5. Computer Algebra

Computer systems doing various mathematical computations are inevitable in modern science and technology. We are able to compute the orbits of planets and stars, to command nuclear reactors, to describe and model many of the natural forces. These computations can be essentially *numerical and symbolical*.

Although *numerical computations* may involve not only elementary arithmetical operations of numbers (addition, subtraction, multiplication, division) but also more sophisticated calculations, like computing numerical values of mathematical functions, finding roots of polynomials or computing numerical eigenvalues of matrices, these operations can be carried out *only on numbers*. Furthermore, in most cases these numbers are not exact. Their degree of precision depends on the floating-point arithmetic of the given computer hardware architecture.

Unlike numerical calculations, the *symbolic and algebraic computations* operate on *symbols* which represent mathematical objects. These objects may be numbers as integers, rational numbers, real and complex numbers, but may also be polynomials, rational and trigonometric functions, equations, algebraic structures as groups, rings, ideals, algebras or elements of them, or even sets, lists, tables.

Computer systems with the ability to handle symbolic computations are called *computer algebra systems* or *symbolic and algebraic systems* or *formula manipulation systems*. In most cases these systems are able to handle numerical and graphical computations as well. The word „symbolic" is to emphasise that, during the problem solving procedure, the objects are represented by symbols, and the adjective „algebraic" refers to the algebraic origin of the operations on these symbolic objects.

To characterise the notion computer algebra, one can describe it as a collection of *computer programs* developed basically to perform

- exact representations of mathematical objects and
- arithmetic with these objects.

On the other hand, computer algebra can be viewed as a *discipline* which has been developed in order to invent, analyse and implement efficient mathematical algorithms based on exact arithmetic for scientific research and applications.

Since computer algebra systems are able to perform computations essentially with arbitrary precision and error-free first we have to clarify the data structures mapped to the various objects. The Subsection 5.1 deals with the problems of representing mathematical objects.

Further, we describe those symbolic algorithms which are in many area indispensable in the modern science and practice.

The problems of natural sciences are mainly expressed in terms of mathematical equations. Research in solving symbolic linear systems is based upon the well-known elimination methods. Finding the solutions of non-linear systems first we analyse the versions of the *Euclidean algorithm* and the method of *resultants*. In the mid-sixties of the last century Bruno Buchberger in his PhD thesis presented a method to solve multivariate polynomial equations of arbitrary degree. We know this method as the *Gröbner basis method*. At that time the mathematical community paid little attention to his work, but since then it became the base of a powerful set of tools for computing with higher degree polynomial equations. We go into the details in the Subsections 5.2 and 5.3.

The next area to be introduced is the field of *symbolic integration*. Although the nature of the problem was understood long ago (Liouville's principle), it was only 1969 when Robert Risch invented an algorithm to solve the following: given an elementary function $f(x)$ of a real variable x, decide whether the indefinite integral $\int f(x)dx$ is again an elementary function and if so, compute the integral. We describe the method in the Subsection **??**.

At the end of this section we offer a brief survey of the theoretical and practical relations of symbolic algorithms (Subsection 5.5) devoting an independent part to the present computer algebra systems.

5.1. Data representation

In computer algebra, one encounters mathematical objects of different kinds. In order to be able to manipulate these objects on a computer, one first has to represent and store them in the memory of that computer. This can cause several theoretical and practical difficulties. In this subsection we examine these questions.

Consider the *integers*. We know from our curriculum that the set of integers is countable, whereas the computers can store only finitely many of them. The range of values for such a *single-precision integer* is limited by the number of distinct encodings that can be made in the computer word, which is typically 32 or 64 bits in length. Hence, one cannot directly use the computer's integers to represent the mathematical integers, but must be prepared to write programs to handle „arbitrarily" large integers represented by several computer integers. The word arbitrary large does not mean infinitely large since some architectural constraints or the memory size limits in any case. Moreover, one has to construct data structures over which efficient operations can be built. In fact, there are two standard ways of performing such a representation.

