(z) = 1/(1-z) = 1 + z + z^2 + z^3 + \cdots$. Then $zA\big(z(1+z)\big) = F(z)$, which is the generating function of the Fibonacci numbers. From this

$$zA\big(z(1+z)\big) = z + z^2(1+z) + z^3(1+z)^2 + z^4(1+z)^3 + \cdots .$$

The coefficient of z^{n+1} on the left-hand side is F_{n+1}, that is the $(n+1)$th Fibonacci number, while the coefficient of z^{n+1} on the right-hand side is

$$\sum_{k\geq 0}\binom{n-k}{k},$$

after using the binomial formula in each term. Hence

$$F_{n+1} = \sum_{k\geq 0}\binom{n-k}{k} = \sum_{k=0}^{\lfloor\frac{n+1}{2}\rfloor}\binom{n-k}{k}. \tag{5.12}$$

Remember that the binomial formula can be generalised for all real r, namely

$$(1+z)^r = \sum_{n\geq 0}\binom{r}{n}z^n ,$$

which is the generating function of the binomial coefficients. Here $\binom{r}{n}$ is a generalisation of the combinations for any real number r, that is

$$\binom{r}{n} = \begin{cases} \dfrac{r(r-1)(r-2)\ldots(r-n+1)}{n(n-1)\ldots 1}, & \text{if } n > 0 , \\ 1, & \text{fi } n = 0 , \\ 0, & \text{if } n < 0 . \end{cases}$$

We can obtain useful formulae using this generalisation for negative r. Let

$$\frac{1}{(1-z)^m} = (1-z)^{-m} = \sum_{k\geq 0}\binom{-m}{k}(-z)^k .$$

Since, by a simple computation, we get

$$\binom{-m}{k} = (-1)^k\binom{m+k-1}{k},$$

the following formula can be obtained

$$\frac{1}{(1-z)^{m+1}} = \sum_{k\geq 0} \binom{m+k}{k} z^k \ .$$

Then

$$\frac{z^m}{(1-z)^{m+1}} = \sum_{k\geq 0} \binom{m+k}{k} z^{m+k} = \sum_{k\geq 0} \binom{m+k}{m} z^{m+k} = \sum_{k\geq 0} \binom{k}{m} z^k \ ,$$

and

$$\sum_{k\geq 0} \binom{k}{m} z^k = \frac{z^m}{(1-z)^{m+1}} \ , \tag{5.13}$$

where m is a natural number.

5.2.2. Solving recurrence equations with generating functions

If the generating function of the general solution of a recurrence equation to be solved can be expanded in such a way that the coefficients are in closed form, then this method is successful.

Let the recurrence equation be

$$F(x_n, x_{n-1}, \ldots, x_{n-k}) = 0 \ . \tag{5.14}$$

To solve it, let us consider the generating function

$$X(z) = \sum_{n\geq 0} x_n z^n .$$

If (5.14) can be written as $G(X(z)) = 0$ and can be solved for $X(z)$, then $X(z)$ can be expanded into series in such a way that x_n can be written in closed form, equation (5.14) can be solved.

Now we give a general method for solving linear nonhomogeneous recurrence equations. After this we give three examples for the nonlinear case. In the first two examples the number of elements in some sets of binary trees, while in the third example the number of leaves of binary trees is computed. The corresponding recurrence equations (5.15), (5.17) and (5.18) will be solved using generating functions.

Linear nonhomogeneous recurrence equations with constant coefficients

Multiply both sides of equation (5.8) by z^n. Then

$$a_0 x_n z^n + a_1 x_{n+1} z^n + \cdots + a_k x_{n+k} z^n = f(n) z^n \ .$$

Summing up both sides of the equation term by term we get

$$a_0 \sum_{n\geq 0} x_n z^n + a_1 \sum_{n\geq 0} x_{n+1} z^n + \cdots + a_k \sum_{n\geq 0} x_{n+k} z^n = \sum_{n\geq 0} f(n) z^n \ .$$

Then

$$a_0 \sum_{n\geq 0} x_n z^n + \frac{a_1}{z} \sum_{n\geq 0} x_{n+1} z^{n+1} + \cdots + \frac{a_k}{z^k} \sum_{n\geq 0} x_{n+k} z^{n+k} = \sum_{n\geq 0} f(n) z^n \ .$$

Let

$$X(z) = \sum_{n\geq 0} x_n z^n \quad \text{and} \quad F(z) = \sum_{n\geq 0} f(n) z^n \ .$$

The equation can be written as

$$a_0 X(z) + \frac{a_1}{z}\Big(X(z) - x_0\Big) + \cdots + \frac{a_k}{z^k}\Big(X(z) - x_0 - x_1 z - \cdots - x_{k-1} z^{k-1}\Big) = F(z) \ .$$

This can be solved for $X(z)$. If $X(z)$ is a rational fraction, then it can be decomposed into partial (elementary) fractions which, after expanding them into series, will give us the general solution x_n of the original recurrence equation. We can also try to use the expansion into series in the case when the function is not a rational fraction.

Example 5.11 Solve the following equation using the above method

$$x_{n+1} - 2x_n = 2^{n+1} - 2, \text{ ha } n \geq 0 \quad \text{és } x_0 = 0 \ .$$

After multiplying and summing we have

$$\frac{1}{z}\sum_{n\geq 0} x_{n+1} z^{n+1} - 2\sum_{n\geq 0} x_n z^n = 2\sum_{n\geq 0} 2^n z^n - 2\sum_{n\geq 0} z^n \ ,$$

and

$$\frac{1}{z}\Big(X(z) - x_0\Big) - 2X(z) = \frac{2}{1-2z} - \frac{2}{1-z} \ .$$

Since $x_0 = 0$, after decomposing the right-hand side into partial fractions[1]), the solution of the equation is

$$X(z) = \frac{2z}{(1-2z)^2} + \frac{2}{1-z} - \frac{2}{1-2z} \ .$$

After differentiating the generating function

$$\frac{1}{1-2z} = \sum_{n\geq 0} 2^n z^n$$

term by term we get

$$\frac{2}{(1-2z)^2} = \sum_{n\geq 1} n 2^n z^{n-1} \ .$$

Thus

$$X(z) = \sum_{n\geq 0} n 2^n z^n + 2\sum_{n\geq 0} z^n - 2\sum_{n\geq 0} 2^n z^n = \sum_{n\geq 0} \Big((n-2)2^n + 2\Big) z^n \ ,$$

therefore

$$x_n = (n-2)2^n + 2 \ .$$

The number of binary trees

Let us denote by b_n the number of binary trees with n vertices. Then $b_1 = 1$, $b_2 = 2$, $b_3 = 5$ (see figure 5.2). Let $b_0 = 1$. (We will see later that this is a good choice.)

In a binary tree with n vertices, with the exception of the root, there are altogether $n-1$ vertices in the left and right subtrees. If the left subtree has k vertices and the right subtree has $n-1-k$ vertices, then there exists $b_k b_{n-1-k}$ such binary trees. Summing over

[1]For decomposing the fraction into partial fractions we can use the Undetermined Coefficients Method.

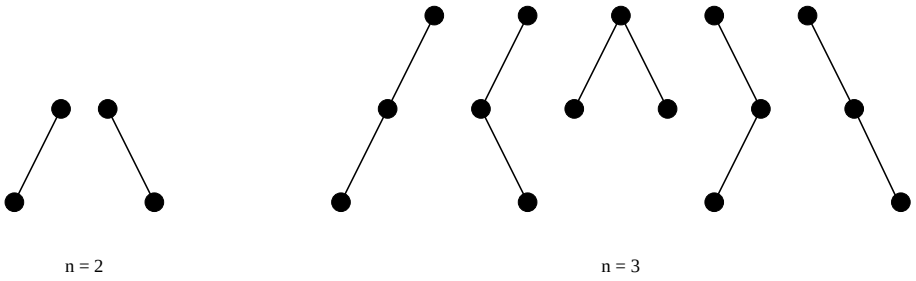


Figure 5.2. Binary trees with two and three vertices.

$k = 0, 1, \ldots, n-1$, we obtain exactly the number of binary trees, b_n. Thus for any natural number $n \geq 1$ the recurrence equation in b_n is

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-1} b_0 \ . \tag{5.15}$$

This can also be written as

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} \ .$$

Multiplying both sides by z^n, then summing over all n, we obtain

$$\sum_{n\geq 1} b_n z^n = \sum_{n\geq 1} \left(\sum_{k=0}^{n-1} b_k b_{n-1-k} \right) z^n \ . \tag{5.16}$$

Let $B(z) = \sum_{n\geq 0} b_n z^n$ be the generating function of the numbers b_n. The left-hand side of (5.16) is exactly $B(z) - 1$ (because $b_0 = 1$). The right-hand side looks like a product of two generating functions. To see which functions are in consideration, let us use the notation

$$A(z) = zB(z) = \sum_{n\geq 0} b_n z^{n+1} = \sum_{n\geq 1} b_{n-1} z^n \ .$$

Then the right-hand side of (5.16) is exactly $A(z)B(z)$, which is $zB^2(z)$. Therefore

$$B(z) - 1 = zB^2(z), \quad B(0) = 1 \ .$$

Solving this equation for $B(z)$ gives

$$B(z) = \frac{1 \pm \sqrt{1-4z}}{2z} \ .$$

We have to choose the negative sign because $B(0) = 1$. Thus

$$\begin{aligned} B(z) &= \frac{1}{2z}\left(1 - \sqrt{1-4z}\right) = \frac{1}{2z}\left(1 - (1-4z)^{1/2}\right) \\ &= \frac{1}{2z}\left(1 - \sum_{n\geq 0}\binom{1/2}{n}(-4z)^n\right) = \frac{1}{2z}\left(1 - \sum_{n\geq 0}\binom{1/2}{n}(-1)^n 2^{2n} z^n\right) \\ &= \frac{1}{2z} - \binom{1/2}{0}\frac{2^0 z^0}{2z} + \binom{1/2}{1}\frac{2^2 z}{2z} - \cdots - \binom{1/2}{n}(-1)^n \frac{2^{2n} z^n}{2z} + \cdots \\ &= \binom{1/2}{1}2 - \binom{1/2}{2}2^3 z + \cdots - \binom{1/2}{n}(-1)^n 2^{2n-1} z^{n-1} + \cdots \\ &= \sum_{n\geq 0}\binom{1/2}{n+1}(-1)^n 2^{2n+1} z^n = \sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n} z^n \ . \end{aligned}$$

Therefore $b_n = \dfrac{1}{n+1}\binom{2n}{n}$. The numbers b_n are also called the Catalan numbers.

Remark. In the previous computation we used the following formula that can be proved easily

$$\binom{1/2}{n+1} = \frac{(-1)^n}{2^{2n+1}(n+1)}\binom{2n}{n} .$$

The number of leaves of all binary trees of *n* vertices

Let us count the number of leaves (vertices with degree 1) in the set of all binary trees of n vertices. Denote this number by f_n. We remark that the root is not considered leaf even if it is of degree 1. It is easy to see that $f_2 = 2$, $f_3 = 6$. Let $f_0 = 0$ and $f_1 = 1$, conventionally. Later we will see that these values are good.

As in the case of numbering the binary trees, consider the binary trees of n vertices having k vertices in the left subtree and $n-k-1$ vertices in the right subtree. There are b_k such left subtrees and b_{n-1-k} right subtrees. If we consider such a left subtree and all such right subtrees, then together there are f_{n-1-k} leaves in the right subtrees. So for a given k there are $b_{n-1-k}f_k + b_k f_{n-1-k}$ leaves. After summing we have

$$f_n = \sum_{k=0}^{n-1}(f_k b_{n-1-k} + b_k f_{n-1-k}) \ .$$

By an easy computation we get

$$f_n = 2(f_0 b_{n-1} + f_1 b_{n-2} + \cdots + f_{n-1}b_0), \quad n \geq 2 \ . \tag{5.17}$$

This is a recurrence equation, the solution of which is f_n. Let

$$F(z) = \sum_{n\geq 0} f_n z^n \quad \text{and} \quad B(z) = \sum_{n\geq 0} b_n z^n \ .$$

Multiplying both sides of (5.17) by z^n and summing gives

$$\sum_{n\geq 2} f_n z^n = 2\sum_{n\geq 2}\left(\sum_{k=0}^{n-1} f_k b_{n-1-k}\right) z^n \ .$$

Since $f_0 = 0$ and $f_1 = 1$,

$$F(z) - z = 2zF(z)B(z) \ .$$

Thus

$$F(z) = \frac{z}{1 - 2zB(z)} \ ,$$

and since

$$B(z) = \frac{1}{2z}\left(1 - \sqrt{1-4z}\right) \ ,$$

we have

$$F(z) = \frac{z}{\sqrt{1-4z}} = z(1-4z)^{-1/2} = z\sum_{n\geq 0}\binom{-1/2}{n}(-4z)^n \ .$$

After the computations

$$F(z) = \sum_{n\geq 0}\binom{2n}{n}z^{n+1} = \sum_{n\geq 1}\binom{2n-2}{n-1}z^n \ ,$$

and

$$f_n = \binom{2n-2}{n-1} \quad \text{or} \quad f_{n+1} = \binom{2n}{n} = (n+1)b_n \ .$$

The number of binary trees with n vertices and k leaves

A bit harder problem: how many binary trees are there with n vertices and k leaves? Let us denote this number by $b_n^{(k)}$. It is easy to see that $b_n^{(k)} = 0$, if $k > \lfloor (n+1)/2 \rfloor$. By a simple reasoning the case $k = 1$ can be solved. The result is $b_n^{(1)} = 2^{n-1}$ for any natural number $n \geq 1$. Let $b_0^{(0)} = 1$, conventionally. We will see later that this is a good choice. Let us consider, as in the case of previous problems, the left and right subtrees. If the left subtree has i vertices and j leaves, then the right subtree has $n - i - 1$ vertices and $k - j$ leaves. The number of these trees is $b_i^{(j)} b_{n-i-1}^{(k-j)}$. Summing over k and j gives

$$b_n^{(k)} = 2b_{n-1}^{(k)} + \sum_{i=1}^{n-2}\sum_{j=1}^{k-1} b_i^{(j)} b_{n-i-1}^{(k-j)} \ . \tag{5.18}$$

For solving this recurrence equation the generating function

$$B^{(k)}(z) = \sum_{n\geq 0} b_n^{(k)} z^n, \qquad \text{where } k \geq 1$$

will be used. Multiplying both sides of equation (5.18) by z^n and summing over $n = 0, 1, 2, \ldots$, we get

$$\sum_{n\geq 1} b_n^{(k)} z^n = 2\sum_{n\geq 1} b_{n-1}^{(k)} z^n + \sum_{n\geq 1}\left(\sum_{i=1}^{n-2}\sum_{j=1}^{k-1} b_i^{(j)} b_{n-i-1}^{(k-j)}\right) z^n \ .$$

Changing the order of summation gives

$$\sum_{n\geq 1} b_n^{(k)} z^n = 2\sum_{n\geq 1} b_{n-1}^{(k)} z^n + \sum_{j=1}^{k-1}\sum_{n\geq 1}\left(\sum_{i=1}^{n-2} b_i^{(j)} b_{n-i-1}^{(k-j)}\right) z^n \ .$$

Thus

$$B^{(k)}(z) = 2zB^{(k)}(z) + z\left(\sum_{j=1}^{k-1} B^{(j)}(z)B^{(k-j)}(z)\right)$$

or

$$B^{(k)}(z) = \frac{z}{1-2z}\left(\sum_{j=1}^{k-1} B^{(j)}(z)B^{(k-j)}(z)\right) . \tag{5.19}$$

Step by step, we can write the following:

$$B^{(2)}(z) = \frac{z}{1-2z}\Big(B^{(1)}(z)\Big)^2 ,$$

$$B^{(3)}(z) = \frac{2z^2}{(1-2z)^2}\Big(B^{(1)}(z)\Big)^3 ,$$

$$B^{(4)}(z) = \frac{5z^3}{(1-2z)^3}\Big(B^{(1)}(z)\Big)^4 .$$

Let us try to find the solution in the form

$$B^{(k)}(z) = \frac{c_k z^{k-1}}{(1-2z)^{k-1}}\Big(B^{(1)}(z)\Big)^k ,$$

where $c_2 = 1$, $c_3 = 2$, $c_4 = 5$. Substituting in (5.19) gives a recursion for the numbers c_k

$$c_k = \sum_{i=1}^{k-1} c_i c_{k-i} .$$

We solve this equation using the generating function method. If $k = 2$, then $c_2 = c_1c_1$, and so $c_1 = 1$. Let $c_0 = 1$. If $C(z) = \sum_{n\geq 0} c_n z^n$ is the generating function of the numbers c_n, then, using the formula of multiplication of the generating functions we obtain

$$C(z) - 1 - z = (C(z) - 1)^2 \quad \text{or} \quad C^2(z) - 3C(z) + z + 2 = 0 ,$$

thus

$$C(z) = \frac{3 - \sqrt{1-4z}}{2}.$$

Since $C(0) = 1$, only the negative sign can be chosen. After expanding the generating function we get

$$\begin{aligned} C(z) &= \frac{3}{2} - \frac{1}{2}(1-4z)^{1/2} = \frac{3}{2} - \frac{1}{2}\sum_{n\geq 0}\frac{-1}{2n-1}\binom{2n}{n}z^n \\ &= \frac{3}{2} + \sum_{n\geq 0}\frac{1}{2(2n-1)}\binom{2n}{n}z^n = 1 + \sum_{n\geq 1}\frac{1}{2(2n-1)}\binom{2n}{n}z^n . \end{aligned}$$

From this

$$c_n = \frac{1}{2(2n-1)}\binom{2n}{n}, \quad n \geq 1 .$$

Since $b_n^{(1)} = 2^{n-1}$ for $n \geq 1$, it can be proved easily that $B^{(1)} = z/(1-2z)$. Thus

$$B^{(k)}(z) = \frac{1}{2(2k-1)}\binom{2k}{k}\frac{z^{2k-1}}{(1-2z)^{2k-1}} .$$

Using the formula

$$\frac{1}{(1-z)^m} = \sum_{n\geq 0}\binom{n+m-1}{n}z^n ,$$

therefore

$$\begin{aligned} B^{(k)}(z) &= \frac{1}{2(2k-1)}\binom{2k}{k}\sum_{n\geq 0}\binom{2k+n-2}{n}2^n z^{2k+n-1} \\ &= \frac{1}{2(2k-1)}\binom{2k}{k}\sum_{n\geq 2k-1}\binom{n-1}{n-2k+1}2^{n-2k+1}z^n . \end{aligned}$$

Thus

$$b_n^{(k)} = \frac{1}{2k-1}\binom{2k}{k}\binom{n-1}{2k-2}2^{n-2k}$$

or

$$b_n^{(k)} = \frac{1}{n}\binom{2k}{k}\binom{n}{2k-1}2^{n-2k} .$$

5.2.3. The Z-transform method

When solving linear nonhomogeneous equations using generating functions, the solution is usually done by the expansion of a rational fraction. The Z-transform method can help us in expanding such a function. Let $P(z)/Q(z)$ be a rational fraction, where the degree of $P(z)$ is less than the degree of $Q(z)$. If the roots of the denominator are known, the rational fraction can be expanded into partial fractions using the Undetermined Coefficient Method.

Let us first consider the case when the denominator has distinct roots $\alpha_1, \alpha_2, \ldots, \alpha_k$. Then

$$\frac{P(z)}{Q(z)} = \frac{A_1}{z-\alpha_1} + \cdots + \frac{A_i}{z-\alpha_i} + \cdots + \frac{A_k}{z-\alpha_k} .$$

It is easy to see that

$$A_i = \lim_{z\to\alpha_i}(z-\alpha_i)\frac{P(z)}{Q(z)}, \qquad i = 1, 2, \ldots, k .$$

But

$$\frac{A_i}{z-\alpha_i} = \frac{A_i}{-\alpha_i\left(1-\dfrac{1}{\alpha_i}z\right)} = \frac{-A_i\beta_i}{1-\beta_i z} ,$$

where $\beta_i = 1/\alpha_i$. Now, by expanding this partial fraction, we get

$$\frac{-A_i\beta_i}{1-\beta_i z} = -A_i\beta_i\left(1+\beta_i z+\cdots+\beta_i^n z^n+\cdots\right).$$

Denote the coefficient of z^n by $C_i(n)$, then $C_i(n) = -A_i\beta_i^{n+1}$, so

$$C_i(n) = -A_i\beta_i^{n+1} = -\beta_i^{n+1}\lim_{z\to\alpha_i}(z-\alpha_i)\frac{P(z)}{Q(z)} ,$$

or

$$C_i(n) = -\beta_i^{n+1} \lim_{z\to\alpha_i} \frac{(z-\alpha_i)P(z)}{Q(z)} .$$

After the transformation $z \to 1/z$ and using $\beta_i = 1/\alpha_i$ we obtain

$$C_i(n) = \lim_{z\to\beta_i} \left((z-\beta_i) z^{n-1} \frac{p(z)}{q(z)} \right) ,$$

where

$$\frac{p(z)}{q(z)} = \frac{P(1/z)}{Q(1/z)} .$$

Thus in the expansion of $X(z) = \dfrac{P(z)}{Q(z)}$ the coefficient of z^n is

$$C_1(n) + C_2(n) + \cdots + C_k(n) .$$

If α is a root of the polynomial $Q(z)$, then $\beta = 1/\alpha$ is a root of $q(z)$. E.g. if

$$\frac{P(z)}{Q(z)} = \frac{2z^z}{(1-z)(1-2z)} , \quad \text{then} \quad \frac{p(z)}{q(z)} = \frac{2}{(z-1)(z-2)} .$$

If the root is multiple, e.g. if β_i has multiplicity p, then its corresponding in the solution is

$$C_i(n) = \frac{1}{(p-1)!} \lim_{z\to\beta_i} \frac{d^{p-1}}{dz^{p-1}} \left((z-\beta_i)^p z^{n-1} \frac{p(z)}{q(z)} \right) .$$

Here $\dfrac{d^p}{dz^p} f(z)$ is the derivative of order p of the function $f(z)$.