- radix notation (a generalisation of conventional decimal notation), in which n is represented as $\sum_{i=0}^{k-1} d_i B^i$, where the digits d_i $(0 \le i \le k-1)$ are single precision integers. These integers can be chosen from the *canonical digit set* $\{0 \le d_i \le B-1\}$ or from the *symmetrical digit set* $\{-\lfloor B/2 \rfloor < d_i \le \lfloor B/2 \rfloor\}$, where the base B could be, in principle, any positive integer greater than 1. For efficiency, B is chosen so that $B-1$ is representable in a single computer word. The length k of the linear list $(d_0, d_1, \ldots, d_{k-1})$ used to represent a *multiprecision integer* may be dynamic (i.e. chosen approximately for the particular integer being represented) or static (i.e. pre-specified fixed length),

depending on whether the linear list is implemented using linked list allocation or using array (sequential) notation. The sign of n is stored within the list, possibly as the sign of d_0 or one or more of the other entries.

- modular notation, in which n is represented by its value modulo a sufficient number of large (but representable in one computer word) primes. From the images one can reconstruct n using the chinese remainder algorithm.

The modular form is fast for addition, subtraction and multiplication but is much slower for divisibility tasks. Hence, the choice of representation influences the algorithms that will be chosen. Indeed, not only the choice of representation influences the algorithms to be used but also the algorithms influence the choice of representation.

Example 5.1 For the sake of simplicity in the next example we work only with natural numbers. Suppose that we have a computer architecture with machine word 32 bits in length, i.e. our computer is able to perform integer arithmetic with the integers in range $I_1 = [0, 2^{32} - 1] = [0, 4\,294\,967\,295]$. Using this arithmetic we carry out a new arithmetic by which we are able to perform integer arithmetic with the integers in range $I_2 = [0, 10^{50}]$.

Using radix representation let $B = 10^4$, and let

$$\begin{aligned} n_1 &= 12345678909876543210123456789 0 , \\ n_2 &= 2110 . \end{aligned}$$

Then,

$$\begin{aligned} n_1 &= [7890, 3456, 1012, 5432, 9876, 7890, 3456, 12] , \\ n_2 &= [2110] , \\ n_1 + n_2 &= [0, 3457, 1012, 5432, 9876, 7890, 3456, 12] , \\ n_1 \cdot n_2 &= [7900, 3824, 6049, 1733, 9506, 9983, 3824, 6049, 2] , \end{aligned}$$

where the addition and the multiplication were computed using radix notation.

Switching to modular representation we have to chose pairwise relative prime integers from the interval I_1 such that their product is greater than 10^{50}. Let for example the primes be

$$\begin{aligned} m_1 = 4294967291, \ m_2 = 4294967279, \ m_3 = 4294967231 , \\ m_4 = 4294967197, \ m_5 = 4294967189, \ m_6 = 4294967161 , \end{aligned}$$

where $\prod_{i=1}^{6} m_i > 10^{50}$. Then, an integer from the interval I_2 can be represented by a 6-tuple from the interval I_1. Therefore

$$\begin{aligned} n_1 \equiv 2009436698 \pmod{m_1}, \quad n_1 \equiv 961831343 \pmod{m_2} , \\ n_1 \equiv 4253639097 \pmod{m_3}, \quad n_1 \equiv 1549708 \pmod{m_4} , \\ n_1 \equiv 2459482973 \pmod{m_5}, \quad n_1 \equiv 3373507250 \pmod{m_6} , \end{aligned}$$

furthermore $n_2 \equiv 2110 \pmod{m_i}$, $(1 \le i \le 6)$. Hence

$$\begin{aligned} n_1 + n_2 &= [2009438808, 961833453, 4253641207, 1551818, 2459485083, 3373509360] , \\ n_1 \cdot n_2 &= [778716563, 2239578042, 2991949111, 3269883880, 1188708718, 1339711723] , \end{aligned}$$

where the addition and the multiplication are carried out by modular arithmetic.

More generally, concerning the choice of representation of other mathematical objects it is worth to distinguish three levels of abstraction:

1. *Object level.* This is the level where the objects are considered as formal mathematical objects. For example $3+3, 4\cdot 4-10$ and 6 are all representations of the integer 6. On the object level the polynomials $(x-1)^2(x+1)$ and x^3-x^2-x+1 are to be considered equal.
2. *Form level.* On this level one has to distinguish between different representations of an object. For example $(x-1)^2(x+1)$ and x^3-x^2-x+1 are to be considered different representations of the same polynomial, namely the former is a product, a latter is a sum.
3. *Data structure level.* On this level one has to consider different ways of representing an object in a computer memory. For example, we distinguish between representations of the polynomial x^3-x^2-x+1 as
   - an array $[1,-1,-1,1]$,
   - a linked list $[1,0]\to[-1,1]\to[-1,2]\to[1,3]$.

In order to represent objects in a computer algebra system, one has to make choices on both the form and the data structure level. Clearly, for many objects various representations are possible. The problem of „how to represent an object" becomes even more difficult when one takes into consideration other criteria, such as memory space, computation time, or readability. Let us see an example. For the polynomial

$$\begin{aligned} f(x) &= (x-1)^2(x+1)^3(2x+3)^4 \\ &= 16x^9-80x^8+88x^7+160x^6-359x^5+x^4+390x^3-162x^2-135x+81 \end{aligned}$$

the product form is more comprehensive, but the second one is more suitable to know the coefficient of, say, x^5. Two other illustrative examples are

- $x^{1000}-1$ and $(x-1)(x^{999}+x^{998}+\cdots+x+1)$,
- $(x+1)^{1000}$ and $x^{1000}+1000x^{999}+\cdots+1000x+1$.

It is very hard to find any good strategy to represent mathematical objects satisfying several criteria. In practice, one object may have several different representations. This, however, gives rise to the problem of detecting equality when different representations of the same object are encountered. In addition, one has to be able to convert from a given representation to others, to be able to simplify the representations.

Consider the *integers.* In the form level one can represent the integers using base B representation, while at the data structure level they can be represented by a linked list or as an array.

Rational numbers can be represented by two integers, a numerator and a denominator. Considering memory constraints one needs to ensure that the rational numbers are in lowest terms and also that the denominator is positive (although other choices, such as numerator positive, are possible as well). This implies that a greatest common divisor computation has to be performed. Since the ring of integers is a Euclidean domain, using the Euclidean algorithm this can be easily computed. The uniqueness of the representation follows from the choice of the denominator's sign.

Multivariate polynomials (elements of $R[x_1,x_2,\ldots,x_n]$, where R is an integral domain) can be represented in the form $a_1x^{e_1}+a_2x^{e_2}+\cdots+a_nx^{e_n}$, where $a_i\in R\setminus\{0\}$ and for $e_i=(e_{i_1},\ldots,e_{i_n})$ one can write x^{e_i} for $x_1^{e_{i_1}}x_2^{e_{i_2}}\cdots x_n^{e_{i_n}}$. In the form level one can consider the

following levels of abstraction:

1. Expanded or factored representation, where the products are multiplied out or the expression is in product form. Compare
   - $x^2y - x^2 + y - 1$, and
   - $\left(x^2 + 1\right)(y - 1)$.
2. Recursive or distributive representation (only for multivariate polynomials). In the bivariate case the polynomial $f(x, y)$ is viewed as an element of the domain $R[x, y]$, $(R[x])[y]$ or $(R[y])[x]$. Compare
   - $x^2y^2 + x^2 + xy^2 - 1$,
   - $(x^2 + x)y^2 + x^2 - 1$, and
   - $(y^2 + 1)x^2 + y^2x - 1$.

At the data structure level there can be dense or sparse representation. Either all terms are considered, or only those having non-zero coefficients. Compare $x^4 + 0x^3 + 0x^2 + 0x - 1$ and $x^4 - 1$. In practice, multivariate polynomials are represented mainly in sparse way.

The traditional approach representing *power series* of form $\sum_{i=0}^{\infty} a_i x^i$ is to truncate at some specified point, and then to regard them essentially as univariate polynomials. However, this is not a real representation since many power series can have the same representation. To overcome this disadvantage, there exist a technique of representing power series by a procedure generating all of the coefficients (rather than by any finite list of coefficients). The generating function is a computable function f such that $f(i) = a_i$. Performing an operation on power series, it is enough to know how to compute the coefficients of the resulting series from the coefficients of the operands. For example, the coefficients h_i of the product of the power series f and g can be computed as $h_i = \sum_{k=0}^{i} f_k g_{i-k}$. In that way, the coefficients are computed when they are needed. This technique is called ***lazy evaluation***.