All these can be summarised in the following algorithm. Let us consider that the coefficients of the equation are in array A, and the constants of the solution are in array C.

LINEAR-NONHOMOGENEOUS(A, k, f)

1 let $a_0 x_n + a_1 x_{n+1} + \cdots + a_k x_{n+k} = f(n)$ be the equation, where $f(n)$ is a rational fraction; multiply both sides by z^n, and sum over all n
2 transform the equation into the form $X(z) = P(z)/Q(z)$, where $X(z) = \sum_{n\geq 0} x_n z^n$, $P(z)$ and $Q(z)$ are polynomials
3 use the transformation $z \to 1/z$, and let the result be $p(z)/q(z)$, where $p(z)$ are $q(z)$ are polynomials
4 denote the roots of $q(z)$ by
  β_1, with multiplicity p_1, $p_1 \geq 1$,
  β_2, with multiplicity p_2, $p_2 \geq 1$,
  $\ldots$
  β_k, with multiplicity p_k, $p_k \geq 1$;
  then the general solution of the original equation is
  $x_n = C_1(n) + C_2(n) + \cdots + C_k(n)$, where
  $C_i(n) = 1/((p_i-1)!) \lim_{z\to\beta_i} \frac{d^{p_i-1}}{dz^{p_i-1}} \left((z-\beta_i)^{p_i} z^{n-1} (p(z)/q(z)) \right) , \quad i = 1, 2, \ldots, k.$
5 **return** C

If we substitute z by $1/z$ in the generating function, the result is the so-called Z-transform, for which similar operations can be defined as for the generating functions. The residue theorem for the Z-transform gives the same result. The name of the method is derived from this observation.

Example 5.12 Solve the recurrence equation

$$x_{n+1} - 2x_n = 2^{n+1} - 2, \quad \text{ha} \quad n \geq 0, \quad x_0 = 0.$$

Multiplying both sides by z^n and summing we obtain

$$\sum_{n\geq 0} x_{n+1}z^n - 2\sum_{n\geq 0} x_n z^n = \sum_{n\geq 0} 2^{n+1}z^n - \sum_{n\geq 0} 2z^n \ ,$$

or

$$\frac{1}{z}X(z) - 2X(z) = \frac{2}{1-2z} - \frac{2}{1-z}, \quad \text{where } X(z) = \sum_{n\geq 0} x_n z^n \ .$$

Thus

$$X(z) = \frac{2z^2}{(1-z)(1-2z)^2} \ .$$

After the transformation $z \to 1/z$ we get

$$\frac{p(z)}{q(z)} = \frac{2z}{(z-1)(z-2)^2} \ ,$$

where the roots of the denominator are 1 with multiplicity 1 and 2 with multiplicity 2. Thus

$$C_1 = \lim_{z\to 1} \frac{2z^n}{(z-2)^2} = 2 \qquad \text{and}$$

$$C_2 = \lim_{z\to 2} \frac{d}{dz}\left(\frac{2z^n}{z-1}\right) = 2\lim_{z\to 2} \frac{nz^{n-1}(z-1) - z^n}{(z-1)^2} = 2^n(n-2) \ .$$

Therefore the general solution is

$$x_n = 2^n(n-2) + 2, \quad n \geq 0 \ .$$

Example 5.13 Solve the recurrence equation

$$x_{n+2} = 2x_{n+1} - 2x_n, \quad \text{if} \quad n \geq 0, \quad x_0 = 0, \quad x_1 = 1.$$

Multiplying by z^n and summing gives

$$\frac{1}{z^2}\sum_{n\geq 0} x_{n+2}z^{n+2} = \frac{2}{z}\sum_{n\geq 0} x_{n+1}z^{n+1} - 2\sum_{n\geq 0} x_n z^n \ ,$$

so

$$\frac{1}{z^2}\Big(F(z) - z\Big) = \frac{2}{z}F(z) - 2F(z) \ ,$$

that is

$$F(z)\left(\frac{1}{z^2} - \frac{2}{z} + 2\right) = -\frac{1}{z} \ .$$

Then

$$F(1/z) = \frac{-z}{z^2 - 2z + 2} \ .$$

The roots of the denominator are $1+i$ and $1-i$. Let us compute $C_1(n)$ and $C_2(n)$:

$$C_1(n) = \lim_{z\to 1+i} \frac{-z^{n+1}}{z-(1-i)} = \frac{i(1+i)^n}{2} \quad \text{and}$$

$$C_2(n) = \lim_{z\to 1-i} \frac{-z^{n+1}}{z-(1+i)} = \frac{-i(1-i)^n}{2} .$$

Since

$$1+i = \sqrt{2}\left(\cos\frac{\pi}{4} + i\sin\frac{\pi}{4}\right), \quad 1-i = \sqrt{2}\left(\cos\frac{\pi}{4} - i\sin\frac{\pi}{4}\right) ,$$

raising to the nth power gives

$$(1+i)^n = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} + i\sin\frac{n\pi}{4}\right), \quad (1-i)^n = (\sqrt{2})^n\left(\cos\frac{n\pi}{4} - i\sin\frac{n\pi}{4}\right) ,$$

$$x_n = C_1(n) + C_2(n) = \left(\sqrt{2}\right)^n \sin\frac{n\pi}{4} .$$

Exercises

5.2-1 How many binary trees are there with n vertices and no empty left and right subtrees?

5.2-2 How many binary trees are there with n vertices, in which each vertex which is not a leaf, has exactly two descendants?

5.2-3 Solve the following recurrent equation using generating functions.

$$H_n = 2H_{n-1} + 1, \quad H_0 = 0 .$$

(H_n is the number of moves in the problem of the Towers of Hanoi.)

5.2-4 Solve the following recurrent equation using the Z-transform method.

$$F_{n+2} = F_{n+1} + F_n + 1, \text{ ha } n \geq 0, \text{ és } \quad F_0 = 0, F_1 = 1 .$$

5.2-5 Solve the following system of recurrence equations:

$$\begin{aligned} u_n &= v_{n-1} + u_{n-2} , \\ v_n &= u_n + u_{n-1} , \end{aligned}$$

where $u_0 = 1, u_1 = 2, v_0 = 1$.

5.3. Numerical solution

Using the following function we can solve the linear recurrent equations numerically. The equation is given in the form

$$a_0 x_n + a_1 x_{n+1} + \cdots + a_k x_{n+k} = f(n) ,$$

where $a_0, a_k \neq 0, k \geq 1$. The coefficients $a_0, a_1, \ldots, a_k$ are kept in array A, the initial values $x_0, x_1, \ldots, x_{k-1}$ in array X. To find x_n we will compute step by step the values $x_k, x_{k+1}, \ldots, x_n$, keeping in the previous k values of the sequence in the first k positions of X (i.e. in the positions with indices $0, 1, \ldots, k-1$).

RECURRENCE(A, X, k, n, f)

```
 1 for j ← k to n
 2     do v ← A[0] · X[0]
 3        for i ← 1 to k − 1
 4            do v ← v + A[i] · X[i]
 5        v ← (f(j − k) − v)/A[k]
 6        if j ≠ n
 7          then for i ← 0 to k − 2
 8                    do X[i] ← X[i + 1]
 9               X[k − 1] ← v
10 return v
```

Lines 2–5 compute the values x_j $(j = k, k+1, \ldots, n)$ (using the previous k values), denoted by v in the algorithm. In lines 7–9, if n is not yet reached, we copy the last k values in the first k positions of X. In line 10 x_n is obtained. It is easy to see that the computation time is $\Theta(kn)$, if we do not count the time to compute the value of the function.

Exercises

5.3-1 How many additions, subtractions, multiplications and divisions are required using the algorithm RECURRENCE, while it computes x_{1000} using the data given in Example 5.4?

Problems

5-1. Existence of a solution of homogeneous equation using generating function

Prove that a linear homogeneous equation cannot be solved using generating functions (because $X(z) = 0$ is obtained) if and only if $x_n = 0$ for all n.

5-2. Complex roots in the case of Z-transform

What happens if the roots of the denominator are complex when applying the Z-transform method? The solution of the recurrence equation must be real. Does the method ensure this?

Chapter notes

The recurrence equations are discussed in detail by Elaydi [13], Flajolet and Sedgewick [55], Greene and Knuth [23], Mickens [43].

Knuth [30] and Graham, Knuth and Patashnik [22] deal with generating functions. In the book of Vilenkin [68] there are a lot of simple and interesting problems about recurrences and generating functions.

In [39] Lovász also presents problems on generating function.

Counting the binary trees is from Knuth [30], counting the leaves in the set of all binary trees and counting the binary trees with n vertices and k leaves are from Z. Kása [32].

6. Reliable computation

Any planned computation will be subject to different kinds of unpredictable influences during execution. Here are some examples:

1. Loss or change of stored data during execution.
2. Random, physical errors in the computer.
3. Unexpected interactions between different parts of the system working simultaneously, or loss of connections in a network.
4. Bugs in the program.
5. Malicious attacks.

Up to now, it does not seem that the problem of bugs can be solved just with the help of appropriate algorithms. The discipline of software engineering addresses this problem by studying and improving the structure of programs and the process of their creation.

Malicious attacks are addressed by the discipline of computer security. A large part of the recommended solutions involves cryptography.

Problems of kind (3) are very important and a whole discipline, distributed computing has been created to deal with them.

The problem of storage errors is similar to the problems of reliable communication, studied in information theory: it can be viewed as communication from the present to the future. In both cases, we can protect against noise with the help of ***error-correcting codes*** (you will see some examples below).

In this chapter, we will discuss some sample problems, mainly from category (2). In this category, distinction should also be made between permanent and transient errors. An error is ***permanent*** when a part of the computing device is damaged physically and remains faulty for a long time, until some outside intervention by repairmen to it. It is ***transient*** if it happens only in a single step: the part of the device in which it happened is not damaged, in the next step it operates correctly again. For example, if a position in memory turns from 0 to 1 by accident, but a subsequent write operation can write a 0 again then a transient error happened. If the bit turned to 1 and the computer cannot change it to 0 again, this is a permanent error.

Some of these problems, especially the ones for transient errors, are as old as computing. The details of any physical errors depend on the kind of computer it is implemented on (and, of course, on the kind of computation we want to carry out). But after abstracting

away from a lot of distracting details, we are left with some clean but challenging theoretical formulations, and some rather pleasing solutions. There are also interesting connections to other disciplines, like statistical physics and biology.

The computer industry has been amazingly successful over the last five decades in making the computer components smaller, faster, and at the same time more reliable. Among the daily computer horror stories seen in the press, the one conspicuously missing is where the processor wrote a 1 in place of a 0, just out of caprice. (It indisputably happens, but too rarely to become the identifiable source of some visible malfunction.) On the other hand, the generality of some of the results on the correction of transient errors makes them applicable in several settings. Though individual physical processors are very reliable (error rate is maybe once in every 10^{20} executions), when considering a whole network as performing a computation, the problems caused by unreliable network connections or possibly malicious network participants is not unlike the problems caused by unreliable processors.

The key idea for making a computation reliable is *redundancy*, which might be formulated as the following two procedures:

1. Store information in such a form that losing any small part of it is not fatal: it can be restored using the rest of the data. For example, store it in multiple copies.
2. Perform the needed computations repeatedly, to make sure that the faulty results can be outvoted.

Our chapter will only use these methods, but there are other remarkable ideas which we cannot follow up here. For example, method (2) seems especially costly; it is desirable to avoid a lot of repeated computation. The following ideas target this dilemma.

1. Perform the computation directly on the information in its redundant form: then maybe recomputations can be avoided.
2. Arrange the computation into "segments" such a way that those partial results that are to be used later, can be cheaply checked at each "milestone" between segments. If the checking finds error, repeat the last segment.

6.1. Probability theory

The present chapter does not require great sophistication in probability theory but there are some facts coming up repeatedly which I will review here. If you need additional information, you will find it in any graduate probability theory text.

6.1.1. Terminology

A ***probability space*** is a triple $(\Omega, \mathcal{A}, \mathbf{P})$ where Ω is the set of ***elementary events***, $\mathcal{A}$ is a set of subsets of Ω called the set of ***events*** and $\mathbf{P} : \mathcal{A} \to [0,1]$ is a function. For $E \in \mathcal{A}$, the value $\mathbf{P}(E)$ is called the ***probability*** of event E. It is required that $\Omega \in \mathcal{A}$ and that $E \in \mathcal{A}$ implies $\Omega \setminus E \in \mathcal{A}$. Further, if a (possibly infinite) sequence of sets is in $\mathcal{A}$ then so is their union. Also, it is assumed that $\mathbf{P}(\Omega) = 1$ and that if $E_1, E_2, \ldots \in \mathcal{A}$ are disjoint then

$$\mathbf{P}\Big(\bigcup_i E_i\Big) = \sum_i \mathbf{P}(E_i).$$

For $\mathbf{P}(F) > 0$, the ***conditional probability*** of E given F is defined as

$$\mathbf{P}(E \mid F) = \mathbf{P}(E \cap F)/\mathbf{P}(F).$$

Events $E_1, \ldots, E_n$ are ***independent*** if for any sequence $1 \le i_1 < \cdots < i_k \le n$ we have

$$\mathbf{P}(E_{i_1} \cap \cdots \cap E_{i_k}) = \mathbf{P}(E_{i_1}) \cdots \mathbf{P}(E_{i_k}).$$

Example 6.1 Let $\Omega = \{1, \ldots, n\}$ where $\mathcal{A}$ is the set of all subsets of Ω and $\mathbf{P}(E) = |E|/n$. This is an example of a ***discrete*** probability space: one that has a countable number of elements.

More generally, a discrete probability space is given by a countable set $\Omega = \{\omega_1, \omega_2, \ldots\}$, and a sequence $p_1, p_2, \ldots$ with $p_i \ge 0$, $\sum_i p_i = 1$. The set $\mathcal{A}$ of events is the set of all subsets of Ω, and for an event $E \subset \Omega$ we define $\mathbf{P}(E) = \sum_{\omega_i \in E} p_i$.

A ***random variable*** over a probability space Ω is a function f from Ω to the real numbers, with the property that every set of the form $\{\, \omega : f(\omega) < c \,\}$ is an event: it is in $\mathcal{A}$. Frequently, random variables are denoted by capital letters X, Y, Z, possibly with indices, and the argument ω is omitted from $X(\omega)$. The event $\{\, \omega : X(\omega) < c \,\}$ is then also written as $[\, X < c \,]$. This notation is freely and informally extended to more complicated events. The ***distribution*** of a random variable X is the function $F(c) = \mathbf{P}[\, X < c \,]$. We will frequently only specify the distribution of our variables, and not mention the underlying probability space, when it is clear from the context that it can be specified in one way or another. We can speak about the ***joint distribution*** of two or more random variables, but only if it is assumed that they can be defined as functions on a common probability space. Random variables $X_1, \ldots, X_n$ with a joint distribution are ***independent*** if every n-tuple of events of the form $[\, X_1 < c_1 \,], \ldots, [\, X_n < c_n \,]$ is independent.

The ***expected value*** of a random variable X taking values $x_1, x_2, \ldots$ with probabilities $p_1, p_2, \ldots$ is defined as

$$\mathbf{E}X = p_1 x_1 + p_2 x_2 + \cdots .$$

It is easy to see that the expected value is a linear function of the random variable:

$$\mathbf{E}(\alpha X + \beta Y) = \alpha \mathbf{E}X + \beta \mathbf{E}Y,$$

even if X, Y are not independent. On the other hand, if variables X, Y are independent then the expected values can also be multiplied:

$$\mathbf{E}XY = \mathbf{E}X \cdot \mathbf{E}Y. \tag{6.1}$$

There is an important simple inequality called the ***Markov inequality***, which says that for an arbitrary nonnegative random variable X and any value $\lambda > 0$ we have

$$\mathbf{P}[\, X \ge \lambda \,] \le \mathbf{E}X/\lambda. \tag{6.2}$$

6.1.2. Combinatorial estimates

Stirling's formula gives an asymptotic expression of $n!$. It is frequently used in probability theory since combinatorial formulas often contain $n!$. Rather than using Stirling's formula

directly, let us develop two simple inequalities which should always suffice for us.

Unless stated otherwise, $\log x$ will denote logarithm base 2 of x, and $\ln x$ will denote the natural logarithm. Notice that the function $\ln x$ is an increasing function. Therefore on the interval $[n, n+1]$, the constant $\ln n$ is a lower bound to it, and $\ln(n+1)$ is an upper bound. It follows that

$$\ln(n!) = \sum_{i=1}^{n} \ln i < \int_1^{n+1} \ln x \, dx < \sum_{i=1}^{n} \ln(i+1) = \ln((n+1)!).$$

Hence,

$$\int_1^n \ln x \, dx < \ln(n!) = \ln n + \ln(n-1)! < \ln n + \int_1^n \ln x \, dx.$$

Now, $\int \ln x \, dx = x(\ln x - 1)$. Hence,

$$n(\ln n - 1) + 1 < \ln(n!) < n(\ln n - 1) + 1 + \ln n. \tag{6.3}$$

In exponential form:

$$e\left(\frac{n}{e}\right)^n < n! < en\left(\frac{n}{e}\right)^n.$$

6.1.3. The law of large numbers (with "large deviations")

In what follows the bounds

$$\frac{x}{1+x} \le \ln(1+x) \le x \qquad \text{for } x > -1 \tag{6.4}$$

will be useful. Of these, the well-known upper bound $\ln(1+x) \le x$ holds since the graph of the function $\ln(1+x)$ is below its tangent line drawn at the point $x = 0$. The lower bound is obtained from the identity $\frac{1}{1+x} = 1 - \frac{x}{1+x}$ and

$$-\ln(1+x) = \ln \frac{1}{1+x} = \ln\left(1 - \frac{x}{1+x}\right) \le -\frac{x}{1+x}.$$

Consider n independent random variables $X_1, \ldots, X_n$ that are identically distributed, with

$$\mathbf{P}[\, X_i = 1 \,] = p, \quad \mathbf{P}[\, X_i = 0 \,] = 1 - p.$$

Let

$$S_n = X_1 + \cdots + X_n.$$

We want to estimate the probability $\mathbf{P}[\, S_n \ge fn \,]$ for any constant $0 < f < 1$. The "law of large numbers" says that if $f > p$ then this probability converges fast to 0 as $n \to \infty$ while if $f < p$ then it converges fast to 1. Let

$$D(f,p) = f \ln \frac{f}{p} + (1-f) \ln \frac{1-f}{1-p} \tag{6.5}$$

$$> f \ln \frac{f}{p} - f = f \ln \frac{f}{ep}, \tag{6.6}$$

where the inequality (useful for small f and $ep < f$) comes via $1 > 1 - p > 1 - f$ and $\ln(1-f) \ge -\frac{f}{1-f}$ from (6.4). Using the concavity of logarithm, it can be shown that $D(f,p)$ is always nonnegative, and is 0 only if $f = p$ (see Exercise 6.1-1.).

Theorem 6.1 (Large deviations for coin-toss). *If $f > p$ then*

$$\mathbf{P}[\, S_n \geq fn \,] \leq e^{-nD(f,p)}.$$

This theorem shows that if $f > p$ then $\mathbf{P}[\, S_n > fn \,]$ converges to 0 exponentially fast. Inequality (6.6) will allow the following simplification:

$$\mathbf{P}[\, S_n \geq fn \,] \leq e^{-nf \ln \frac{f}{ep}} = \left(\frac{ep}{f}\right)^{nf}, \tag{6.7}$$

useful for small f and $ep < f$.

Proof. For a certain real number $\alpha > 1$ (to be chosen later), let Y_n be the random variable that is α if $X_n = 1$ and 1 if $X_n = 0$, and let $P_n = Y_1 \cdots Y_n = \alpha^{S_n}$: then

$$\mathbf{P}[\, S_n \geq fn \,] = \mathbf{P}[\, P_n \geq \alpha^{fn} \,].$$

Applying the Markov inequality (6.2) and (6.1), we get

$$\mathbf{P}[\, P_n \geq \alpha^{fn} \,] \leq \mathbf{E} P_n / \alpha^{fn} = (\mathbf{E} Y_1 / \alpha^f)^n,$$

where $\mathbf{E} Y_1 = p\alpha + (1-p)$. Let us choose $\alpha = \frac{f(1-p)}{p(1-f)}$, this is > 1 if $p < f$. Then we get $\mathbf{E} Y_1 = \frac{1-p}{1-f}$, and hence

$$\mathbf{E} Y_1 / \alpha^f = \frac{p^f (1-p)^{1-f}}{f^f (1-f)^{1-f}} = e^{-D(f,p)}.$$

■

This theorem also yields some convenient estimates for binomial coefficients. Let

$$h(f) = -f \ln f - (1-f) \ln(1-f).$$

This is sometimes called the ***entropy*** of the probability distribution $(f, 1-f)$ (measured in logarithms over base e instead of base 2). From inequality (6.4) we obtain the estimate

$$-f \ln f \leq h(f) \leq f \ln \frac{e}{f} \tag{6.8}$$

which is useful for small f.