Since computer algebra programs compute in a symbolic way with arbitrary accuracy, in addition to examining the *time complexity* of the algorithms it is also important to examine their *space complexity.*[1] Consider the simple problem of solving a linear system having n equations an n unknowns with integer coefficients which require ω computer word of storage. Using Gaussian elimination it is easy to see that each coefficient of the reduced linear system may need $2^{n-1}\omega$ computer words of storage. In other words, Gaussian elimination suffers from exponential growth in the size of the coefficients. We remark that if we applied the same method to linear systems having polynomial coefficients, we would have both exponential growth in the size of the numerical coefficients of the polynomials and exponential growth in the degrees of the polynomials themselves. In spite of the observed exponential growth the final result of the Gaussian elimination will always be of reasonable size, because by Cramer's rule we know that each component of the solution to such a linear system is a ratio of two determinants, each of which requires approximately $n\omega$ computer words. The phenomenon described above is called ***intermediate expression swell.*** This appears often in computer algebra algorithms.

[1] We consider the running time as the number of operations executed, according to the RAM-model. Considering the Turing-machine model and using machine words with constant length, we do not have this problem since in this case space is always lowered by the time.

Example 5.2 Using only integer arithmetic we solve the following system of linear equations:

$$\begin{aligned} 37x + 22y + 22z &= 1\,, \\ 31x - 14y - 25z &= 97\,, \\ -11x + 13y + 15z &= -86\,. \end{aligned}$$

First we eliminate the variable x from the second equation. We multiply the first row by 31, the second by -37 and take their sum. If we apply this strategy for the third equation eliminating the variable x we got the following system.

$$\begin{aligned} 37x + 22y + 22z &= 1\,, \\ 1200y + 1607z &= -3558\,, \\ 723y + 797z &= -3171\,. \end{aligned}$$

Now, we eliminate the variable y multiplying the second equation by 723, the third one by -1200, then taking their sum. The result is

$$\begin{aligned} 37x + 22y + 22z &= 1\,, \\ 1200y + 1607z &= -3558\,, \\ 205461z &= 1232766\,. \end{aligned}$$

Continuing this process for eliminating the variables, finally we have the following system:

$$\begin{aligned} 1874311479932400x &= 5622934439797200\,, \\ 246553200y &= -2712085200\,, \\ 205461z &= 1232766\,. \end{aligned}$$

After some simplification we got that $x = 3, y = -11, z = 6$. If we apply greatest common divisor computations in each elimination step the coefficient growth will be less drastic.

In order to avoid the intermediate expression swell phenomenon one uses modular techniques. Instead of performing the operations in the base structure R (e.g. Euclidean ring) they are performed in some factor structure, and then, the result is transformed back to R (Figure 5.1). In general, modular computations can be performed efficiently, and the reconstruction steps can be made with some interpolation strategy. We note that the modular algorithms are very common in computer algebra, but it is not a universal technique.

5.2. Common roots of polynomials

Let R be an integral domain and let

$$f(x) = f_0 + f_1 x + \cdots + f_{m-1}x^{m-1} + f_m x^m \in R[x],\ f_m \neq 0\,, \tag{5.1}$$

$$g(x) = g_0 + g_1 x + \cdots + g_{n-1}x^{n-1} + g_n x^n \in R[x],\ g_n \neq 0 \tag{5.2}$$

arbitrary polynomials with $n, m \in \mathbb{N}, n + m > 0$. Let us give a necessary and sufficient condition of f and g sharing a common root in R.

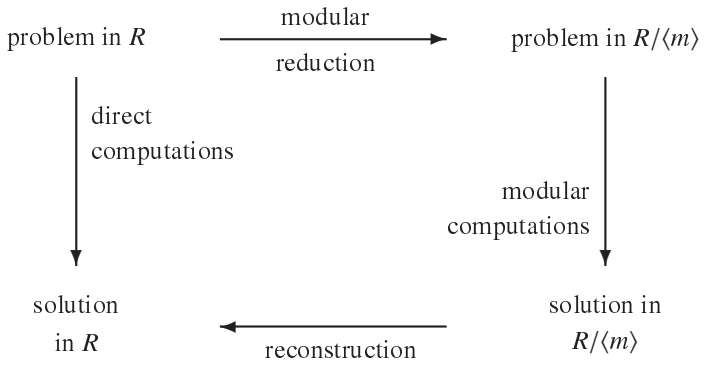

Figure 5.1. The general scheme of modular computations

5.2.1. Classical and extended Euclidean algorithm

If T is a field then $T[x]$ is a Euclidean domain. Recall that we call an integral domain R Euclidean together with the function $\varphi : R \setminus \{0\} \to \mathbb{N}$ if for all $a, b \in R$ $(b \neq 0)$ there exist $q, r \in R$ such that $a = qb + r$, where $r = 0$ or $\varphi(r) < \varphi(b)$; furthermore for all $a, b \in R \setminus \{0\}$ we have $\varphi(ab) \geq \varphi(a)$. The element $q = a \operatorname{quo} b$ is called the ***quotient*** and $r = a \operatorname{rem} b$ is called the ***remainder***. If we are working in a Euclidean domain we would like the greatest common divisor to be unique. For this, a unique element has to be chosen from each equivalence class obtained by multiplying by the units of the ring R. (For example in the case of integers we always choose the non-negative one form the classes $\{0\}, \{-1, 1\}, \{-2, 2\}, \ldots$) Thus every element $a \in R$ has a unique form

$$a = \operatorname{unit}(a) \cdot \operatorname{normal}(a) \ ,$$

where $\operatorname{normal}(a)$ is called the ***normal form*** of a. Let us consider a Euclidean domain $R = T[x]$ over a field T. Let the normal form of $a \in R$ be the corresponding normalised monic polynomial, that is $\operatorname{normal}(a) = a/\operatorname{lc}(a)$, where $\operatorname{lc}(a)$ denotes the leading coefficient of the polynomial a. Let us summarise these important cases:

- If $R = \mathbb{Z}$ then $\operatorname{unit}(a) = \operatorname{sgn}(a)$ $(a \neq 0)$ and $\varphi(a) = \operatorname{normal}(a) = |\, a \,|$,
- if $R = T[x]$ (T is a field) then $\operatorname{unit}(a) = \operatorname{lc}(a)$ (the leading coefficient of the polynomial a with the convention of $\operatorname{unit}(0) = 1$), $\operatorname{normal}(a) = a/\operatorname{lc}(a)$ and $\varphi(a) = \deg a$.

The following algorithm computes the greatest common divisor of two arbitrary elements of a Euclidean domain. We note that this is one of the most ancient algorithms of the world, already known by Euclid around 300 B.C.

iteration	r	c	d
–	–	18	30
1	18	30	18
2	12	18	12
3	6	12	6
4	0	6	0

(a) The operation of CLASSICAL-EUCLIDEAN$(-18, 30)$.

iteration	r	c	d
–	–	$x^4 - \frac{17}{3}x^3 + \frac{13}{3}x^2 - \frac{23}{3}x + \frac{14}{3}$	$x^3 - \frac{20}{3}x^2 + 7x - 2$
1	$4x^2 - \frac{38}{3}x + \frac{20}{3}$	$x^3 - \frac{20}{3}x^2 + 7x - 2$	$4x^2 - \frac{38}{3}x + \frac{20}{3}$
2	$-\frac{23}{4}x + \frac{23}{6}$	$4x^2 - \frac{38}{3}x + \frac{20}{3}$	$-\frac{23}{4}x + \frac{23}{6}$
3	0	$-\frac{23}{4}x + \frac{23}{6}$	0

(b) The operation of CLASSICAL-EUCLIDEAN$(12x^4 - 68x^3 + 52x^2 - 92x + 56, -12x^3 + 80x^2 - 84x + 24)$.