Corollary 6.2 *We have, for $f \leq 1/2$:*

$$\sum_{i \leq fn}^{n} \binom{n}{i} \leq e^{nh(f)} \leq \left(\frac{e}{f}\right)^{fn}. \tag{6.9}$$

In particular, taking $f = k/n$ with $k \leq n/2$ gives

$$\binom{n}{k} = \binom{n}{fn} \leq \left(\frac{e}{f}\right)^{fn} = \left(\frac{ne}{k}\right)^{k}. \tag{6.10}$$

Proof. Theorem 6.1 says for the case $f > p = 1/2$:

$$2^{-n} \sum_{i \geq fn}^{n} \binom{n}{i} = \mathbf{P}[\, S_n \geq fn \,] \leq e^{-nD(f,p)} = 2^{-n} e^{nh(f)},$$

$$\sum_{i \geq fn}^{n} \binom{n}{i} \leq e^{nh(f)}.$$

Substituting $g = 1 - f$, and noting the symmetries $\binom{n}{f} = \binom{n}{g}$, $h(f) = h(g)$ and (6.8) gives formula (6.9). ■

Remark 6.3 *Inequality* (6.7) *also follows from the trivial estimate* $\mathbf{P}[\, S_n \geq fn \,] \leq \binom{n}{fn} p^{fn}$ *combined with* (6.10).

Exercises

6.1-1 Prove that the statement made in the main text that $D(f, p)$ is always nonnegative, and is 0 only if $f = p$.

6.1-2 For $f = p + \delta$, derive from Theorem 6.1 the useful bound

$$\mathbf{P}[\, S_n \geq fn \,] \leq e^{-2\delta^2 n}.$$

Hint: Let $F(x) = D(x, p)$, and use the Taylor formula: $F(p + \delta) = F(p) + F'(p)\delta + F''(p + \delta')\delta^2/2$, where $0 \leq \delta' \leq \delta$.]

6.1-3 Prove that in Theorem 6.1, the assumption that X_i are independent and identically distributed can be weakened: replaced by the single inequality

$$\mathbf{P}[\, X_i = 1 \mid X_1, \ldots, X_{i-1} \,] \leq p.$$

6.2. Logic circuits

In a model of computation taking errors into account, the natural assumption is that errors occur *everywhere*. The most familiar kind of computer, which is separated into a single processor and memory, seems extremely vulnerable under such conditions: while the processor is not "looking", noise may cause irreparable damage in the memory. Let us therefore rather consider computation models that are *parallel*: information is being processed everywhere in the system, not only in some distinguished places. Then error correction can be built into the work of every part of the system. We will concentrate on the best known parallel computation model: Boolean circuits.

6.2.1. Boolean functions and expressions

Let us look inside a computer, (actually inside an integrated circuit, with a microscope). Discouraged by a lot of physical detail irrelevant to abstract notions of computation, we will decide to look at the blueprints of the circuit designer, at the stage when it shows the

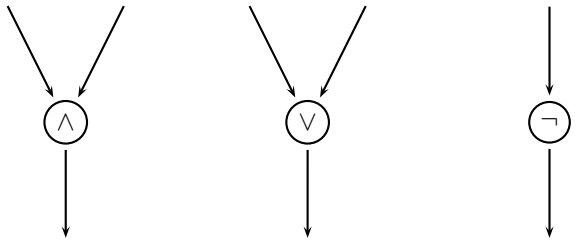


Figure 6.1. AND, OR and NOT gate.

smallest elements of the circuit still according to their computational functions. We will see a network of lines that can be in two states (of electric potential), "high" or "low", or in other words "true" or "false", or, as we will write, 1 or 0. The points connected by these lines are the familiar ***logic components***: at the lowest level of computation, a typical computer processes ***bits***. Integers, floating-point numbers, characters are all represented as strings of bits, and the usual arithmetical operations can be composed of bit operations.

Definition 6.4 *A **Boolean vector function** is a mapping $f : \{0,1\}^n \to \{0,1\}^m$. Most of the time, we will take $m = 1$ and speak of a **Boolean function**.*

The variables in $f(x_1, \ldots, x_n)$ are sometimes called ***Boolean variables***, ***Boolean variables*** or ***bits***.

Example 6.2 Given an undirected graph G with N nodes, suppose we want to study the question whether it has a Hamiltonian cycle (a sequence $(u_1, \ldots, u_n)$ listing all vertices of G such that (u_i, u_{i+1}) is an edge for each $i < n$ and also (u_n, u_1) is an edge). This question is described by a Boolean function f as follows. The graph can be described with $\binom{N}{2}$ Boolean variables x_{ij} $(1 \le i < j \le N)$: x_{ij} is 1 if and only if there is an edge between nodes i and j. We define $f(x_{12}, x_{13}, \ldots, x_{N-1,N}) = 1$ if there is a Hamiltonian cycle in G and 0 otherwise.

Example 6.3 [Boolean vector function] Let $n = m = 2k$, let the input be two integers u, v, written as k-bit strings: $x = (u_1, \ldots, u_k, v_1, \ldots, v_k)$. The output of the function is their product $y = u \cdot v$ (written in binary): if $u = 5 = (101)_2$, $v = 6 = (110)_2$ then $y = u \cdot v = 30 = (11110)_2$.

There are only four one-variable Boolean functions: the identically 0, identically 1, the identity and the ***negation***: $x \to \neg x = 1 - x$. We mention only the following two-variable Boolean functions: the operation of ***conjunction*** (logical AND):

$$x \wedge y = \begin{cases} 1 & \text{if } x = y = 1 \ , \\ 0 & \text{otherwise} \ , \end{cases}$$

this is the same as multiplication. The operation of ***disjunction***, or logical OR:

$$x \vee y = \begin{cases} 0 & \text{if } x = y = 0 \ , \\ 1 & \text{otherwise} \ . \end{cases}$$

It is easy to see that $x \vee y = \neg(\neg x \wedge \neg y)$: in other words, disjunction $x \vee y$ can be expressed using the functions $\neg, \wedge$ and the operation of ***composition***. The following two-argument Boolean functions are also frequently used:

$$x \to y = \neg x \vee y \qquad \text{(implication)},$$
$$x \leftrightarrow y = (x \to y) \wedge (y \to x) \qquad \text{(equivalence)},$$
$$x \oplus y = x + y \bmod 2 = \neg(x \leftrightarrow y) \qquad \text{(binary addition)}.$$

A finite number of Boolean functions is sufficient to express all others: thus, arbitrarily complex Boolean functions can be "computed" by "elementary" operations. In some sense, this is what happens inside computers.

Definition 6.1 *A set of Boolean functions is a* ***complete basis*** *if every other Boolean function can be obtained by repeated composition from its elements.*

Claim 6.5 *The set* $\{\wedge, \vee, \neg\}$ *forms a complete basis; in other words, every Boolean function can be represented by a Boolean expression using only these connectives.*

The proof can be found in all elementary introductions to propositional logic. Note that since $\vee$ can be expressed using $\{\wedge, \neg\}$, this latter set is also a complete basis (and so is $\{\vee, \neg\}$).

From now on, under a ***Boolean expression*** (formula), we mean an expression built up from elements of some given complete basis. If we do not mention the basis then the complete basis $\{\wedge, \neg\}$ will be meant.

In general, one and the same Boolean function can be expressed in many ways as a Boolean expression. Given such an expression, it is easy to compute the value of the function. However, most Boolean functions can still be expressed only by very large Boolean expression (see Exercise 6.2-4.).

6.2.2. Circuits

A Boolean expression is sometimes large since when writing it, there is no possibility for *reusing partial results*. (For example, in the expression

$$((x \vee y \vee z) \wedge u) \vee (\neg(x \vee y \vee z) \wedge v),$$

the part $x \vee y \vee z$ occurs twice.) This deficiency is corrected by the following more general formalism.

A Boolean circuit is essentially an acyclic directed graph, each of whose nodes computes a Boolean function (from some complete basis) of the bits coming into it on its input edges, and sends out the result on its output edges (see Figure 6.2). Let us give a formal definition.

Definition 6.6 *Let* Q *be a complete basis of Boolean functions. For an integer* N *let* $V = \{1, \dots, N\}$ *be a set of* ***nodes****. A* ***Boolean circuit*** *over* Q *is given by the following tuple:*

$$\mathcal{N} = (V, \{k_v : v \in V\}, \{\arg_j(v) : v \in V;\ j = 1, \dots, k_v\}, \{b_v : v \in V\}). \tag{6.11}$$

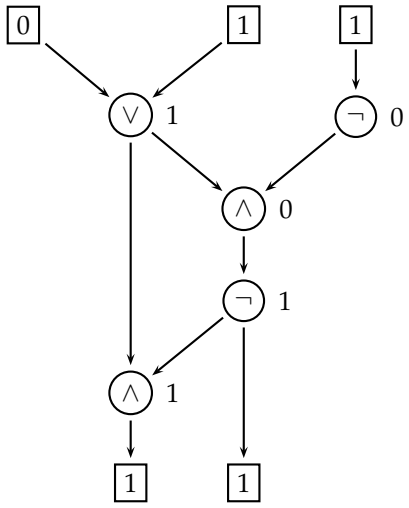


Figure 6.2. The assignment (values on nodes, configuration) gets propagated through all the gates. This is the "computation".

For every node v there is a natural number k_v showing its number of ***inputs****. The* ***sources****, nodes v with $k_v = 0$, are called* ***input nodes:*** *we will denote them, in increasing order, as*

$$\mathrm{inp}_i \quad (i = 1, \ldots, n).$$

To each non-input node v a Boolean function

$$b_v(y_1, \ldots, y_{k_v})$$

from the complete basis Q is assigned: it is called the ***gate*** *of node v. It has as many arguments as the number of entering edges. The* ***sinks*** *of the graph, nodes without outgoing edges, will be called* ***output nodes:*** *they can be denoted by*

$$\mathrm{out}_i \quad (i = 1, \ldots, m).$$

(Our Boolean circuits will mostly have just a single output node.) To every non-input node v and every $j = 1, \ldots, k_v$ belongs a node $\arg_j(v) \in V$ (the node sending the value of input variable y_j of the gate of v). The circuit defines a graph $G = (V, E)$ whose set of edges is

$$E = \{\, (\arg_j(v), v) : v \in V,\ j = 1, \ldots, k_v \,\}.$$

We require $\arg_j(v) < v$ for each j, v (we identified the with the natural numbers $1, \ldots, N$): this implies that the graph G is acyclic. The ***size*** *$|\mathcal{N}|$ of the circuit $\mathcal{N}$ is the number of nodes. The* ***depth*** *of a node v is the maximal length of directed paths leading from an input node to v. The* ***depth*** *of a circuit is the maximum depth of its output nodes.*

Definition 6.7 *An* ***input assignment****, or* ***input configuration*** *to our circuit $\mathcal{N}$ is a vector*

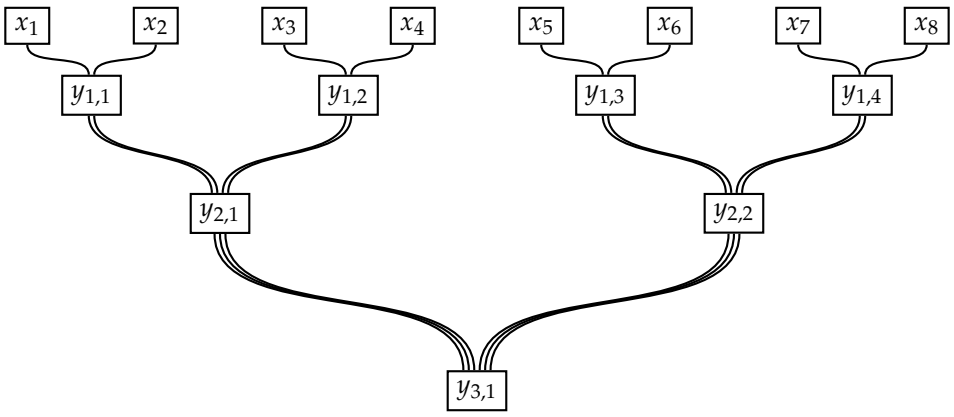


Figure 6.3. Naive parallel addition.

$\boldsymbol{x} = (x_1, \ldots, x_n)$ *with* $x_i \in \{0, 1\}$ *giving value* x_i *to node* inp_i*:*

$$\mathrm{val}_{\boldsymbol{x}}(v) = y_v(\boldsymbol{x}) = x_i$$

for $v = \mathrm{inp}_i$, $i = 1, \ldots, n$. *The function* $y_v(\boldsymbol{x})$ *can be extended to a unique* ***configuration*** $v \mapsto y_v(\boldsymbol{x})$ *on all other nodes of the circuit as follows. If gate* b_v *has* k *arguments then*

$$y_v = b_v(y_{\mathrm{arg}_1(v)}, \ldots, y_{\mathrm{arg}_k(v)}). \tag{6.12}$$

For example, if $b_v(x, y) = x \wedge y$, *and* $u_j = \mathrm{arg}_j(v)$ $(j = 1, 2)$ *are the input nodes to* v *then* $y_v = y_{u_1} \wedge y_{u_2}$. *The process of extending the configuration by the above equation is also called the* ***computation*** *of the circuit. The vector of the values* $y_{\mathrm{out}_i}(\boldsymbol{x})$ *for* $i = 1, \ldots, m$ *is the* ***result*** *of the computation. We say that the Boolean circuit* ***computes*** *the vector function*

$$\boldsymbol{x} \mapsto (y_{\mathrm{out}_1}(\boldsymbol{x}), \ldots, y_{\mathrm{out}_m}(\boldsymbol{x})).$$

The assignment procedure can be performed in ***stages****: in stage* t, *all nodes of depth* t *receive their values.*

We assign values to the edges as well: the value assigned to an edge is the one assigned to its start node.

6.2.3. Fast addition by a Boolean circuit

The depth of a Boolean circuit can be viewed as the shortest time it takes to compute the output vector from the input vector by this circuit. An example application of Boolean circuits, let us develop a circuit that computes the sum of its input bits very fast. We will need this result later in the present chapter for error-correcting purposes.

Definition 6.8 *We will say that a Boolean circuit computes a* ***near-majority*** *if it outputs a bit* y *with the following property: if* $3/4$ *of all input bits is equal to* b *then* $y = b$.

The depth of our circuit is clearly $\Omega(\lg n)$, since the output must have a path to the majority of inputs. In order to compute the majority, we will also solve the task of summing the input bits.

Theorem 6.9

1. *Over the complete basis consisting of the set of all 3-argument Boolean functions, for each n there is a Boolean circuit of input size n and depth $\leq 3\log(n+1)$ whose output vector represents the sum of the input bits as a binary number.*
2. *Over this same complete basis, for each n there is a Boolean circuit of input size n and depth $\leq 2\log(n+1)$ computing a near-majority.*

Proof. First we prove (1). For simplicity, assume $n = 2^k - 1$: if n is not of this form, we may add some fake inputs. The naive approach would be proceed according to Figure 6.3: to first compute $y_{1,1} = x_1 + x_2$, $y_{1,2} = x_3 + x_4$, ..., $y_{1,2^{k-1}} = x_{2^k-1} + x_{2^k}$. Then, to compute $y_{2,1} = y_{1,1} + y_{1,2}$, $y_{2,2} = y_{1,3} + y_{1,4}$, and so on. Then $y_{k,1} = x_1 + \cdots + x_{2^k}$ will indeed be computed in k stages.

It is somewhat troublesome that $y_{i,j}$ here is a number, not a bit, and therefore must be represented by a *bit vector*, that is by group of nodes in the circuit, not just by a single node. However, the general addition operation

$$y_{i+1,j} = y_{i,2j-1} + y_{i,2j},$$

when performed in the naive way, will typically take more than a constant number of steps: the numbers $y_{i,j}$ have length up to $i+1$ and therefore the addition may add i to the depth, bringing the total depth to $1 + 2 + \cdots + k = \Omega(k^2)$.

The following observation helps to decrease the depth. Let a, b, c be three numbers in binary notation: for example, $a = \sum_{i=0}^{k} a_i 2^i$. There are simple parallel formulas to represent the sum of these three numbers as the sum of two others, that is to compute $a+b+c = d+e$ where d, e are numbers also in binary notation:

$$\begin{aligned} d_i &= a_i + b_i + c_i \bmod 2, \\ e_{i+1} &= \lfloor (a_i + b_i + c_i)/2 \rfloor. \end{aligned} \tag{6.13}$$

Since both formulas are computed by a single 3-argument gate, 3 numbers can be reduced to 2 (while preserving the sum) in a single parallel computation step. Two such steps reduce 4 numbers to 2. In $2(k-1)$ steps therefore they reduce a sum of 2^k terms to a sum of 2 numbers of length $\leq k$. Adding these two numbers in the regular way increases the depth by k: we found that 2^k bits can be be added in $3k-2$ steps.

To prove (2), construct the circuit as in the proof of (1), but without the last addition: the output is two k-bit numbers whose sum we are interested in. The highest-order nonzero bit of these numbers is at some position $< k$. If the sum is more than 2^{k-1} then one these numbers has a nonzero bit at position $(k-1)$ or $(k-2)$. We can determine this in two applications of 3-input gates. ■

Exercises

6.2-1 Show that $\{1, \oplus, \wedge\}$ is a complete basis.

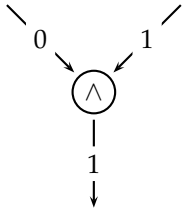


Figure 6.4. Failure at a gate.

6.2-2 Show that the function x NOR $y = \neg(x \vee y)$ forms a complete basis by itself.
6.2-3 Let us fix the complete basis $\{\wedge, \neg\}$. Prove Claim 6.5 (or look up its proof in a textbook). Use it to give an upper bound for an arbitrary Boolean function f of n variables, on:

1. the smallest size of a Boolean expression for f;
2. the smallest size of a Boolean circuit for f;
3. the smallest depth of a Boolean circuit for f;

6.2-4 Show that for every n there is a Boolean function f of n variables such that every Boolean circuit in the complete basis $\{\wedge, \neg\}$ computing f contains $\Omega(2^n/n)$ nodes. *Hint*: for a constant $c > 0$, upperbound the number of Boolean circuits with at most $c2^n/n$ nodes and compare it with the number of Boolean functions over n variables.
6.2-5 Consider a circuit $\mathcal{M}_3^r$ with 3^r inputs, whose single output bit is computed from the inputs by r levels of 3-input majority gates. Show that there is an input vector $\boldsymbol{x}$ which is 1 in only $n^{1/\lg 3}$ positions but with which $\mathcal{M}_3^r$ outputs 1. Thus a small minority of the inputs, when cleverly arranged, can command the result of this circuit.

6.3. Expensive fault-tolerance in Boolean circuits

Let $\mathcal{N}$ be a Boolean circuit as given in Definition 6.6. When noise is allowed then the values

$$y_v = \mathrm{val}_{\boldsymbol{x}}(v)$$

will not be determined by the formula (6.12) anymore. Instead, they will be random variables Y_v. The random assignment $(Y_v : v \in V)$ will be called a ***random configuration***.

Definition 6.10 *At vertex v, let*

$$Z_v = b_v(Y_{\arg_1(v)}, \ldots, Y_{\arg_k(v)}) \oplus Y_v. \tag{6.14}$$

*In other words, $Z_v = 1$ if gate Y_v is not equal to the value computed by the noise-free gate b_v from its inputs $Y_{\arg_j(v)}$. (See Figure 6.4.) The set of vertices where Z_v is non-zero is the set of **faults**.*

Let us call the difference $\mathrm{val}_{\boldsymbol{x}}(v) \oplus Y_v$ *the **deviation** at node v.*

Let us impose conditions on the kind of noise that will be allowed. Each fault should occur only with probability at most ϵ, two specific faults should only occur with probability at most ϵ^2, and so on.

Definition 6.11 *For an $\epsilon > 0$, let us say that the random configuration $(Y_v : v \in V)$ is **ϵ-admissible** if*

1. $Y_{\mathrm{inp}(i)} = x_i$ *for* $i = 1, \ldots, n$.
2. *For every set C of non-input nodes, we have*

$$\mathbf{P}[\, Z_v = 1 \text{ for all } v \in C \,] \leq \epsilon^{|C|}. \tag{6.15}$$

In other words, in an ϵ-admissible random configuration, the probability of having faults at k different specific gates is at most ϵ^k. This is how we require that not only is the fault probability low but also, faults do not "conspire". The admissibility condition is satisfied if faults occur independently with probability $\leq \epsilon$.

Our goal is to build a circuit that will work correctly, with high probability, despite the ever-present noise: in other words, in which errors *do not accumulate*. This concept is formalized below.

Definition 6.12 *We say that the circuit $\mathcal{N}$ with output node w is (ϵ, δ)-**resilient** if for all inputs $\boldsymbol{x}$, all ϵ-admissible configurations $\mathbf{Y}$, we have $\mathbf{P}[\, Y_w \neq \mathrm{val}_{\boldsymbol{x}}(w) \,] \leq \delta$.*

Let us explore this concept. There is no (ϵ, δ)-resilient circuit with $\delta < \epsilon$, since even the last gate can fail with probability ϵ. So, let us, a little more generously, allow $\delta > 2\epsilon$. Clearly, for each circuit $\mathcal{N}$ and for each $\delta > 0$ we can choose ϵ small enough so that $\mathcal{N}$ is (ϵ, δ)-resilient. But this is not what we are after: hopefully, one does not need more reliable gates every time one builds a larger circuit. So, we hope to find a function

$$F(N, \delta)$$

and an $\epsilon_0 > 0$ with the property that for all $\epsilon < \epsilon_0$, $\delta \geq 2\epsilon$, every Boolean circuit $\mathcal{N}$ of size N there is some (ϵ, δ)-resilient circuit $\mathcal{N}'$ of size $F(N, \delta)$ computing the same function as $\mathcal{N}$. If we achieve this then we can say that we prevented the accumulation of errors. Of course, we want to make $F(N, \delta)$ relatively small, and ϵ_0 large (allowing more noise). The function $F(N, \delta)/N$ can be called the ***redundancy***: the factor by which we need to increase the size of the circuit to make it resilient. Note that the problem is nontrivial even with, say, $\delta = 1/3$. Unless the accumulation of errors is prevented we will lose gradually all information about the desired output, and no $\delta < 1/2$ could be guaranteed.

How can we correct errors? A simple idea is this: do "everything" 3 times and then continue with the result obtained by majority vote.

Definition 6.13 *For odd natural number d, a d-input majority gate is a Boolean function that outputs the value equal to the majority of its inputs.*

Note that a d-input majority can be computed using $O(d)$ gates of type AND and NOT.

Why should majority voting help? The following *informal discussion* helps understanding the benefits and pitfalls. Suppose for a moment that the output is a single bit. If the

probability of each of the three independently computed results failing is δ then the probability that at least 2 of them fails is bounded by $3\delta^2$. Since the majority vote itself can fail with some probability ϵ the total probability of failure is bounded by $3\delta^2 + \epsilon$. We decrease the probability δ of failure, provided the condition $3\delta^2 + \epsilon < \delta$ holds.

We found that if δ is small, then repetition and majority vote can "make it" smaller. Of course, in order to keep the error probability from accumulating, we would have to perform this majority operation repeatedly. Suppose, for example, that our computation has t stages. Our bound on the probability of faulty output after stage i is δ_i. We plan to perform the majority operation after each stage i. Let us perform stage i three times. The probability of failure is now bounded by

$$\delta_{i+1} = \delta_i + 3\delta^2 + \epsilon. \tag{6.16}$$

Here, the error probabilities of the different stages accumulate, and even if $3\delta^2 + \epsilon < \delta$ we only get a bound $\delta_t < (t-1)\delta$. So, this strategy will not work for arbitrarily large computations.

Here is a somewhat mad idea to avoid accumulation: repeat *everything* before the end of stage i three times, not only stage i itself. In this case, the growing bound (6.16) would be replaced with

$$\delta_{i+1} = 3(\delta_i + \delta)^2 + \epsilon.$$

Now if $\delta_i < \delta$ and $12\delta^2 + \epsilon < \delta$ then also $\delta_{i+1} < \delta$, so errors do not accumulate. But we paid an enormous price: the fault-tolerant version of the computation reaching stage $(i+1)$ is 3 times larger than the one reaching stage i. To make t stages fault-tolerant this way will cost a factor of 3^t in size. This way, the function $F(N, \delta)$ introduced above may become exponential in N.

The theorem below formalises the above discussion.

Theorem 6.14 *Let R be a finite and complete basis for Boolean functions. If $2\epsilon \le \delta \le 0.01$ then every function can be computed by an (ϵ, δ)-resilient circuit over R.*

Proof. For simplicity, we will prove the result for a complete basis that contains the three-argument majority function and contains not functions with more than three arguments. We also assume that faults occur independently.

Let $\mathcal{N}$ be a noise-free circuit of depth t computing function f. We will prove that there is an (ϵ, δ)-resilient circuit $\mathcal{N}'$ of depth $2t$ computing f. The proof is by induction on t. The sufficient conditions on ϵ and δ will emerge from the proof.

The statement is certainly true for $t = 1$, so suppose $t > 1$. Let g be the output gate of the circuit $\mathcal{N}$, then $f(\boldsymbol{x}) = g(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}))$. The subcircuits $\mathcal{N}_i$ computing the functions f_i have depth $\le t-1$. By the inductive assumption, there exist (ϵ, δ)-resilient circuits $\mathcal{N}'_i$ of depth $\le 2t-2$ that compute f_i. Let $\mathcal{M}$ be a new circuit containing copies of the circuits $\mathcal{N}'_i$ (with the corresponding input nodes merged), with a new node in which $f(\boldsymbol{x})$ is computed as g is applied to the outputs of $\mathcal{N}'_i$. Then the probability of error of $\mathcal{M}$ is at most $3\delta + \epsilon < 4\delta$ if $\epsilon < \delta$ since each circuit $\mathcal{N}'_i$ can err with probability δ and the node with gate g can fail with probability ϵ.

Let us now form $\mathcal{N}'$ by taking three copies of $\mathcal{M}$ (with the inputs merged) and adding a new node computing the majority of the outputs of these three copies. The error probability of $\mathcal{N}'$ is at most $3(4\delta)^2 + \epsilon = 48\delta^2 + \epsilon$. Indeed, error will be due to either a fault at the majority gate or an error in at least two of the three independent copies of $\mathcal{M}$. So under

condition

$$48\delta^2 + \epsilon \le \delta, \tag{6.17}$$

the circuit $\mathcal{N}'$ is (ϵ, δ)-resilient. This condition will be satisfied by $2\epsilon \le \delta \le 0.01$. ■

The circuit $\mathcal{N}'$ constructed in the proof above is at least 3^t times larger than $\mathcal{N}$. So, the redundancy is enormous. Fortunately, we will see a much more economical solution. But there are interesting circuits with small depth, for which the 3^t factor is not extravagant.

Theorem 6.15 *Over the complete basis consisting of all 3-argument Boolean functions, for all sufficiently small $\epsilon > 0$, if $2\epsilon \le \delta \le 0.01$ then for each n there is an (ϵ, δ)-resilient Boolean circuit of input size n, depth $\le 4\lg(n+1)$ and size $(n+1)^7$ outputting a near-majority (as given in Definition 6.8).*

Proof. Apply Theorem 6.14 to the circuit from part (1) of Theorem 6.9: it gives a new, $4\lg(n+1)$-deep (ϵ, δ)-resilient circuit computing a near-majority. The size of any such circuit with 3-input gates is at most $3^{4\lg(n+1)} = (n+1)^{4\lg 3} < (n+1)^7$. ■

Exercises

6.3-1 Exercise 6.2-5. suggests that the iterated majority vote $\mathcal{M}_3^r$ is not safe against manipulation. However, it works very well under some circumstances. Let the input to $\mathcal{M}_3^r$ be a vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ of independent Boolean random variables with $\mathbf{P}[\, X_i = 1 \,] = p < 1/6$. Denote the (random) output bit of the circuit by Z. Assuming that our majority gates can fail with probability $\le \epsilon \le p/2$ independently, prove

$$\mathbf{P}[\, Z = 1 \,] \le \max\{10\epsilon,\ 0.3(p/0.3)^{2^k}\}.$$

Hint: Define $g(p) = \epsilon + 3p^2$, $g_0(p) = p$, $g_{i+1}(p) = g(g_i(p))$, and prove $\mathbf{P}[\, Z = 1 \,] \le g_r(p)$.]

6.3-2 We say that a circuit $\mathcal{N}$ computes the function $f(x_1, \ldots, x_n)$ in an (ϵ, δ)-***input-robust*** way, if the following holds: For any input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, for any vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ of independent Boolean random variables "perturbing it" in the sense $\mathbf{P}[\, X_i \ne x_i \,] \le \epsilon$, for the output Y of circuit $\mathcal{N}$ on input $\boldsymbol{X}$ we have $\mathbf{P}[\, Y = f(\boldsymbol{x}) \,] \ge 1 - \delta$. Show that if the function $x_1 \oplus \cdots \oplus x_n$ is computable on an $(\epsilon, 1/4)$-input-robust circuit then $\epsilon \le 1/n$.

6.4. Safeguarding intermediate results

In this section, we will see ways to introduce fault-tolerance that scale up better. Namely, we will show:

Theorem 6.16 *There are constants R_0, ϵ_0 such that for*

$$F(n, \delta) = N \lg(n/\delta),$$

for all $\epsilon < \epsilon_0$, $\delta \ge 3\epsilon$, for every deterministic computation of size N there is an (ϵ, δ)-resilient computation of size $R_0 F(N, \delta)$ with the same result.

Let us introduce a concept that will simplify the error analysis of our circuits, making it independent of the input vector x.

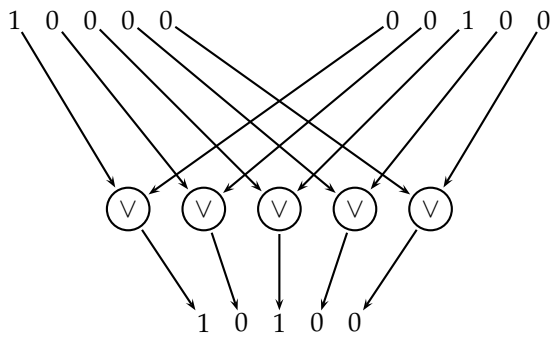


Figure 6.5. An executive organ.

Definition 6.17 *In a Boolean circuit $\mathcal{N}$, let us call a majority gate at a node v a* ***correcting majority gate*** *if for every input vector $\boldsymbol{x}$ of $\mathcal{N}$, all input wires of node v have the same value. Consider a computation of such a circuit $\mathcal{N}$. This computation will make some nodes and wires of $\mathcal{N}$* ***tainted****. We define taintedness by the following rules:*

1. *The input nodes are untainted.*
2. *If a node is tainted then all of its output wires are tainted.*
3. *A correcting majority gate is tainted if either it fails or a majority of its inputs are tainted.*
4. *Any other gate is tainted if either it fails or one of its inputs is tainted.*

Clearly, if for all ϵ-admissible random configurations the output is tainted with probability $\leq \delta$ then the circuit is (ϵ, δ)-resilient.

6.4.1. Cables

So far, we have only made use of redundancy idea (2) of the introduction to the present chapter: repeating computation steps. Let us now try to use idea (1) (keeping information in redundant form) in Boolean circuits. To protect information travelling from gate to gate, we replace each wire of the noiseless circuit by a "cable" of k wires (where k will be chosen appropriately). Each wire within the cable is supposed to carry the same bit of information, and we hope that a majority will carry this bit even if some of the wires fail.

Definition 6.18 *In a Boolean circuit $\mathcal{N}'$, a certain set of edges is allowed to be called a* ***cable*** *if in a noise-free computation of this circuit, each edge carries the same Boolean value. The* ***width*** *of the cable is its number of elements. Let us fix an appropriate constant threshold ϑ. Consider any possible computation of the noisy version of the circuit $\mathcal{N}'$, and a cable of width k in $\mathcal{N}'$. This cable will be called* ***ϑ-safe*** *if at most ϑk of its wires are tainted.*

Let us take a Boolean circuit $\mathcal{N}$ that we want to make resilient. As we replace wires of $\mathcal{N}$ with cables of $\mathcal{N}'$ containing k wires each, we will replace each noiseless 2-argument gate at a node v by a module called the ***executive organ*** of k gates, which for each $i = 1, \ldots, k$,

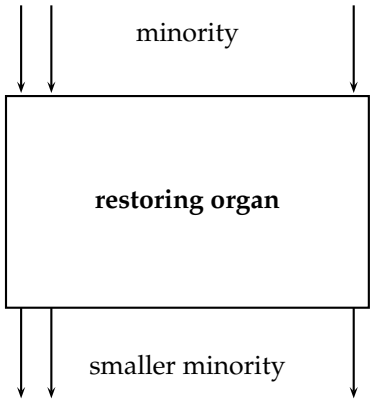


Figure 6.6. A restoring organ.

passes the ith wire both incoming cables into the ith node of the organ. Each of these nodes contains a gate of one and the same type b_v. The wires emerging from these nodes form the ***output cable*** of the executive organ.

The number of tainted wires in this output cable may become too high: indeed, if there were ϑk tainted wires in the x cable and also in the y cable then there could be as many as $2\vartheta k$ such wires in the $g(x,y)$ cable (not even counting the possible new taints added by faults in the executive organ). The crucial part of the construction is to attach to the executive organ a so-called ***restoring organ***: a module intended to decrease the taint in a cable.

6.4.2. Compressors

How to build a restoring organ? Keeping in mind that this organ itself must also work in noise, one solution is to build (for an appropriate δ') a special (ϵ, δ')-resilient circuit that computes the near-majority of its k inputs in k independent copies. Theorem 6.15 provides a circuit of size $k(k+1)^7$ to do this.

It turns out that, at least asymptotically, there is a better solution. We will look for a *very simple* restoring organ: one whose own noise we can analyse easily. What could be simpler than a circuit having only *one level* of gates? We fix an odd positive integer constant d (for example, $d = 3$). Each gate of our organ will be a d-input majority gate.

Definition 6.19 *A **multigraph** is a graph in which between any two vertices there may be several edges, not just 0 or 1. Let us call a bipartite multigraph with k inputs and k outputs, **d-half-regular**, if each output node has degree d. Such a graph is a (d, α, γ, k)**-compressor** if it has the following property: for every set E of at most $\leq \alpha k$ inputs, the number of those output points connected to at least $d/2$ elements of E (with multiplicity) is at most $\gamma\alpha k$.*

The compressor property is interesting generally when $\gamma < 1$. For example, in an $(5, 0.1, 0.5, k)$-compressor the outputs have degree 5, and the majority operation in these nodes decreases every error set confined to 10% of all input to just 5% of all outputs. A compressor with the right parameters could serve as our restoring organ: it decreases a minority to a smaller minority and may in this way restore the safety of a cable. But, are there

compressors?

Theorem 6.20 *For all $\gamma < 1$, all integers d with*

$$1 < \gamma(d-1)/2, \tag{6.18}$$

there is an α such that for all integer $k > 0$ there is a (d, α, γ, k)-compressor.

Note that for $d = 3$, the theorem does not guarantee a compressor with $\gamma < 1$.

Proof. We will not give an explicit construction for the multigraph, we will just show that it exists. We will select a d-half-regular multigraph randomly (each such multigraph with the same probability), and show that it will be a (d, α, γ, k)-compressor with positive probability. This proof method is called the ***probabilistic method***. Let

$$s = \lfloor d/2 \rfloor.$$

Our construction will be somewhat more general, allowing $k' \neq k$ outputs. Let us generate a random bipartite d-half-regular multigraph with k inputs and k' outputs in the following way. To each output, we draw edges from d random input nodes chosen independently and with uniform distribution over all inputs. Let A be an input set of size αk, let v be an output node and let E_v be the event that v has $s+1$ or more edges from A. Then we have

$$\mathbf{P}(E_v) \leq \binom{d}{s+1}\alpha^{s+1} = \binom{d}{s}\alpha^{s+1} =: p.$$

On the average (in expected value), the event E_v will occur for pk' different output nodes v. For an input set A, let F_A be the event that the set of nodes v for which E_v holds has size $> \gamma\alpha k'$. By inequality (6.7) we have

$$\mathbf{P}(F_A) \leq \left(\frac{ep}{\gamma\alpha}\right)^{k'\gamma\alpha}.$$

The number M of sets A of inputs with $\leq \alpha k$ elements is, using inequality (6.8),

$$M \leq \sum_{i \leq \alpha k}\binom{k}{i} \leq \left(\frac{e}{\alpha}\right)^{\alpha k}.$$

The probability that our random graph is not a compressor is at most as large as the probability that there is at least one input set A for which event F_A holds. This can be bounded by

$$M \cdot \mathbf{P}(F_A) \leq e^{-\alpha D k'}$$

where

$$D = -(\gamma s - k/k')\ln\alpha - \gamma(\ln\tbinom{d}{s} - \ln\gamma + 1) - k/k'.$$

As we decrease α the first term of this expression dominates. Its coefficient is positive according to the assumption (6.18). We will have $D > 0$ if

$$\alpha < \exp\left(-\frac{\gamma(\ln\tbinom{d}{s} - \ln\gamma + 1) + k/k'}{\gamma s - k/k'}\right).$$

■

Example 6.4 Choosing $\gamma = 0.4$, $d = 7$, the value $\alpha = 10^{-7}$ will work.

We turn a (d, α, γ, k)-compressor into a restoring organ $\mathcal{R}$, by placing d-input majority gates into its outputs. If the majority elements sometimes fail then the output of $\mathcal{R}$ is random. Assume that at most αk inputs of $\mathcal{R}$ are tainted. Then $(\gamma + \rho)\alpha k$ outputs can only be tainted if $\alpha\rho k$ majority gates fail. Let

$$p_{\mathcal{R}}$$

be the probability of this event. Assuming that the gates of $\mathcal{R}$ fail independently with probability $\leq \epsilon$, inequality (6.7) gives

$$p_{\mathcal{R}} \leq \left(\frac{e\epsilon}{\alpha\rho}\right)^{\alpha\rho k}. \tag{6.19}$$

Example 6.5 Choose $\gamma = 0.4$, $d = 7$, $\alpha = 10^{-7}$ as in Example 6.4., further $\rho = 0.14$ (this will satisfy the inequality (6.20) needed later). With $\epsilon = 10^{-9}$, we get $p_{\mathcal{R}} \leq e^{-10^{-8}k}$.

The attractively small degree $d = 7$ led to an extremely unattractive probability bound on the failure of the whole compressor. This bound does decrease exponentially with cable width k, but only an extremely large k would make it small.

Example 6.6 Choosing again $\gamma = 0.4$, but $d = 41$ (voting in each gate of the compressor over 41 wires instead of 7), leads to somewhat more realistic results. This choice allows $\alpha = 0.15$. With $\rho = 0.14$, $\epsilon = 10^{-9}$ again, we get $p_{\mathcal{R}} \leq e^{-0.32k}$.

These numbers look less frightening, but we will still need many scores of wires in the cable to drive down the probability of compression failure. And although in practice our computing components fail with frequency much less than 10^{-9}, we may want to look at the largest ϵ that still can be tolerated.

6.4.3. Propagating safety

Compressors allow us to construct a reliable Boolean circuit all of whose cables are safe.

Definition 6.21 *Given a Boolean circuit $\mathcal{N}$ with a single bit of output (for simplicity), a cable width k and a Boolean circuit $\mathcal{R}$ with k inputs and k outputs, let*

$$\mathcal{N}' = \mathrm{Cab}(\mathcal{N}, \mathcal{R})$$

be the Boolean circuit that we obtain as follows. The input nodes of $\mathcal{N}'$ are the same as those of $\mathcal{N}$. We replace each wire of $\mathcal{N}$ with a cable of width k, and each gate of $\mathcal{N}$ with an executive organ followed by a restoring organ that is a copy of the circuit $\mathcal{R}$. The new circuit has k outputs: the outputs of the restoring organ of $\mathcal{N}'$ belonging to the last gate of $\mathcal{N}$.

In noise-free computations, on every input, the output of $\mathcal{N}'$ is the same as the output of $\mathcal{N}$, but in k identical copies.

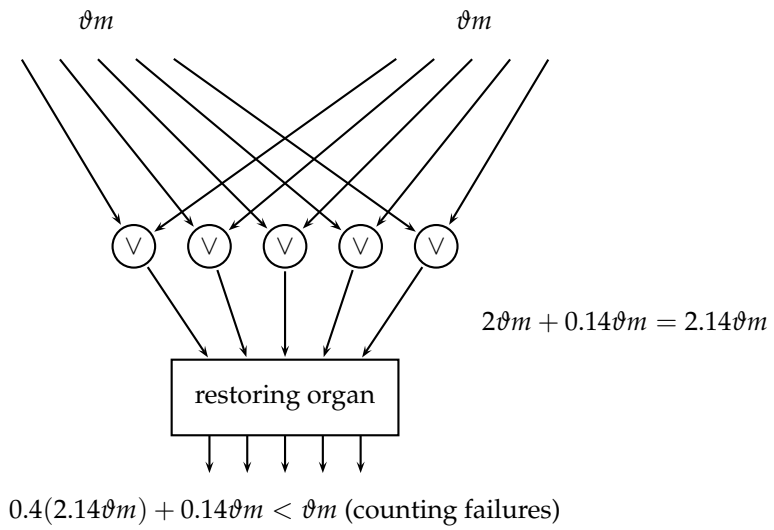


Figure 6.7. An executive organ followed by a restoring organ.