Figure 5.2. Illustration of the operation of the CLASSICAL-EUCLIDEAN algorithm in $\mathbb{Z}$ and $\mathbb{Q}[x]$. In the case **(a)** the input is $a = -18, b = 30, a, b \in \mathbb{Z}$. The first two lines of the pseudocode compute the absolute values of the input numbers. The loop between lines 3 and 6 is executed four times, the values r, c and d in these iterations is shown in the table. The CLASSICAL-EUCLIDEAN$(-18,30)$ algorithm outputs 6 as result. In the case **(b)** the input parameters are $a = 12x^4 - 68x^3 + 52x^2 - 92x + 56, b = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Q}[x]$. The first two lines compute the normal form of the polynomials, and the **while** loop is executed three times. The output of the algorithm is the polynomial normal$(c) = x - 2/3$.

CLASSICAL-EUCLIDEAN(a, b)

```
1  c ← normal(a)
2  d ← normal(b)
3  while d ≠ 0
4        do r ← c rem d
5           c ← d
6           d ← r
7  return normal(c)
```

In the ring of integers the remainder in line 4 becomes $c - \lfloor c/d \rfloor$. When $R = T[x]$, where T is a field, the remainder in line 4 can be calculated by the algorithm EUCLIDEAN-DIVISION-UNIVARIATE-POLYNOMIALS(c, d), the analysis of which is left to Exercise 5.2-1..

Figure 5.2 shows the operation of the CLASSICAL-EUCLIDEAN algorithm in $\mathbb{Z}$ and in $\mathbb{Q}[x]$. We note that in $\mathbb{Z}$ the program only enters the **while** loop with non-negative numbers and the remainder is always non-negative, so the normalisation in line 7 is not needed.

Before examining the running time of the CLASSICAL-EUCLIDEAN algorithm we deal with an extended version of it.

EXTENDED-EUCLIDEAN(a, b)

```
1  (r_0, u_0, v_0) ← (normal(a), 1, 0)
2  (r_1, u_1, v_1) ← (normal(b), 0, 1)
3  while r_1 ≠ 0
4        do q ← r_0 quo r_1
5           r ← r_0 − q r_1
6           u ← (u_0 − q u_1)
7           v ← (v_0 − q v_1)
8           (r_0, u_0, v_0) ← (r_1, u_1, v_1)
9           (r_1, u_1, v_1) ← (r, u, v)
10 return (normal(r_0), u_0/(unit(a) · unit(r_0)), v_0/(unit(b) · unit(r_0)))
```

It is known that in the Euclidean domain R the greatest common divisor of the elements $a, b \in R$ can be expressed in the form $\gcd(a, b) = au + bv$ with appropriate elements $u, v \in R$. However, this pair u, v is not unique. For, if u_0, v_0 are appropriate then so are $u_1 = u_0 + bt$ and $v_1 = v_0 - at$, for all $t \in R$:

$$au_1 + bv_1 = a(u_0 + bt) + b(v_0 - at) = au_0 + bv_0 = \gcd(a, b) \ .$$

The CLASSICAL-EUCLIDEAN algorithm is completed in a way that beside the greatest common divisor it outputs an appropriate pair $u, v \in R$ as discussed above.

Let $a, b \in R$, where R is a Euclidean domain together with the function φ. The equations

$$r_0 = u_0 a + v_0 b \quad \text{and} \quad r_1 = u_1 a + v_1 b \tag{5.3}$$

are obviously fulfilled due to the initialisation in the first two lines of the pseudocode EXTENDED-EUCLIDEAN. We show that the equations (5.3) are invariant under the transformations of the **while** loop of the pseudocode. Let us presume that the conditions (5.3) are fulfilled before an iteration of the loop. Then lines 4–5 of the pseudocode imply

$$r = r_0 - qr_1 = u_0 a + v_0 b - q(au_1 + bv_1) = a(u_0 - qu_1) + b(v_0 - qv_1) \ ,$$

hence, because of lines 6–7

$$r = a(u_0 - qu_1) + b(v_0 - qv_1) = au + bv.$$

Lines 8–9 perform the following operations: u_0, v_0 take the values of u_1 and v_1, then u_1, v_1 take the values of u and v, while r_0, r_1 takes the value of r_1 and r. Thus the equalities in (5.3) are fulfilled after the iteration of the **while** loop as well. Since in each iteration of the loop $\varphi(r_1) < \varphi(r_0)$, the series $\{\varphi(r_i)\}$ obtained in lines 8–9 is a strictly decreasing series of natural numbers, so sooner or later the control steps out of the **while** loop. The greatest common divisor is the last non-zero remainder in the series of Euclidean divisions, that is r_0 in lines 8–9.