Lemma 6.22 *There are constants $d, \epsilon_0, \vartheta, \rho > 0$ and for every cable width k a circuit $\mathcal{R}$ of size $2k$ and gate size $\leq d$ with the following property. For every Boolean circuit $\mathcal{N}$ of gate size ≤ 2 and number of nodes N, for every $\epsilon < \epsilon_0$, for every ϵ-admissible configuration of $\mathcal{N}' = \mathrm{Cab}(\mathcal{N}, \mathcal{R})$, the probability that not every cable of $\mathcal{N}'$ is ϑ-safe is $< 2N(\frac{e\epsilon}{\vartheta\rho})^{\vartheta\rho k}$.*

Proof. We know that there are d, α and $\gamma < 1/2$ with the property that for all k a (d, α, γ, k)-compressor exists. Let ρ be chosen to satisfy

$$\gamma(2 + \rho) + \rho \leq 1, \tag{6.20}$$

and define

$$\vartheta = \alpha/(2 + \rho). \tag{6.21}$$

Let $\mathcal{R}$ be a restoring organ built from a (d, α, γ, k)-compressor. Consider a gate v of circuit $\mathcal{N}$, and the corresponding executive organ and restoring organ in $\mathcal{N}'$. Let us estimate the probability of the event E_v that the input cables of this combined organ are ϑ-safe but its output cable is not. Assume that the two incoming cables are safe: then at most $2\vartheta k$ of the outputs of the executive organ are tainted due to the incoming cables: new taint can still occur due to failures. Let E_{v1} be the event that the executive organ taints at least $\rho\vartheta k$ more of these outputs. Then $\mathbf{P}(E_{v1}) \leq (\frac{e\epsilon}{\rho\vartheta})^{\rho\vartheta k}$, using the estimate (6.19). The outputs of the executive organ are the inputs of the restoring organ. If no more than $(2 + \rho)\vartheta k = \alpha k$ of these are tainted then, in case the organ operates perfectly, it would decrease the number of tainted wires to $\gamma(2 + \rho)\vartheta k$. Let E_{v2} be the event that the restoring organ taints an additional $\rho\vartheta k$ of these wires. Then again, $\mathbf{P}(E_{v2}) \leq (\frac{e\epsilon}{\rho\vartheta})^{\rho\vartheta k}$. If neither E_{v1} nor E_{v2} occur then at most $\gamma(2 + \rho)\vartheta k + \rho\vartheta k \leq \vartheta k$ (see (6.20)) tainted wires emerge from the restoring organ, so the outgoing cable is safe. Therefore $E_v \subset E_{v1} \cup E_{v2}$ and hence $\mathbf{P}(E_v) \leq 2(\frac{e\epsilon}{\rho\vartheta})^{\rho\vartheta k}$.

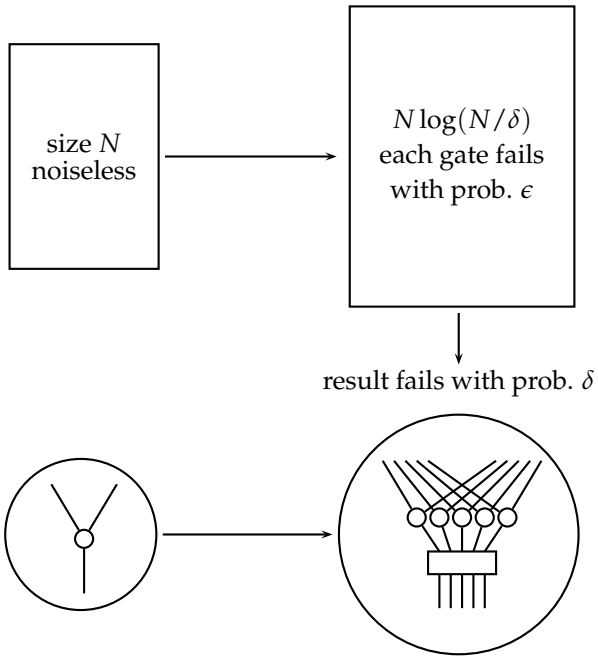


Figure 6.8. Reliable circuit from a fault-free circuit.

Let $V = \{1, \ldots, N\}$ be the nodes of the circuit $\mathcal{N}$. Since the incoming cables of the whole circuit $\mathcal{N}'$ are safe, the event that there is some cable that is not safe is contained in $E_1 \cup E_2 \cup \cdots \cup E_N$; hence the probability is bounded by $2N(\frac{e\epsilon}{\rho\vartheta})^{\rho\vartheta k}$. ∎

6.4.4. Endgame

Proof. of Theorem 6.16 We will prove the theorem only for the case when our computation is a Boolean circuit with a single bit of output. The generalisation with more bits of output is straightforward. The proof of Lemma 6.22 gives us a circuit $\mathcal{N}'$ whose output cable is safe except for an event of probability $< 2N(\frac{e\epsilon}{\rho\vartheta})^{\rho\vartheta k}$. Let us choose k in such a way that this becomes $\leq \delta/3$:

$$k \geq \frac{\lg(6N/\delta)}{\rho\vartheta \log \frac{\rho\vartheta}{e\epsilon_0}}. \tag{6.22}$$

It remains to add a little circuit to this output cable to extract from it the majority reliably. This can be done using Theorem 6.15, adding a small extra circuit of size $(k+1)^7$ that can be called the ***coda*** to $\mathcal{N}'$. Let us call the resulting circuit $\mathcal{N}''$.

The probability that the output cable is unsafe is $< \delta/3$. The probability that the output cable is safe but the "coda" circuit fails is bounded by 2ϵ. So, the probability that $\mathcal{N}''$ fails is $\leq 2\epsilon + \delta/3 \leq \delta$, by the assumption $\delta \geq 3\epsilon$.

Let us estimate the size of $\mathcal{N}''$. By (6.22), we can choose cable width $k = O(\log(N/\delta))$.

We have $|\mathcal{N}'| \leq 2kN$, hence

$$|\mathcal{N}''| \leq 2kN + (k+1)^7 = O(N \lg(N/\delta)).$$

■

Example 6.7 Take the constants of Example 6.6., with ϑ defined in equation (6.21): then $\epsilon_0 = 10^{-9}$, $d = 41$, $\gamma = 0.4$, $\rho = 0.14$, $\alpha = 0.15$, $\vartheta = 0.07$, giving

$$\frac{1}{\rho\vartheta \ln \frac{\rho\vartheta}{e\epsilon_0}} \approx 6.75,$$

so making k as small as possible (ignoring that it must be integer), we get $k \approx 6.75 \ln(N/\delta)$. With $\delta = 10^{-8}$, $N = 10^{12}$ this allows $k = 323$. In addition to this truly unpleasant cable size, the size of the "coda" circuit is $(k+1)^7 \approx 4 \cdot 10^{17}$, which dominates the size of the rest of $\mathcal{N}''$ (though as $N \to \infty$ it becomes asymptotically negligible).

As Example 6.7. shows, the actual price in redundancy computable from the proof is unacceptable in practice. The redundancy $O(\lg(N/\delta))$ sounds good, since it is only logarithmic in the size of the computation, and by choosing a rather large majority gate (41 inputs), the factor 6.75 in the $O(\cdot)$ here also does not look bad; still, we do not expect the final price of reliability to be this high. How much can this redundancy improved by optimisation or other methods? Problem 6-6. shows that in a slightly more restricted error model (all faults are independent and have the same probability), with more randomisation, better constants can be achieved. Exercises 6.4-1., 6.4-2. and 6.4-6. are concerned with an improved construction for the "coda" circuit. Exercise 6.5-2. shows that the coda circuit can be omitted completely. But none of these improvements bring redundancy to acceptable level. Even aside from the discomfort caused by their random choice (this can be helped), concentrators themselves are rather large and unwieldy. The problem is probably with using circuits as a model for computation. There is no natural way to break up a general circuit into subunits of non-constant size in order to deal with the reliability problem in modular style.

6.4.5. The construction of compressors

This subsection is sketchier than the preceding ones, and assumes some knowledge of linear algebra.

We have shown that compressors exist. How expensive is it to find a (d, α, γ, k)-compressor, say, with $d = 41$, $\alpha = 0.15$, $\gamma = 0.4$, as in Example 6.6.? In a deterministic algorithm, we could search through all the approximately d^k d-half-regular bipartite graphs. For each of these, we could check all possible input sets of size $\leq \alpha k$: as we know, their number is $\leq (e/\alpha)^{\alpha k} < 2^k$. The cost of checking each subset is $O(k)$, so the total number of operations is $O(k(2d)^k)$. Though this number is exponential in k, recall that in our error-correcting construction, $k = O(\lg(N/\delta))$ for the size N of the noiseless circuit: therefore the total number of operations needed to find a compressor is polynomial in N.

The proof of Theorem 6.20 shows that a randomly chosen d-half-regular bipartite graph is a compressor with large probability. Therefore there is a faster, randomised algorithm for finding a compressor. Pick a random d-half-regular bipartite graph, check if it is a compressor: if it is not, repeat. We will be done in a constant expected number of repetitions. This

is a faster algorithm, but is still exponential in k, since each checking takes $\Omega(k(e/\alpha)^{\alpha k})$ operations.

Is it possible to construct a compressor explicitly, avoiding any search that takes exponential time in k? The answer is yes. We will show here only, however, that the compressor property is implied by a certain property involving linear algebra, which can be checked in polynomial time. Certain explicitly constructed graphs are known that possess this property. These are generally sought after not so much for their compressor property as for their *expander* property (see Exercise 6.4-3.).

For vectors $\boldsymbol{v}, \boldsymbol{w}$, let $(\boldsymbol{v}, \boldsymbol{w})$ denote their inner product. A d-half-regular bipartite multigraph with $2k$ nodes can be defined by an ***incidence matrix*** $\boldsymbol{M} = (m_{ij})$, where m_{ij} is the number of edges connecting input j to output i. Let $\boldsymbol{e}$ be the vector $(1, 1, \ldots, 1)^T$. Then $\boldsymbol{Me} = d\boldsymbol{e}$, so $\boldsymbol{e}$ is an ***eigenvector*** of $\boldsymbol{M}$ with ***eigenvalue*** d. Moreover, d is the largest eigenvalue of $\boldsymbol{M}$. Indeed, denoting by $|\boldsymbol{x}|_1 = \sum_i |x_i|$ for any row vector $\boldsymbol{x} = (x_1, \ldots, x_k)$, we have $|\boldsymbol{xM}|_1 \le |\boldsymbol{x}|_1$.

Theorem 6.23 *Let G be a multigraph defined by the matrix $\boldsymbol{M}$. For all $\gamma > 0$, and*

$$\mu < d\sqrt{\gamma}/2, \tag{6.23}$$

there is an $\alpha > 0$ such that if the second largest eigenvalue of the matrix $\boldsymbol{M}^T\boldsymbol{M}$ is μ^2 then G is a (d, α, γ, k)-compressor.

Proof. The matrix $\boldsymbol{M}^T\boldsymbol{M}$ has largest eigenvalue d^2. Since it is symmetric, it has a basis of orthogonal eigenvectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k$ of unit length with corresponding nonnegative eigenvalues

$$\lambda_1^2 \ge \cdots \ge \lambda_k^2$$

where $\lambda_1 = d$ and $\boldsymbol{e}_1 = \boldsymbol{e}/\sqrt{k}$. Recall that in the orthonormal basis $\{\boldsymbol{e}_i\}$, any vector $\boldsymbol{f}$ can be written as $\boldsymbol{f} = \sum_i (\boldsymbol{f}, \boldsymbol{e}_i)\boldsymbol{e}_i$. For an arbitrary vector $\boldsymbol{f}$, we can estimate $|\boldsymbol{Mf}|^2$ as follows.

$$\begin{aligned} |\boldsymbol{Mf}|^2 = (\boldsymbol{Mf}, \boldsymbol{Mf}) = (\boldsymbol{f}, \boldsymbol{M}^T\boldsymbol{Mf}) &= \sum_i \lambda_i^2 (\boldsymbol{f}, \boldsymbol{e}_i)^2 \\ &\le d^2(\boldsymbol{f}, \boldsymbol{e}_1)^2 + \mu^2 \sum_{i>1} (\boldsymbol{f}, \boldsymbol{e}_i)^2 \le d^2(\boldsymbol{f}, \boldsymbol{e}_1)^2 + \mu^2(\boldsymbol{f}, \boldsymbol{f}) \\ &= d^2(\boldsymbol{f}, \boldsymbol{e})^2/k + \mu^2(\boldsymbol{f}, \boldsymbol{f}). \end{aligned}$$

Let now $A \subset \{1, \ldots, k\}$ be a set of size αk and $\boldsymbol{f} = (f_1, \ldots, f_k)^T$ where $f_j = 1$ for $j \in A$ and 0 otherwise. Then, coordinate i of $\boldsymbol{Mf}$ counts the number d_i of edges coming from the set A to the node i. Also, $(\boldsymbol{f}, \boldsymbol{e}) = (\boldsymbol{f}, \boldsymbol{f}) = |A|$, the number of elements of A. We get

$$\sum_i d_i^2 = |\boldsymbol{Mf}|^2 \le d^2(\boldsymbol{f}, \boldsymbol{e})^2/k + \mu^2(\boldsymbol{f}, \boldsymbol{f}) = d^2\alpha^2 k + \mu^2 \alpha k,$$

$$k^{-1} \sum_i (d_i/d)^2 \le \alpha^2 + (\mu/d)^2 \alpha.$$

Suppose that there are $c\alpha k$ nodes i with $d_i > d/2$, then this says

$$c\alpha \le 4(\mu/d)^2 \alpha + 4\alpha^2.$$

Since (6.23) implies $4(\mu/d)^2 < \gamma$, it follows that $\boldsymbol{M}$ is a $(d, \alpha, \gamma, k, k)$-compressor for small enough α. ■

It is actually sufficient to look for graphs with large k and $\mu/d < c < 1$ where d, c are constants. To see this, let us define the product of two bipartite multigraphs with $2k$ vertices by the multigraph belonging to the product of the corresponding matrices.

Suppose that $\boldsymbol{M}$ is symmetric: then its second largest eigenvalue is μ and the ratio of the two largest eigenvalues of $\boldsymbol{M}^r$ is $(\mu/d)^r$. Therefore using $\boldsymbol{M}^r$ for a sufficiently large r as our matrix, the condition (6.23) can be satisfied. Unfortunately, taking the power will increase the degree d, taking us probably even farther away from practical realisability.

We found that there is a construction of a compressor with the desired parameters as soon as we find multigraphs with arbitrarily large sizes $2k$, with symmetric matrices $\boldsymbol{M}_k$ and with a ratio of the two largest eigenvalues of $\boldsymbol{M}_k$ bounded by a constant $c < 1$ independent of k. There are various constructions of such multigraphs (see the references in the historical overview). The estimation of the desired eigenvalue quotient is never very simple.

Exercises

6.4-1 The proof of Theorem 6.16 uses a "coda" circuit of size $(k+1)^7$. Once we proved this theorem we could, of course, apply it to the computation of the final majority itself: this would reduce the size of the coda circuit to $O(k \lg k)$. Try out this approach on the numerical examples considered above to see whether it results in a significant improvement.

6.4-2 The proof of Theorem 6.20 provided also bipartite graphs with the compressor property, with k inputs and $k' < 0.8k$ outputs. An idea to build a smaller "coda" circuit in the proof of Theorem 6.16 is to concatenate several such compressors, decreasing the number of cables in a geometric series. Explore this idea, keeping in mind, however, that as k decreases, the "exponential" error estimate in inequality (6.19) becomes weaker.

6.4-3 Let us call a d-halfregular bipartite multigraph with a set A of k inputs and a set B of k outputs a (d, α, λ, k)-***expander*** if it has the following property: for every set $E \subset A$ with $|E| \leq \alpha k$, the number of those elements of B connected to E is at least $\lambda \alpha k$. Prove the following theorem analogous to Theorem 6.20: For all $\lambda < d$, there is an α such that for all $k > 0$ there is a (d, α, λ, k)-expander. [Hint: Analogously to the proof of Theorem 6.20, show that a random d-half-regular multigraph is an expander with large probability.]

6.4-4 In a noisy Boolean circuit, let $F_v = 1$ if the gate at vertex v fails and 0 otherwise. Further, let $T_v = 1$ if v is tainted, and 0 otherwise. Suppose that the distribution of the random variables F_v does not depend on the Boolean input vector. Show that then the joint distribution of the random variables T_v is also independent of the input vector.

6.4-5 This exercise extends the result of Exercise 6.3-1. to random input vectors: it shows that if a random input vector has only a small number of errors, then the iterated majority vote $\mathcal{M}_3^r$ of Exercise 6.2-5. may still work for it, if we rearrange the input wires randomly. Let $k = 3^r$, and let $\boldsymbol{j} = (j_1, \ldots, j_k)$ be a vector of integers $j_i \in \{1, \ldots, k\}$. We define a Boolean circuit $C(\boldsymbol{j})$ as follows. This circuit takes input vector $\boldsymbol{x} = (x_1, \ldots, x_k)$, computes the vector $\boldsymbol{y} = (y_1, \ldots, y_k)$ where $y_i = x_{j_i}$ (in other words, just leads a wire from input node j_i to an "intermediate node" i) and then inputs $\boldsymbol{y}$ into the circuit $\mathcal{M}_3^r$.

Denote the (possibly random) output bit of $C(\boldsymbol{j})$ by Z. For any fixed input vector $\boldsymbol{x}$, assuming that our majority gates can fail with probability $\leq \epsilon \leq \alpha/2$ independently, denote $q(\boldsymbol{j}, \boldsymbol{x}) := \mathbf{P}[\, Z = 1 \,]$. Assume that the input is a vector $\boldsymbol{X} = (X_1, \ldots, X_k)$ of (not necessarily independent) Boolean random variables, with $p(\boldsymbol{x}) := \mathbf{P}[\, \boldsymbol{X} = \boldsymbol{x} \,]$. Denoting $|\boldsymbol{X}| = \sum_i X_i$,

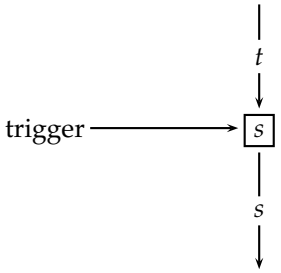


Figure 6.9. A shift register.

assume $\mathbf{P}[\,|\boldsymbol{X}| > \alpha k\,] \le \rho < 1$. Prove that there is a choice of the vector $\boldsymbol{j}$ for which

$$\sum_{\boldsymbol{x}} p(\boldsymbol{x})q(\boldsymbol{j},\boldsymbol{x}) \le \rho + \max\{10\epsilon,\ 0.3(\alpha/0.3)^{2^k}\}.$$

The choice may depend on the distribution of the random vector $\boldsymbol{X}$. [Hint: Choose the vector $\boldsymbol{j}$ (and hence the circuit $C(\boldsymbol{j})$) randomly, as a random vector $\boldsymbol{J} = (J_1, \ldots, J_k)$ where the variables J_i are independent and uniformly distributed over $\{1, \ldots, k\}$, and denote $s(\boldsymbol{j}) := \mathbf{P}[\,\boldsymbol{J} = \boldsymbol{j}\,]$. Then prove

$$\sum_{j} s(\boldsymbol{j}) \sum_{\boldsymbol{x}} p(\boldsymbol{x})q(\boldsymbol{j},\boldsymbol{x}) \le \rho + \max\{10\epsilon,\ 0.3(\alpha/0.3)^{2^k}\}.$$

For this, interchange the averaging over $\boldsymbol{x}$ and $\boldsymbol{j}$. Then note that $\sum_j s(\boldsymbol{j})q(\boldsymbol{j},\boldsymbol{x})$ is the probability of $Z = 1$ when the "wires" J_i are chosen randomly "on the fly" during the computation of the circuit.]

6.4-6 Taking the notation of Exercise 6.4-4. suppose, like there, that the random variables F_v are independent of each other, and their distribution does not depend on the Boolean input vector. Take the Boolean circuit $\mathrm{Cab}(\mathcal{N},\mathcal{R})$ introduced in Definition 6.21, and define the random Boolean vector $\boldsymbol{T} = (T_1, \ldots, T_k)$ where $T_i = 1$ if and only if the ith output node is tainted. Apply Exercise 6.4-5. to show that there is a circuit $C(\boldsymbol{j})$ that can be attached to the output nodes to play the role of the "coda" circuit in the proof of Theorem 6.16. The size of $C(\boldsymbol{j})$ is only linear in k, not $(k+1)^7$ as for the coda circuit in the proof there. But, we assumed a little more about the fault distribution, and also the choice of the "wiring" $\boldsymbol{j}$ depends on the circuit $\mathrm{Cab}(\mathcal{N},\mathcal{R})$.

6.5. The reliable storage problem

6.5.1. Clocked circuits

An obvious element of ordinary computations is missing from the above described Boolean circuit model: *repetition*. If we want to repeat some computation steps, then we need to introduce *timing* into the work of computing elements and to *store* the partial results between

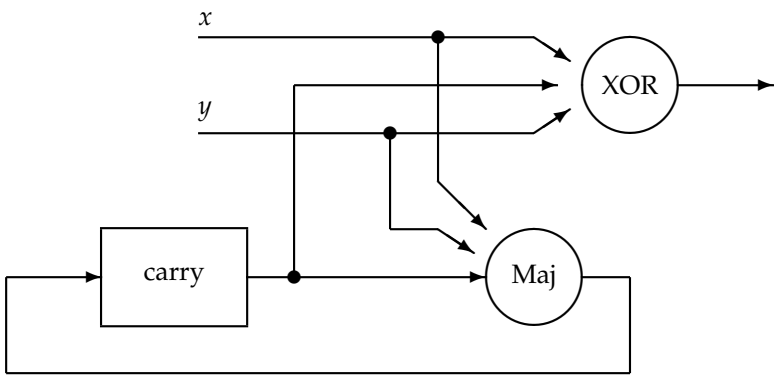


Figure 6.10. Part of a circuit which computes the sum of two binary numbers x, y. We feed the digits of x and y beginning with the lowest-order ones, at the input nodes. The digits of the sum come out on the output edge. A shift register holds the carry.

consecutive steps. Let us look at the drawings of the circuit designer again. We will see components like in Figure 6.9, with one ingoing edge and no operation associated with them; these will be called ***shift registers***. The shift registers are controlled by one central ***clock*** (invisible on the drawing). At each clock pulse, the assignment value on the incoming edge jumps onto the outgoing edges and "stays in" the register. Figure 6.10 shows how shift registers may be used inside a circuit.

Definition 6.24 *A **clocked circuit** over a complete basis Q is given by a tuple just like a Boolean circuit in (6.11). Also, the circuit defines a graph $G = (V, E)$ similarly. Recall that we identified nodes with the natural numbers $1, \dots, N$. To each non-input node v either a gate b_v is assigned as before, or a **shift register**: in this case $k_v = 1$ (there is only one argument). We do not require the graph to be acyclic, but we do require every directed cycle (if there is any) to pass through at least one shift register.*

*The circuit works in a sequence $t = 0, 1, 2, \dots$ of **clock cycles**. Let us denote the input vector at clock cycle t by $\boldsymbol{x}^t = (x_1^t, \dots, x_n^t)$, the shift register states by $\boldsymbol{s}^t = (s_1^t, \dots, s_k^t)$, and the output vector by $\boldsymbol{y}^t = (y_1^t, \dots, y_m^t)$. The part of the circuit going from the inputs and the shift registers to the outputs and the shift registers defines two Boolean vector functions $\lambda : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ and $\tau : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^k$. The operation of the clocked circuit is described by the following equations (see Figure 6.11, which does not show any inputs and outputs).*

$$\boldsymbol{y}^t = \lambda(\boldsymbol{s}^t, \boldsymbol{x}^t), \quad \boldsymbol{s}^{t+1} = \tau(\boldsymbol{s}^t, \boldsymbol{x}^t). \tag{6.24}$$

Frequently, we have no inputs or outputs during the work of the circuit, so the equations (6.24) can be simplified to

$$\boldsymbol{s}^{t+1} = \tau(\boldsymbol{s}^t). \tag{6.25}$$

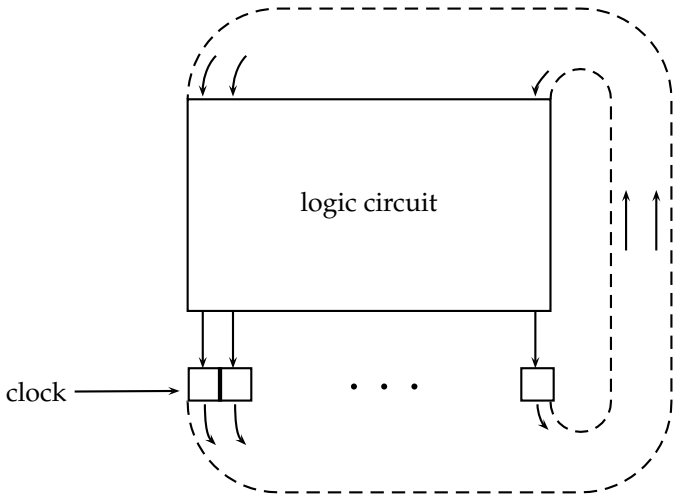


Figure 6.11. A "computer" consists of some memory (shift registers) and a Boolean circuit operating on it. We can define the ***size of computation*** as the size of the computer times the number of steps.

How to use a clocked circuit described by this equation for computation? We write some initial values into the shift registers, and propagate the assignment using the gates, for the given clock cycle. Now we send a clock pulse to the register, causing it to write new values to their output edges (which are identical to the input edges of the circuit). After this, the new assignment is computed, and so on.

How to compute a *function* $f(x)$ with the help of such a circuit? Here is a possible convention. We enter the input x (only in the first step), and then run the circuit, until it signals at an extra output edge when desired result $f(x)$ can be received from the other output nodes.

Example 6.8 This example uses a convention different from the above described one: new input bits are supplied in every step, and the output is also delivered continuously. For the binary adder of Figure 6.10, let u^t and v^t be the two input bits in cycle t, let c^t be the content of the carry, and w^t be the output in the same cycle. Then the equations (6.24) now have the form

$$w^t = u^t \oplus v^t \oplus c^t, \quad c^{t+1} = \mathrm{Maj}(u^t, v^t, c^t),$$

where Maj is the majority operation.

6.5.2. Storage

A clocked circuit is an interesting parallel computer but let us pose now a task for it that is trivial in the absence of failures: information storage. We would like to store a certain amount of information in such a way that it can be recovered after some time, despite failures

in the circuit. For this, the transition function τ introduced in (6.25) cannot be just the identity: it will have to perform some *error-correcting* operations. The restoring organs discussed earlier are natural candidates. Indeed, suppose that we use k memory cells to store a bit of information. We can call the content of this k-tuple ***safe*** when the number of memory cells that dissent from the correct value is under some threshold ϑk. Let the rest of the circuit be a restoring organ built on a (d, α, γ, k)-compressor with $\alpha = 0.9\vartheta$. Suppose that the input cable is safe. Then the probability that after the transition, the new output cable (and therefore the new state) is not safe is $O(e^{-ck})$ for some constant c. Suppose we keep the circuit running for t steps. Then the probability that the state is not safe in some of these steps is $O(te^{-ck})$ which is small as long as t is significantly smaller than e^{ck}. When storing m bits of information, the probability that any of the bits loses its safety in some step is $O(mte^{-cm})$.

To make this discussion rigorous, an error model must be introduced for clocked circuits. Since we will only consider simple transition functions τ like the majority vote above, with a single computation step between times t and $t+1$, we will make the model also very simple.

Definition 6.25 *Consider a clocked circuit described by equation (6.25), where at each time instant $t = 0, 1, 2, \ldots$, the configuration is described by the bit vector $\boldsymbol{s}^t = (s_1^t, \ldots, s_n^t)$. Consider a sequence of random bit vectors $\boldsymbol{Y}^t = (Y_1^t, \ldots, Y_n^t)$ for $t = 0, 1, 2, \ldots$. Similarly to (6.14) we define*

$$Z_{i,t} = \tau(\boldsymbol{Y}^{t-1}) \oplus Y_i^t. \tag{6.26}$$

*Thus, $Z_{i,t} = 1$ says that a **failure** occurs at the space-time point (i, t). The sequence $\{\boldsymbol{Y}^t\}$ will be called **ϵ-admissible** if (6.15) holds for every set C of space-time points with $t > 0$.*

By the just described construction, it is possible to keep m bits of information for T steps in

$$O(m \lg(mT)) \tag{6.27}$$

memory cells. More precisely, the cable $\boldsymbol{Y}^T$ will be safe with large probability in any admissible evolution $\boldsymbol{Y}^t$ $(t = 0, \ldots, T)$.

Cannot we do better? The reliable information storage problem is related to the problem of ***information transmission***: given a ***message*** $\boldsymbol{x}$, a ***sender*** wants to transmit it to a ***receiver*** through a ***noisy channel***. Only now sender and receiver are the same person, and the noisy channel is just the passing of time. Below, we develop some basic concepts of reliable information transmission, and then we will apply them to the construction of a reliable data storage scheme that is more economical than the above seen naive, repetition-based solution.

6.5.3. Error-correcting codes

Error detection

To protect information, we can use redundancy in a way more efficient than repetition. We might even add only a single redundant bit to our message. Let $\boldsymbol{x} = (x_1, \ldots, x_6)$, $(x_i \in \{0, 1\})$ be the word we want to protect. Let us create the ***error check bit***

$$x_7 = x_1 \oplus \cdots \oplus x_6.$$

For example, $\boldsymbol{x} = 110010$, $\boldsymbol{x}' = 1100101$. Our ***codeword*** $\boldsymbol{x}' = (x_1, \dots, x_7)$ will be subject to noise and it turns into a new word, $\boldsymbol{y}$. If $\boldsymbol{y}$ differs from $\boldsymbol{x}'$ in a single changed (not deleted or added) bit then we will ***detect*** this, since then $\boldsymbol{y}$ violates the ***error check relation***

$$y_1 \oplus \cdots \oplus y_7 = 0.$$

We will not be able to correct the error, since we do not know which bit was corrupted.

Correcting a single error

To also ***correct*** corrupted bits, we need to add more error check bits. We may try to add two more bits:

$$x_8 = x_1 \oplus x_3 \oplus x_5,$$
$$x_9 = x_1 \oplus x_2 \oplus x_5 \oplus x_6.$$

Then an uncorrupted word $\boldsymbol{y}$ must satisfy the error check relations

$$y_1 \oplus \cdots \oplus y_7 = 0,$$
$$y_1 \oplus y_3 \oplus y_5 \oplus y_8 = 0,$$
$$y_1 \oplus y_2 \oplus y_5 \oplus y_6 \oplus y_9 = 0,$$

or, in matrix notation $\boldsymbol{H}\boldsymbol{y} \bmod 2 = 0$, where

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = (\boldsymbol{h}_1, \dots, \boldsymbol{h}_9).$$

Note $\boldsymbol{h}_1 = \boldsymbol{h}_5$. The matrix $\boldsymbol{H}$ is called the ***error check matrix***, or ***parity check matrix***.

Another way to write the error check relations is

$$y_1\boldsymbol{h}_1 \oplus \cdots \oplus y_5\boldsymbol{h}_5 \oplus \cdots \oplus y_9\boldsymbol{h}_9 = 0.$$

Now if $\boldsymbol{y}$ is corrupted, even if only in a single position, unfortunately we still cannot correct it: since $\boldsymbol{h}_1 = \boldsymbol{h}_5$, the error could be in position 1 or 5 and we could not tell the difference. If we choose our error-check matrix $\boldsymbol{H}$ in such a way that the column vectors $\boldsymbol{h}_1, \boldsymbol{h}_2, \dots$ are *all different* (of course also from 0), then we can always correct an error, provided there is only one. Indeed, if the error was in position 3 then

$$\boldsymbol{H}\boldsymbol{y} \bmod 2 = \boldsymbol{h}_3.$$

Since all vectors $\boldsymbol{h}_1, \boldsymbol{h}_2, \dots$ are different, if we see the vector $\boldsymbol{h}_3$ we can imply that the bit y_3 is corrupted. This code is called the ***Hamming code***. For example, the following error check matrix defines the Hamming code of size 7:

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} = (\boldsymbol{h}_1, \dots, \boldsymbol{h}_7). \tag{6.28}$$

In general, if we have s error check bits then our code can have size $2^s - 1$, hence the number of bits left to store information, the ***information bits*** is $k = 2^s - s - 1$. So, to protect m bits of information from a single error, the Hamming code adds $\approx \log m$ error check bits. This is much better than repeating every bit 3 times.

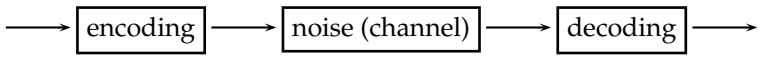


Figure 6.12. Transmission through a noisy channel.

Codes

Let us summarise the error-correction scenario in general terms. In order to fight noise, the sender ***encodes*** the ***message*** $\boldsymbol{x}$ by an ***encoding function*** ϕ_* into a longer string $\phi_*(\boldsymbol{x})$ which, for simplicity, we also assume to be binary. This ***codeword*** will be changed by noise into a string $\boldsymbol{y}$. The receiver gets $\boldsymbol{y}$ and applies to it a ***decoding function*** ϕ^*.

Definition 6.26 *The pair of functions* $\phi_* : \{0,1\}^m \to \{0,1\}^n$ *and* $\phi^* : \{0,1\}^n \to \{0,1\}^m$ *is called a* ***code*** *if* $\phi^*(\phi_*(\boldsymbol{x})) = \boldsymbol{x}$ *holds for all* $\boldsymbol{x} \in \{0,1\}^m$. *The strings* $\boldsymbol{x} \in \{0,1\}^m$ *are called* ***messages****, words of the form* $\boldsymbol{y} = \phi_*(\boldsymbol{x}) \in \{0,1\}^n$ *are called* ***codewords****. (Sometimes the set of all codewords by itself is also called a code.) For every message* $\boldsymbol{x}$*, the set of words* $C_{\boldsymbol{x}} = \{\, \boldsymbol{y} : \phi^*(\boldsymbol{y}) = \boldsymbol{x} \,\}$ *is called the* ***decoding set*** *of* $\boldsymbol{x}$*. (Of course, different decoding sets are disjoint.) The number*

$$R = m/n$$

is called the ***rate*** *of the code.*

We say that our code that ***corrects*** t ***errors*** *if for all possible messages* $\boldsymbol{x} \in \{0,1\}^m$*, if the received word* $\boldsymbol{y} \in \{0,1\}^n$ *differs from the codeword* $\phi_*(\boldsymbol{x})$ *in at most* t *positions, then* $\phi^*(\boldsymbol{y}) = \boldsymbol{x}$.

If the rate is R then the n-bit codewords carry Rn bits of useful information. In terms of decoding sets, a code corrects t errors if each decoding set $C_{\boldsymbol{x}}$ contains all words that differ from $\phi_*(\boldsymbol{x})$ in at most t symbols (the set of these words is a kind of "ball" of radius t).

The Hamming code corrects a single error, and its rate is close to 1. One of the important questions connected with error-correcting codes is how much do we have to lower the rate in order to correct more errors.

Having a notion of codes, we can formulate the main result of this section about information storage.

Theorem 6.27 (Network information storage). *There are constants* $\epsilon, c_1, c_2, R > 0$ *with the following property. For all sufficiently large* m*, there is a code* (ϕ_*, ϕ^*) *with message length* m *and codeword length* $n \le m/R$*, and a Boolean clocked circuit* $\mathcal{N}$ *of size* $O(n)$ *with* n *inputs and* n *outputs, such that the following holds. Suppose that at time 0, the memory cells of the circuit contain string* $\boldsymbol{Y}_0 = \phi_*(\boldsymbol{x})$*. Suppose further that the evolution* $\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_t$ *of the circuit has* ϵ*-admissible failures. Then we have*

$$\mathbf{P}[\, \phi^*(\boldsymbol{Y}_t) \neq \boldsymbol{x} \,] < t(c_1\epsilon)^{-c_2 n}.$$

This theorem shows that it is possible to store m bits information for time t, in a clocked circuit of size

$$O(\max(\lg t, m)).$$

As long as the storage time t is below the exponential bound e^{cm} for a certain constant c, this circuit size is only a constant times larger than the amount m of information it stores. (In contrast, in (6.27) we needed an extra factor $\lg m$ when we used a separate restoring organ for each bit.)

The theorem says nothing about how difficult it is to compute the codeword $\phi_*(\boldsymbol{x})$ at the beginning and how difficult it is to carry out the decoding $\phi^*(\boldsymbol{Y}_t)$ at the end. Moreover, it is desirable to perform these two operations also in a noise-tolerant fashion. We will return to the problem of decoding later.

Linear algebra

Since we will be dealing more with bit matrices, it is convenient to introduce the algebraic structure

$$\mathbb{F}_2 = (\{0,1\}, +, \cdot),$$

which is a two-element ***field***. Addition and multiplication in $\mathbb{F}_2$ are defined modulo 2 (of course, for multiplication this is no change). It is also convenient to vest the set $\{0,1\}^n$ of binary strings with the structure $\mathbb{F}_2^n$ of an n-dimensional vector space over the field $\mathbb{F}_2$. Most theorems and algorithms of basic linear algebra apply to arbitrary fields: in particular, one can define the row rank of a matrix as the maximum number of linearly independent rows, and similarly the column rank. Then it is a theorem that the row rank is equal to the column rank. From now on, in algebraic operations over bits or bit vectors, we will write + in place of $\oplus$ unless this leads to confusion. To save space, we will frequently write column vectors horizontally: we write

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = (x_1, \ldots, x_n)^T,$$

where $\boldsymbol{A}^T$ denotes the transpose of matrix $\boldsymbol{A}$. We will write

$$\boldsymbol{I}_r$$

for the identity matrix over the vector space $\mathbb{F}_2^r$.

Linear codes

Let us generalise the idea of the Hamming code.

Definition 6.28 *A code (ϕ_*, ϕ^*) with message length m and codeword length n is **linear** if, when viewing the message and code vectors as vectors over the field $\mathbb{F}_2$, the encoding function can be computed according to the formula*

$$\phi_*(\boldsymbol{x}) = \boldsymbol{G}\boldsymbol{x},$$

*with an $m \times n$ matrix $\boldsymbol{G}$ called the **generator matrix** of the code. The number m is called the the number of **information bits** in the code, the number*

$$k = n - m$$

*the number of **error-check bits**.*

Example 6.9 The matrix $\boldsymbol{H}$ in (6.28) can be written as $\boldsymbol{H} = (\boldsymbol{K}, \boldsymbol{I}_3)$, with

$$\boldsymbol{K} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

Then the error check relation can be written as

$$\boldsymbol{y} = \begin{pmatrix} \boldsymbol{I}_4 \\ -\boldsymbol{K} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_4. \end{pmatrix}.$$

This shows that the bits $y_1, \ldots, y_4$ can be taken to be the message bits, or "information bits", of the code, making the Hamming code a linear code with the generator matrix $(\boldsymbol{I}_4, -\boldsymbol{K})^T$. (Of course, $-\boldsymbol{K} = \boldsymbol{K}$ over the field $\mathbb{F}_2$.)

The following statement is proved using standard linear algebra, and it generalises the relation between error check matrix and generator matrix seen in Example 6.9..

Claim 6.29 *Let $k, m > 0$ be given with $n = m + k$.*

1. *For every $n \times m$ matrix $\boldsymbol{G}$ of rank m over $\mathbb{F}_2$ there is a $k \times n$ matrix $\boldsymbol{H}$ of rank k with the property*
$$\{\, \boldsymbol{Gx} : \boldsymbol{x} \in \mathbb{F}_2^m \,\} = \{\, \boldsymbol{y} \in \mathbb{F}_2^n : \boldsymbol{Hy} = 0 \,\}. \tag{6.29}$$
2. *For every $k \times n$ matrix $\boldsymbol{H}$ of rank k over $\mathbb{F}_2$ there is an $n \times m$ matrix $\boldsymbol{G}$ of rank m with property (6.29).*

Definition 6.30 *For a vector $\boldsymbol{x}$, let $|\boldsymbol{x}|$ denote the number of its nonzero elements: we will also call it the **weight** of $\boldsymbol{x}$.*

In what follows it will be convenient to define a code starting from an error-check matrix $\boldsymbol{H}$. If the matrix has rank k then the code has rate

$$R = 1 - k/n.$$

We can fix any subset S of k linearly independent columns, and call the indices $i \in S$ ***error check bits*** and the indices $i \notin S$ the ***information bits***. (In Example 6.9., we chose $S = \{5, 6, 7\}$.) Important operations can performed over a code, however, without fixing any separation into error-check bits and information bits.

6.5.4. Refreshers

Correcting a single error was not too difficult; finding a similar scheme to correct 2 errors is much harder. However, in storing n bits, typically ϵn (much more than 2) of those bits will be corrupted in every step. There are ingenious and quite efficient codes of positive rate (independent of n) correcting even this many errors. When applied to information storage, however, the error-correction mechanism itself must also work in noise, so we are looking for a particularly simple one. It works in our favour, however, that not all errors need to be corrected: it is sufficient to cut down their number, similarly to the restoring organ in reliable

Boolean circuits above.

For simplicity, as gates of our circuit we will allow certain Boolean functions with a large, but constant, number of arguments. On the other hand, our Boolean circuit will have just depth 1, similarly to a restoring organ of Section 6.4. The output of each gate is the input of a memory cell (shift register). For simplicity, we identify the gate and the memory cell and call it a ***cell***. At each clock tick, a cell reads its inputs from other cells, computes a Boolean function on them, and stores the result (till the next clock tick). But now, instead of majority vote among the input values cells, the Boolean function computed by each cell will be slightly more complicated.

Our particular restoring operations will be defined, with the help of a certain $k \times n$ parity check matrix $\boldsymbol{H} = (h_{ij})$. Let $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ be a vector of bits. For some $j = 1, \ldots, n$, let V_j (from "vertical") be the set of those indices i with $h_{ij} = 1$. For integer $i = 1, \ldots, k$, let H_i (from "horizontal") be the set of those indices j with $h_{ij} = 1$. Then the condition $\boldsymbol{Hx} = 0$ can also be expressed by saying that for all i, we have $\sum_{j \in H_i} x_j \equiv 0 \pmod 2$. The sets H_i are called the ***parity check sets*** belonging to the matrix $\boldsymbol{H}$. From now on, the indices i will be called ***checks***, and the indices j ***locations***.

Definition 6.31 *A linear code $\boldsymbol{H}$ is a **low-density parity-check code** with bounds $K, N > 0$ if the following conditions are satisfied:*

1. *For each j we have $|V_j| \leq K$;*
2. *For each i we have $|H_i| \leq N$.*

In other words, the weight of each row is at most N and the weight of each column is at most K.

In our constructions, we will keep the bounds K, N constant while the length n of codewords grows. Consider a situation when $\boldsymbol{x}$ is a codeword corrupted by some errors. To check whether bit x_j is incorrect we may check all the sums

$$s_i = \sum_{j \in H_i} x_j$$

for all $i \in V_j$. If all these sums are 0 then we would not suspect x_j to be in error. If only one of these is nonzero, we will know that $\boldsymbol{x}$ has some errors but we may still think that the error is not in bit x_j. But if a significant number of these sums is nonzero then we may suspect that x_j is a culprit and may want to change it. This idea suggests the following definition.

Definition 6.32 *For a low-density parity-check code $\boldsymbol{H}$ with bounds K, N, the **refreshing operation** associated with the code is the following, to be performed simultaneously for all locations j:*

> *Find out whether more than $\lfloor K/2 \rfloor$ of the sums s_i are nonzero among the ones for $i \in V_j$. If this is the case, flip x_j.*

*Let $\boldsymbol{x}^{\boldsymbol{H}}$ denote the vector obtained from $\boldsymbol{x}$ by this operation. For parameters $0 < \vartheta, \gamma < 1$, let us call $\boldsymbol{H}$ a $(\vartheta, \gamma, K, N, k, n)$-**refresher** if for each vector $\boldsymbol{x}$ of length n with weight $|\boldsymbol{x}| \leq \vartheta n$ the weight of the resulting vector decreases thus: $|\boldsymbol{x}^{\boldsymbol{H}}| \leq \gamma \vartheta n$.*

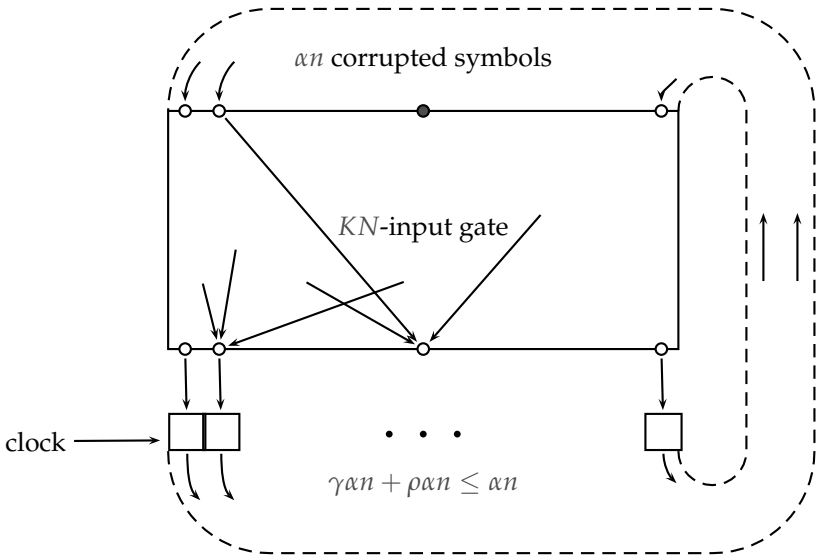


Figure 6.13. Using a refresher.

Notice the similarity of refreshers to compressors. The following lemma shows the use of refreshers, and is an example of the advantages of linear codes.

Lemma 6.33 *For an $(\vartheta, \gamma, K, N, k, n)$-refresher $\boldsymbol{H}$, let $\boldsymbol{x}$ be an n-vector and $\boldsymbol{y}$ a codeword of length n with $|\boldsymbol{x} - \boldsymbol{y}| \le \vartheta n$. Then $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| \le \gamma\vartheta n$.*

Proof. Since $\boldsymbol{y}$ is a codeword, $\boldsymbol{H}\boldsymbol{y} = 0$, implying $\boldsymbol{H}(\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{H}\boldsymbol{x}$. Therefore the error correction flips the same bits in $\boldsymbol{x} - \boldsymbol{y}$ as in $\boldsymbol{x}$: $(\boldsymbol{x} - \boldsymbol{y})^{\boldsymbol{H}} - (\boldsymbol{x} - \boldsymbol{y}) = \boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{x}$, giving $\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y} = (\boldsymbol{x} - \boldsymbol{y})^{\boldsymbol{H}}$. So, if $|\boldsymbol{x} - \boldsymbol{y}| \le \vartheta n$, then $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| = |(\boldsymbol{x} - \boldsymbol{y})^{\boldsymbol{H}}| \le \gamma\vartheta n$. ■

Theorem 6.34 *There is a parameter $\vartheta > 0$ and integers $K > N > 0$ such that for all sufficiently large code length n and $k = Nn/K$ there is a $(\vartheta, 1/2, K, N, k, n)$-refresher with at least $n - k = 1 - N/K$ information bits.*

In particular, we can choose $N = 100$, $K = 120$, $\vartheta = 1.31 \cdot 10^{-4}$.

We postpone the proof of this theorem, and apply it first.

Proof. of Theorem 6.27 Theorem 6.34 provides us with a device for information storage. Indeed, we can implement the operation $\boldsymbol{x} \to \boldsymbol{x}^{\boldsymbol{H}}$ using a single gate g_j of at most KN inputs for each bit j of $\boldsymbol{x}$. Now as long as the inequality $|\boldsymbol{x} - \boldsymbol{y}| \le \vartheta n$ holds for some codeword $\boldsymbol{y}$, the inequality $|\boldsymbol{x}^{\boldsymbol{H}} - \boldsymbol{y}| \le \gamma\vartheta n$ follows with $\gamma = 1/2$. Of course, some gates will fail and introduce new deviations resulting in some $\boldsymbol{x}'$ rather than $\boldsymbol{x}^{\boldsymbol{H}}$. Let $e\epsilon < \vartheta/2$ and $\rho = 1 - \gamma (= 1/2)$. Then just as earlier, the probability that there are more than $\rho\vartheta n$ failures is bounded by the exponentially decreasing expression $(e\epsilon/\rho\vartheta)^{\rho\vartheta n}$. With fewer than $\rho\vartheta n$ new deviations, we will still have $|\boldsymbol{x}' - \boldsymbol{y}| < (\gamma + \rho)\vartheta n < \vartheta n$. The probability that at any time $\le t$ the number of

failures is more than $\rho\vartheta n$ is bounded by

$$t(e\epsilon/\rho\vartheta)^{\rho\vartheta n} < t(6\epsilon/\vartheta)^{(1/2)\vartheta n}.$$

■

Example 6.10 Let $\epsilon = 10^{-9}$. Using the sample values in Theorem 6.34 we can take $N = 100$, $K = 120$, so the information rate is $1 - N/K = 1/6$. With the corresponding values of ϑ, and $\gamma = \rho = 1/2$, we have $\rho\vartheta = 6.57 \cdot 10^{-5}$. The probability that there are more than $\rho\vartheta n$ failures is bounded by

$$(e\epsilon/\rho\vartheta)^{\rho\vartheta n} = (10^{-4}e/6.57)^{6.57\cdot 10^{-5}n} \approx e^{-6.63\cdot 10^{-4}n}.$$

This is exponentially decreasing with n, albeit initially very slowly: it is not really small until $n = 10^4$. Still, for $n = 10^6$, it gives $e^{-663} \approx 1.16 \cdot 10^{-288}$.

Decoding?

In order to use a refresher for information storage, first we need to enter the encoded information into it, and at the end, we need to decode the information from it. How can this be done in a noisy environment? We have nothing particularly smart to say here about encoding besides the reference to the general reliable computation scheme discussed earlier. On the other hand, it turns out that if ϵ is sufficiently small then *decoding can be avoided* altogether.

Recall that in our codes, it is possible to designate certain symbols as information symbols. So, in principle it is sufficient to read out these symbols. The question is only how likely it is that any one of these symbols will be corrupted. The following theorem upper-bounds the probability for any symbol to be corrupted, at any time.

Theorem 6.35 *For parameters $\vartheta, \gamma > 0$, integers $K > N > 0$, code length n, with $k = Nn/K$, consider a $(\vartheta, 1/2, K, N, k, n)$-refresher. Build a Boolean clocked circuit $\mathcal{N}$ of size $O(n)$ with n inputs and n outputs based on this refresher, just as in the proof of Theorem 6.27. Suppose that at time 0, the memory cells of the circuit contain string $\boldsymbol{Y}_0 = \phi_*(\boldsymbol{x})$. Suppose further that the evolution $\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_t$ of the circuit has ϵ-admissible failures. Let $\boldsymbol{Y}_t = (Y_t(1), \ldots, Y_t(n))$ be the bits stored at time t. Then $\epsilon < (2.1KN)^{-10}$ implies*

$$\mathbf{P}[\, Y_t(j) \neq Y_0(j) \,] \leq c\epsilon + t(6\epsilon/\vartheta)^{(1/2)\vartheta n}$$

for some c depending on N, K.

Remark 6.36 *What we are bounding is only the probability of a corrupt symbol in the particular position j. Some of the symbols will certainly be corrupt, but any one symbol one points to will be corrupt only with probability $\leq c\epsilon$.*

The upper bound on ϵ required in the condition of the theorem is very severe, underscoring the theoretical character of this result.

Proof. As usual, it is sufficient to assume $\boldsymbol{Y}_0 = 0$. Let $D_t = \{\, j : Y_t(j) = 1 \,\}$, and let E_t be the set of circuit elements j which fail at time t. Let us define the following sequence of integers:

$$b_0 = 1, \quad b_{u+1} = \lceil (4/3)b_u \rceil, \quad c_u = \lceil (1/3)b_u \rceil.$$

It is easy to see from here by induction

$$b_0 + \cdots + b_{u-1} \le 3b_u \le 9c_u. \tag{6.30}$$

The first members of the sequence b_u are 1,2,3,4,6,8,11,15,18,24,32, and for c_u they are 1,1,1,2,2,3,4,5,6,8,11.

Claim 6.37 *Suppose that $Y_t(j_0) \neq 0$. Then either there is a time $t' < t$ at which $\ge (1/2)\vartheta n$ circuit elements failed, or there is a sequence of sets $B_u \subseteq D_{t-u}$ for $0 \le u < v$ and $C \subseteq E_{t-v}$ with the following properties.*

1. *For $u > 0$, every element of B_u shares some error-check with some element of B_{u-1}. Also every element of C shares some error-check with some element of B_{v-1}.*
2. *We have $|E_{t-u} \cap B_u| < |B_u|/3$ for $u < v$, on the other hand $C \subseteq E_{t-v}$.*
3. *We have $B_0 = \{j_0\}$, $|B_u| = b_u$, for all $u < v$, and $|C| = c_v$.*

Proof. We will define the sequence B_u recursively, and will say when to stop. If $j_0 \in E_t$ then we set $v = 0$, $C = \{0\}$, and stop. Suppose that B_u is already defined. Let us define B_{u+1} (or C if $v = u + 1$). Let B'_{u+1} be the set of those j which share some error-check with an element of B_u, and let $B''_{u+1} = B'_{u+1} \cap D_{t-u-1}$. The refresher property implies that either $|B''_{u+1}| > \vartheta n$ or

$$|B_u \setminus E_{t-u}| \le (1/2)|B''_{u+1}|.$$

In the former case, there must have been some time $t' < t-u$ with $|E_{t'}| > (1/2)\vartheta n$, otherwise D_{t-u-1} could never become larger than ϑn. In the latter case, the property $|E_{t-u} \cap B_u| < (1/3)|B_u|$ implies

$$\begin{aligned}(2/3)|B_u| < |B_u \setminus E_{t-u}| &\le (1/2)|B''_{u+1}|,\\ (4/3)b_u < |B''_{u+1}|.&\end{aligned}$$

Now if $|E_{t-u-1} \cap B''_{u+1}| < (1/3)|B''_{u+1}|$ then let B_{u+1} be any subset of B''_{u+1} with size b_{u+1} (there is one), else let $v = u + 1$ and $C \subseteq E_{t-u-1} \cap B''_{u+1}$ a set of size c_v (there is one). This construction has the required properties. ∎

For a given B_u, the number of different choices for B_{u+1} is bounded by

$$\binom{|B'_{u+1}|}{b_{u+1}} \le \binom{KNb_u}{b_{u+1}} \le \left(\frac{eKNb_u}{b_{u+1}}\right)^{b_{u+1}} \le ((3/4)eKN)^{b_{u+1}} \le (2.1KN)^{b_{u+1}},$$

where we used (6.10). Similarly, the number of different choices for C is bounded by

$$\binom{KNb_{v-1}}{c_v} \le \mu^{c_v} \text{ with } \mu = 2.1KN.$$

It follows that the number of choices for the whole sequence $B_1, \ldots, B_{v-1}, C$ is bounded by

$$\mu^{b_1 + \cdots + b_{v-1} + c_v}.$$

On the other hand, the probability for a fixed C to have $C \subseteq E_v$ is $\le \epsilon^{c_v}$. This way, we can

bound the probability that the sequence ends exactly at v by

$$p_v \le \epsilon^{c_v}\mu^{b_1+\cdots+b_{v-1}+c_v} \le \epsilon^{c_v}\mu^{10c_v},$$

where we used (6.30). For small v this gives

$$p_0 \le \epsilon, \quad p_1 \le \epsilon\mu, \quad p_2 \le \epsilon\mu^3, \quad p_3 \le \epsilon^2\mu^6, \quad p_4 \le \epsilon^2\mu^{10}, \quad p_5 \le \epsilon^3\mu^{16}.$$

Therefore

$$\sum_{v=0}^{\infty} p_v \le \sum_{v=0}^{5} p_v + \sum_{v=6}^{\infty}(\mu^{10}\epsilon)^{c_v} \le \epsilon(1+\mu+\mu^3) + \epsilon^2(\mu^6+\mu^{10}) + \frac{\epsilon^3\mu^{16}}{1-\epsilon\mu^{10}},$$

where we used $\epsilon\mu^{10} < 1$ and the property $c_{v+1} > c_v$ for $v \ge 5$. We can bound the last expression by $c\epsilon$ with an appropriate constant c.

We found that the event $Y_t(j) \ne Y_0(j)$ happens either if there is a time $t' < t$ at which $\ge (1/2)\vartheta n$ circuit elements failed (this has probability bound $t(2e\epsilon/\vartheta)^{(1/2)\vartheta n}$) or an event of probability $\le c\epsilon$ occurs. ■

Expanders

We will construct our refreshers from bipartite multigraphs with a property similar to compressors: expanders (see Exercise 6.4-3.).

Definition 6.38 *Here, we will distinguish the two parts of the bipartite (multi) graphs not as inputs and outputs but as* ***left nodes*** *and* ***right nodes****. A bipartite multigraph B is (N,K)-* ***regular*** *if the points of the left set have degree N and the points in the right set have degree K. Consider such a graph, with the left set having n nodes (then the right set has nN/K nodes). For a subset E of the left set of B, let* $\mathrm{Nb}(E)$ *consist of the points connected by some edge to some element of E. We say that the graph B* ***expands*** *E by a factor λ if we have $|\mathrm{Nb}(E)| \ge \lambda|E|$. For $\alpha, \lambda > 0$, our graph B is an (N,K,α,λ,n)-expander if B expands every subset E of size $\le \alpha n$ of the left set by a factor λ.*

Definition 6.39 *Given an (N,K)-regular bipartite multigraph B, with left set $\{u_1,\ldots,u_n\}$ and right set $\{v_1,\ldots,v_k\}$, we assign to it a low-density parity-check code $\boldsymbol{H}(B)$ as follows: $h_{ij} = 1$ if v_i is connected to u_j, and 0 otherwise.*

We will create our low-density parity-check code $\boldsymbol{H}(B)$ with the help of an expander graph B. Now for every possible error set E, the set $\mathrm{Nb}(E)$ describes the set of parity check that the elements of E participate in. Under some conditions, the lower bound on the size of $\mathrm{Nb}(E)$ guarantees that a sufficient number of errors will be corrected.

Theorem 6.40 *Let B be an $(N,K,\alpha,(7/8)N,n)$-expander with integer αn. Let $k = Nn/K$. Then $\boldsymbol{H}(B)$ is a $((3/4)\alpha, 1/2, K, N, k, n)$-refresher.*

Proof. More generally, for any $\epsilon > 0$, let B be an $(N,K,\alpha,(3/4+\epsilon)N,n)$-expander with integer αn. We will prove that $\boldsymbol{H}(B)$ is a $(\alpha(1+4\epsilon)/2, (1-4\epsilon), K, N, k, n)$-refresher. For an n-dimensional bit vector $\boldsymbol{x}$ with $A = \{\, j : x_j = 1 \,\}$, $a = |A| = |\boldsymbol{x}|$, assume

$$a \le n\alpha(1+4\epsilon)/2. \tag{6.31}$$

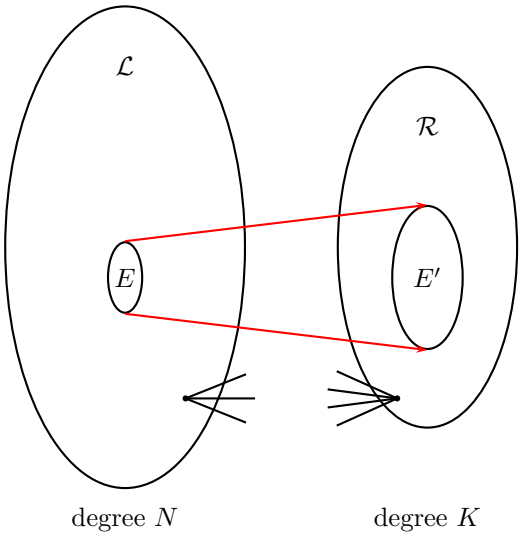


Figure 6.14. A regular expander.

Our goal is to show $|\boldsymbol{x}^{\boldsymbol{H}}| \leq a(1-4\epsilon)$: in other words, that in the corrected vector the number of 1's decreases at least by a factor of $(1-4\epsilon)$.

Let F be the set of bits in A that the error correction operation fails to flip, with $f = |F|$, and G the set of of bits that were 0 but the operation turns them to 1, with $g = |G|$. Our goal is to bound $|F \cup G| = f + g$. The key observation is that each element of G shares at least half of its neighbours with elements of A, and similarly, each element of F shares at least half of its neighbours with other elements of A. Therefore both F and G contribute relatively weakly to the expansion of $A \cup G$. Since this expansion is assumed strong, the size of $|F \cup G|$ must be limited.

Let

$$\delta = |\mathrm{Nb}(A)|/(Na).$$

By expansion, $\delta \geq 3/4 + \epsilon$.

First we show $|A \cup G| \leq \alpha n$. Assume namely that, on the contrary, $|A \cup G| > \alpha n$, and let G' be a subset of G such that $|A \cup G'| = \alpha n =: p$ (an integer, according to the assumptions of the theorem). By expansion,

$$(3/4 + \epsilon)Np \leq \mathrm{Nb}(A \cup G').$$

Each bit in G has at most $N/2$ neighbours that are not neighbours of A; so,

$$|\mathrm{Nb}(A \cup G')| \leq \delta Na + N(p-a)/2.$$

Combining these:

$$\begin{aligned} \delta a + (p-a)/2 &\geq (3/4 + \epsilon)p, \\ a &\geq p(1+4\epsilon)/(4\delta - 2) \geq \alpha n(1+4\epsilon)/2, \end{aligned}$$

since $\delta \leq 1$. This contradiction with (6.31) shows $|A \cup G| \leq \alpha n$.

Now $|A \cup G| \leq \alpha n$ implies (recalling that each element of G contributes at most $N/2$ new neighbours):

$$\begin{aligned}(3/4+\epsilon)N(a+g) \leq |\mathrm{Nb}(A \cup G)| \leq \delta N a + (N/2)g,\\ (3/4+\epsilon)(a+g) \leq \delta a + g/2,\\ (3/4+\epsilon)a + (1/4+\epsilon)g \leq \delta a. \end{aligned} \tag{6.32}$$

Each $j \in F$ must share at least half of its neighbours with others in A. Therefore j contributes at most $N/2$ neighbours on its own; the contribution of the other $N/2$ must be divided by 2, so the the total contribution of j to the neighbours of A is at most $(3/4)N$:

$$\begin{aligned}\delta N a = \mathrm{Nb}(A) \leq N(a-f) + (3/4)Nf = N(a - f/4),\\ \delta a \leq a - f/4.\end{aligned}$$

Combining with (6.32):

$$\begin{aligned}(3/4+\epsilon)a + (1/4+\epsilon)g \leq a - f/4,\\ (1-4\epsilon)a \geq f + (1+4\epsilon)g \geq f + g.\end{aligned}$$

■

Random expanders

Are there expanders good enough for Theorem 6.40? The maximum expansion factor is the degree N and we require a factor of $(7/8)N$. It turns out that random choice works here, too, similarly to the one used in the construction of compressors.

The choice has to be done in a way that the result is an (N, K)-regular bipartite multigraph of left size n. We will start with Nn left nodes $u_1, \ldots, u_{Nn}$ and Nn right nodes $v_1, \ldots, v_{Nn}$. Now we choose a random ***matching***, that is a set of Nn edges with the property that every left node is connected by an edge to exactly one right node. Let us call the resulting graph M. We obtain B now as follows: we collapse each group of N left nodes into a single node: $u_1, \ldots, u_N$ into one node, $u_{N+1}, \ldots, u_{2N}$ into another node, and so on. Similarly, we collapse each group of K right nodes into a single node: $v_1, \ldots, v_K$ into one node, $v_{K+1}, \ldots, v_{2K}$ into another node, and so on. The edges between any pair of nodes in B are inherited from the ancestors of these nodes in M. This results in a graph B with n left nodes of degree N and nN/K right nodes of degree K. The process may give multiple edges between nodes of B, this is why B is called a multigraph. Two nodes of M will be called ***cluster neighbours*** if they are collapsed to the same node of B.

Theorem 6.41 *Suppose*

$$0 < \alpha \leq e^{\frac{-1}{N/8-1}} \cdot (22K)^{\frac{-1}{1-8/N}}.$$

Then the above random choice gives an $(N, K, \alpha, (7/8)N, n)$-expander with positive probability.

Example 6.11 If $N = 48$, $K = 60$ then the inequality in the condition of the theorem becomes

$$\alpha \leq 1/6785.$$

Proof. Let E be a set of size αn in the left set of B. We will estimate the probability that E has too few neighbours. In the above choice of the graph B we might as well start with assigning edges to the nodes of E, in some fixed order of the $N|E|$ nodes of the preimage of E in M. There are $N|E|$ edges to assign. Let us call a node of the right set of M ***occupied*** if it has a cluster neighbour already reached by an earlier edge. Let X_i be a random variable that is 1 if the ith edge goes to an occupied node and 0 otherwise. There are

$$Nn - i + 1 \geq Nn - N\alpha n = Nn(1-\alpha)$$

choices for the ith edge and at most $KN|E|$ of these are occupied. Therefore

$$\mathbf{P}[\, X_i = 1 \mid X_1, \ldots, X_{i-1} \,] \leq \frac{KN|E|}{Nn(1-\alpha)} = \frac{K\alpha}{1-\alpha} =: p.$$

Using the large deviations theorem in the generalisation given in Exercise 6.1-3., we have, for $f > 0$:

$$\mathbf{P}[\, \sum_{i=1}^{N\alpha n} X_i \geq fN\alpha n \,] \leq e^{-N\alpha n D(f,p)} \leq \left(\frac{ep}{f}\right)^{fN\alpha n}.$$

Now, the number of different neighbours of E is $N\alpha n - \sum_i X_i$, hence

$$\mathbf{P}[\, N(E) \leq N\alpha n(1-f) \,] \leq \left(\frac{ep}{f}\right)^{fN\alpha n} = \left(\frac{eK\alpha}{f(1-\alpha)}\right)^{fN\alpha n}.$$

Let us now multiply this with the number

$$\sum_{i \leq \alpha n} \binom{n}{\alpha n} \leq (e/\alpha)^{\alpha n}$$

of sets E of size $\leq \alpha n$:

$$\left(\frac{e}{\alpha}\right)^{\alpha n}\left(\frac{eK\alpha}{f(1-\alpha)}\right)^{fN\alpha n} = \left(\alpha^{fN-1} e \left(\frac{eK}{f(1-\alpha)}\right)^{fN}\right)^{\alpha n} \leq \left(\alpha^{fN-1} e \left(\frac{eK}{0.99f}\right)^{fN}\right)^{\alpha n},$$

where in the last step we assumed $\alpha \leq 0.01$. This is < 1 if

$$\alpha \leq e^{\frac{-1}{fN-1}} \left(\frac{eK}{0.99f}\right)^{\frac{-1}{1-1/(fN)}}.$$

Substituting $f = 1/8$ gives the formula of the theorem. ■

Proof. of Theorem 6.34 Theorem 6.40 shows how to get a refresher from an expander, and Theorem 6.41 shows the existence of expanders. Example 6.11. shows that expander with the needed sample parameters exists. ■

Exercises

6.5-1 Prove Proposition 6.29.

6.5-2 Apply the ideas of the proof of Theorem 6.35 to the proof of Theorem 6.16, showing that the "coda" circuit is not needed: each wire of the output cable carries the correct value with high probability.

Chapter notes

The large deviation theorem (Theorem 6.1), or theorems similar to it, are sometimes attributed to Chernoff or Bernstein. One of its frequently used variants is given in Exercise 6.1-2..

The problem of reliable computation with unreliable components was addressed by John von Neumann in [46] on the model of logic circuits. A complete proof of the result of that paper (with a different restoring organ) appear first in the paper [11] of R. L. Dobrushin and S. I. Ortyukov. Our presentation relied on parts of the paper [50] of N. Pippenger.

The lower-bound result of Dobrushin and Ortyukov in the paper [10] (corrected in [48], [51] and [19]), shows that redundancy of $\log n$ is unavoidable for a general reliable computation whose complexity is n. However, this lower bound only shows the necessity of putting the input into a redundantly encoded form (otherwise critical information may be lost in the first step). As shown in [50], for many important function classes, linear redundancy is achievable.

It seems natural to separate the cost of the initial encoding: it might be possible to perform the rest of the computation with much less redundancy. An important step in this direction has been made by D. Spielman in the paper [63] in (essentially) the clocked-circuit model. Spielman takes a parallel computation with time t running on w elementary components and makes it reliable using only $(\log w)^c$ times more processors and running it $(\log w)^c$ times longer. The failure probability will be $t\exp(-w^{1/4})$. This is small as long as t is not much larger than $\exp(w^{1/4})$. So, the redundancy is bounded by some power of the logarithm of the *space requirement*; the time requirement does not enter explicitly. In Boolean circuits no time- and space- complexity is defined separately. The size of the circuit is analogous to the quantity obtained in other models by taking the product of space and time complexity.

Questions more complex than Problem *6-1.* have been studied in [49]. The method of Problem *6-2.*, for generating random d-regular multigraphs is analysed for example in [6]. It is much harder to generate simple regular graphs (not multigraphs) uniformly. See for example [29].

The result of Exercise 6.2-4. is due to C. Shannon, see [56]. The asymptotically best circuit size for the worst functions was found by Lupanov in [40]. Exercise 6.3-1. is based on [11], and Exercise 6.3-2. is based on [10] (and its corrections).

Problem *6-7.* is based on the starting idea of the $\lg n$ depth sorting networks in [4].

For storage in Boolean circuits we partly relied on A. V. Kuznietsov's paper [33] (the main theorem, on the existence of refreshers is from M. Pinsker). Low density parity check codes were introduced by R. G. Gallager in the book [15], and their use in reliable storage was first suggested by M. G. Taylor in the paper [67]. New, constructive versions of these codes were developed by M. Sipser and D. Spielman in the paper [64], with superfast coding and decoding.

Expanders, invented by Pinsker in [47] and introduced here in Exercise 6.4-3. have been used extensively in theoretical computer science: see for example [45] for some more detail. This book also gives references on the construction of graphs with large eigenvalue-gap. Exercise 6.4-5. and Problem *6-6.* are based on [11].

The use of expanders in the role of refreshers was suggested by Pippenger (private communication): our exposition follows Sipser and Spielman in [61]. Random expanders were found for example by Pinsker. The needed expansion rate ($> 3/4$ times the left degree)

is larger than what can be implied from the size of the eigenvalue gap. As shown in [47] (see the proof in Theorem 6.41) random expanders have the needed expansion rate. Lately, constructive expanders with nearly maximal expansion rate were announced by Capalbo, Reingold, Vadhan and Wigderson in [8].

Reliable computation is also possible in a model of parallel computation that is much more regular than logic circuits: in cellular automata. We cannot present those results here: see for example the papers [20] and [18].

Problems

6-1. Critical value

Consider a circuit $\mathcal{M}_k$ like in Exercise 6.2-5., assuming that each gate fails with probability $\leq \epsilon$ independently of all the others and of the input. Assume that the input vector is all 0, and let $p_k(\epsilon)$ be the probability that the circuit outputs a 1. Show that there is a value $\epsilon_0 < 1/2$ with the property that for all $\epsilon < \epsilon_0$ we have $\lim_{k\to\infty} p_k(\epsilon) = 0$, and for $\epsilon_0 < \epsilon \leq 1/2$, we have have $\lim_{k\to\infty} p_k(\epsilon) = 1/2$. Estimate also the speed of convergence in both cases.

6-2. Regular compressor

We defined a compressor as a d-halfregular bipartite multigraph. Let us call a compressor ***regular*** if it is a d-regular multigraph (the input nodes also have degree d). Prove a theorem similar to Theorem 6.20: for each $\gamma < 1$ there is an integer $d > 1$ and an $\alpha > 0$ such that for all integer $k > 0$ there is a regular (d, α, γ, k)-compressor. *Hint*: Choose a random d-regular bipartite multigraph by the following process: (1. Replace each vertex by a group of d vertices. 2. Choose a random complete matching between the new input and output vertices. 3. Merge each group of d vertices into one vertex again.) Prove that the probability, over this choice, that a d-regular multigraph is a not a compressor is small. For this, express the probability with the help of factorials and estimate the factorials using Stirling's formula.

6-3. Two-way expander

Recall the definition of expanders from Exercise 6.4-3.. Call a (d, α, λ, k)-expander ***regular*** if it is a d-regular multigraph (the input nodes also have degree d). We will call this multigraph a ***two-way expander*** if it is an expander in both directions: from A to B and from B to A. Prove a theorem similar to the one in Problem 6-2.: for all $\lambda < d$ there is an $\alpha > 0$ such that for all integers $k > 0$ there is a two-way regular (d, α, λ, k)-expander.

6-4. Restoring organ from 3-way voting

The proof of Theorem 6.20 did not guarantee a (d, α, γ, k)-compressor with any $\gamma < 1/2$, $d < 7$. If we only want to use 3-way majority gates, consider the following construction. First create a 3-halfregular bipartite graph G with inputs $u_1, \ldots, u_k$ and outputs $v_1, \ldots, v_{3k}$, with a 3-input majority gate in each v_i. Then create new nodes $w_1, \ldots, w_k$, with a 3-input majority gate in each w_j. The gate of w_1 computes the majority of v_1, v_2, v_3, the gate of w_2 computes the majority of v_4, v_5, v_6, and so on. Calculate whether a random choice of the graph G will turn the circuit with inputs $(u_1, \ldots, u_k)$ and outputs $(w_1, \ldots, w_k)$ into a restoring organ. Then consider three stages instead of two, where G has $9k$ outputs and see what is gained.

6-5. Restoring organ from NOR gates

The majority gate is not the only gate capable of strengthening the majority. Recall the

NOR gate introduced in Exercise 6.2-2., and form $\mathrm{NOR}_2(x_1, x_2, x_3, x_4) = (x_1 \,\mathrm{NOR}\, x_2)\,\mathrm{NOR}\,(x_3 \,\mathrm{NOR}\, x_4)$. Show that a construction similar to Problem 6-4. can be carried out with NOR_2 used in place of 3-way majority gates.

6-6. More randomness, smaller restoring organs

Taking the notation of Exercise 6.4-4., suppose like there, that the random variables F_v are independent of each other, and their distribution does not depend on the Boolean input vector. Apply the idea of Exercise 6.4-6. to the construction of each restoring organ. Namely, construct a different restoring organ for each position: the choice depends on the circuit preceding this position. Show that in this case, our error estimates can be significantly improved. The improvement comes, just as in Exercise 6.4-6., since now we do not have to multiply the error probability by the number of all possible sets of size $\leq \alpha k$ of tainted wires. Since we know the distribution of this set, we can average over it.

6-7. Near-sorting with expanders

In this problem, we show that expanders can be used for "near-sorting". Let G be a regular two-way (d, α, λ, k)-expander, whose two parts of size k are A and B. According to a theorem of Kőnig, (the edge-set of) every d-regular bipartite multigraph is the disjoint union of (the edge-sets of) d complete matchings $M_1, \ldots, M_d$. To such an expander, we assign a Boolean circuit of depth d as follows. The circuit's nodes are subdivide into levels $i = 0, 1, \ldots, d$. On level i we have two disjoint sets A_i, B_i of size k of nodes a_{ij}, b_{ij} $(j = 1, \ldots, k)$. The Boolean value on a_{ij}, b_{ij} will be x_{ij} and y_{ij} respectively. Denote the vector of $2k$ values at stage i by $\boldsymbol{z}_i = (x_{i1}, \ldots, y_{ik})$. If (p, q) is an edge in the matching M_i, then we put an $\wedge$ gate into a_{ip}, and a $\vee$ gate into b_{iq}:

$$x_{ip} = x_{(i-1)p} \wedge y_{(i-1)q}, \qquad y_{iq} = x_{(i-1)p} \vee y_{(i-1)q}.$$

This network is trying to "sort" the 0's to A_i and the 1's to B_i in d stages. More generally, the values in the vectors $\boldsymbol{z}_i$ could be arbitrary numbers. Then if $x \wedge y$ still means $\min(x, y)$ and $x \vee y$ means $\max(x, y)$ then each vector $\boldsymbol{z}_i$ is a permutation of the vector $\boldsymbol{z}_0$. Let $\beta = (1 + \lambda)\alpha$. Prove that $\boldsymbol{z}_d$ is ***β-sorted*** in the sense that for all m, at least βm among the m smallest values of $\boldsymbol{z}_d$ is in its left half and at least βm among the m largest values are in its right half.

6-8. Restoring organ from near-sorters

Develop a new restoring organ using expanders, as follows. First, split each wire of the input cable A, to get two sets A'_0, B'_0. Attach the β-sorter of Problem 6-7., getting outputs A'_d, B'_d. Now split the wires of B'_d into two sets A''_0, B''_0. Attach the β-sorter again, getting outputs A''_d, B''_d. Keep only $B = A''_d$ for the output cable. Show that the Boolean vector circuit leading from A to B can be used as a restoring organ.

Bibliography

[1] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling Vol. I.* Prentice-Hall, 1972. 123

[2] A. V. Aho, J. D. Ullman. *The Theory of Parsing, Translation and Compiling Vol. II.* Prentice-Hall, 1973. 123

[3] M. Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, 1998, pp. 10–19. 59, 60

[4] M. Ajtai, J. Komlós, E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983. 191

[5] A. Asteroth, C. Christel. *Theoretische Informatik*. Pearson Studium, 2002. 124

[6] E. Bender, R. Canfield. The asymptotic number of labeled graphs with given degree sequences. *Combinatorial Theory Series A*, 24:296–307, 1978. 191

[7] J. G. Brookshear. *Theory of Computation – Formal Langauges, Automata, and Complexity*. The Benjamin/Cummings Publishing Company, 1989. 124

[8] M. Capalbo, O. Reingold, S. Vadhan, A. Widgerson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 443–452, 2001. IEEE Computer Society. 192

[9] J. Carroll, D. Long. *Theory of Finite Automata*. Prentice Hall, 1989. 124

[10] R. Dobrushin, S. Ortyukov. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Problems of Information Transmission (translated from Russian)*, 13(1):59–65, 1977. 191

[11] R. Dobrushin, S. Ortyukov. Upper bound for the redundancy of self-correcting arrangements of unreliable elements. *Problems of Information Transmission (translated from Russian)*, 13(3):201–208, 1977. 191

[12] D.-Z. Du, K-I. Ko. *Problem Solving in Automata, Languages, and Complexity*. John Wiley & Sons, 2001. 124

[13] S. N. Elaydi. *An Introduction to Difference Equations*. Springer-Verlag, 1999 (2. edition). 149

[14] I. Gaál. *Diophantine Equations and Power Integral Bases: New Computational Methods*. Birkhäuser Boston, 2002. 59

[15] R. Gallager. *Low-density Parity-check Codes*. The MIT Press, 1963. 191

[16] J. Gathen, von zur, J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999. 59

[17] F. Gécseg, I. Peák. *Algebraic Theory of Automata*. Akadémiai Kiadó, 1972. 123

[18] P. Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1–2):45–267, 2001. See also www.arXiv.org/abs/math.PR/0003117 and The Proceedings of the 1997 Symposium on the Theory of Computing. 192

[19] P. Gács, A. Gál. Lower bounds for the complexity of reliable Boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, 40(2):579–583, 1994. 191

[20] P. Gács, J. Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, 1988. 192

[21] D. Giammarresi, R. Montalbano. Deterministic generalized automata. *Theoretical Computer Science*, 215(1–2):191–208, 1999. 123

[22] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994 (2. edition). 149

[23] D. H. Greene, D. E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhäuser, 1990 (3. edition). 149

[24] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978. 123

[25] J. E. Hopcroft, R. Motwani, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2001 (in German: *Einführung in Automatentheorie, Formale Sprachen und Komplexitätstheorie*, Pearson Studium, 2002). 2. edition. 124

[26] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979. 123

[27] T. W. Hungerford. *Abstract Algebra: An Introduction*. Saunders College Publishers, 1990. 59

[28] D. Kelley. *Automata and Formal Languages*. Prentice Hall, 1995. 124

[29] J. Kim, V. Vu. Generating random regular graphs. In *Proceedings of the Thirty Fifth ACM Symposium on Theory of Computing*, pages 213–222, 2003. 191

[30] D. E. Knuth. *Fundamental Algorithms*, *The Art of Computer Programming* 1. kötete. Addison-Wesley, 1968 (3. updated edition). 149, 150

[31] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1995. 124

[32] Z. Kása. *Combinatorică cu aplicaţii* (*Combinatorics with Applications*). Presa Universitară Clujeană, 2003. 150

[33] A. V. Kuznetsov. Information storage in a memory assembled from unreliablecomponents. *Problems of Information Transmission (translated from Russian)*, 9(3):254–264, 1973. 191

[34] M. V. Lawson. *Finite Automata*. Chapman & Hall/CRC, 2004. 124

[35] A. K. Lenstra, H. W. Lenstra, Jr, L. Lovász. Factoring polynomials with integer coefficients. *Mathematische Annalen*, 261:513–534, 1982. 59

[36] R. Lidl, H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1986. 59

[37] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Barlett Publishers, 2001. 124

[38] M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002. 123

[39] L. Lovász. *Combinatorial Problems and Exercises*. Akadémiai Kiadó, 1979. 150

[40] O. B. Lupanov. On a method of circuit synthesis. *Izvestia VUZ (Radiofizika)*, pages 120–140, 1958. 191

[41] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill Book Co., 1974. 123

[42] A. Meduna. *Automata and Languages: Theory and Applications*. Springer-Verlag, 2000. 124

[43] R. E. Mickens. *Difference Equations. Theory and Applications*. Van Nostrand Reinhold, 1990. 149

[44] R. N. Moll, M. A. Arbib, A. J. Kfoury. *An Introduction to Formal Language Theory*. Springer-Verlag, 1988. 124

[45] R. Motwani, P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. 191

[46] J. Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. Princeton University Press, 1956. 191

[47] M. Pinsker. On the complexity of a concentrator. *International Teletraffic Congr.*, 7:318/1–318/4, 1973. 191, 192

[48] N. Pippenger, G. Staomulis, J. N. Tsitsiklis. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Transactions on Information Theory*, 37(3):639–643, 1991. 191

[49] N. Pippenger. Analysis of error correction by majority voting. In Silvio Micali (szerkesztő), *Randomness in Computation*. JAI Press, 1989. 191

[50] N. Pippenger. On networks of noisy gates. In *Proceeding of the 26th IEE FOCS Symposium*, pages 30–38, 1985. 191

[51] R. Reischuk, B. Schmelz, B.. Reliable computation with noisy circuits and decision trees—a general $n \log n$ lower bound. In *Proceedings of the 32-nd IEEE FOCS Symposium*, pages 602–611, 1991. 191

[52] G. Rozenberg, A. Salomaa. *Handbook of Formal Languages, Vols. I–III.* Springer-Verlag, 1997. 123

[53] A. Salomaa. *Theory of Automata*. Pergamon Press, 1969. 123

[54] A. Salomaa. *Formal Languages*. Academic Press, 1987 (2. updated edition). 123

[55] R. Sedgewick, P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, 1996. 149

[56] C. Shannon. The synthesis of two-terminal switching circuits. *The Bell Systems Technical Journal*, 28:59–98, 1949. 191

[57] I. E. Shparlinski. *Finite Fields: Theory and Computation The Meeting Point of Number Theory, Computer Science, Coding Theory, and Cryptography*. Kluwer Academic Publishers, 1999. 59

[58] M. Simon. *Automata Theory*. World Scientific Publishing Company, 1999. 124

[59] D. A. Simovoci, R. L. Tenney. *Theory of Formal Languages with Applications*. World Scientific Publishing Company, 1999. 124

[60] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997. 123

[61] M. Sipser, D. A. Spielman. Implementing subdivision theory. *Game Developer*, 7(2):40–45, 2000. 191

[62] N. P. Smart. *The Algorithmic Resolution of Diophantine Equations*. London Mathematical Society Student Text, Vol. 41. Cambridge University Press, 1998. 59

[63] D. Spielman. Highly fault-tolerant parallel computation. In *Proceedings of the 37th IEEE Foundations of Computer Science Symposium*, pp. 154–163, 1996. 191

[64] D. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the 27th ACM STOC Symposium*, pp. 387–397, 1995 (Továbbá IEEE Transactions on Information Theory 42(6):1723–1732). 191

[65] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994. 124

[66] T. A. Sudkamp. *Languages and Machines*. Addison-Wesley, 1997. 124

[67] M. G. Taylor. Reliable information storage in memories designed from unreliable components. *The Bell Systems Technical Journal*, 47(10):2299–2337, 1968. 191

[68] N. J. Vilenkin. *Combinatorial Mathematics for Recreation*. Mir, 1972. 149

Name index

Subject Index

Contents