Example 5.3 Let us examine the series of remainders in the case of the polynomials

$$a(x) = 63x^5 + 57x^4 - 59x^3 + 45x^2 - 8 \ , \tag{5.4}$$

$$b(x) = -77x^4 + 66x^3 + 54x^2 - 5x + 99 \ : \tag{5.5}$$

$$
\begin{aligned}
r_0 &= x^5 + \frac{19}{21}x^4 - \frac{59}{63}x^3 + \frac{5}{7}x^2 - \frac{8}{63}\,,\\
r_1 &= x^4 - \frac{6}{7}x^3 - \frac{54}{77}x^2 + \frac{5}{77}x - \frac{9}{7}\,,\\
r_2 &= \frac{6185}{4851}x^3 + \frac{1016}{539}x^2 + \frac{1894}{1617}x + \frac{943}{441}\,,\\
r_3 &= \frac{771300096}{420796475}x^2 + \frac{224465568}{420796475}x + \frac{100658427}{38254225}\,,\\
r_4 &= -\frac{125209969836038125}{113868312759339264}x - \frac{3541728593586625}{101216278008301568}\,,\\
r_5 &= \frac{4717580163635699927436051 21}{1803229860333151158054368 75}\,.
\end{aligned}
$$

The values of the variables u_0, v_0 before the execution of line 10:

$$
\begin{aligned}
u_0 &= \frac{113868312759339264}{125209969836038125}x^3 - \frac{662639052858978333785656224}{819649936515068708206531 25}x^2\\
&\quad - \frac{172214445262403690128205 6661}{901614930166575579027184375}x + \frac{145175798748706922498167 8954}{901614930166575579027184375}\,,\\
v_0 &= -\frac{113868312759339264}{125209969836038125}x^4 - \frac{650693816081111838878813536}{819649936515068708206531 25}x^3\\
&\quad + \frac{178270505434627626751446079}{819649936515068708206531 25}x^2 + \frac{638085922305129542614 6353}{819649936515068708206531 25}x\\
&\quad - \frac{179818001183413133012445617}{819649936515068708206531 25}\,.
\end{aligned}
$$

The return values are:

$$
\begin{aligned}
\gcd(a,b) &= 1,\\
u &= \frac{2580775248128}{467729710968369}x^3 - \frac{3823697946464}{779549518280615}x^2\\
&\quad - \frac{27102209423483}{2338648554841845}x + \frac{7615669511954}{779549518280615}\,,\\
v &= \frac{703847794944}{155909903656123}x^4 + \frac{3072083769824}{779549518280615}x^3\\
&\quad - \frac{25249752472633}{2338648554841845}x^2 - \frac{301255883677}{779549518280615}x + \frac{25468935587159}{2338648554841845}\,.
\end{aligned}
$$

We can see that the sizes of the coefficients show a drastic growth. One might ask: why don't we normalise in *every* iteration of the **while** loop? This idea leads to the normalised version of the Euclidean algorithm for polynomials.

EXTENDED-EUCLIDEAN-NORMALISED(a, b)

```
1  e_0 ← unit(a)
2  (r_0, u_0, v_0) ← (normal(a), e_0^{-1}, 0)
3  e_1 ← unit(b)
4  (r_1, u_1, v_1) ← (normal(b), 0, e_1^{-1})
5  while r_1 ≠ 0
6        do q ← r_0 quo r_1
7           s ← r_0 rem r_1
8           e ← unit(s)
9           r ← normal(s)
10          u ← (u_0 − q u_1)/e
11          v ← (v_0 − q v_1)/e
12          (r_0, u_0, v_0) ← (r_1, u_1, v_1)
13          (r_1, u_1, v_1) ← (r, u, v)
14 return (r_0, u_0, v_0)
```

Example 5.4 Let us look at the series of remainders and the series e obtained in the EXTENDED-EUCLIDEAN-NORMALISED algorithm in the case of the polynomials (5.4) and (5.5)

$$\begin{aligned}
r_0 &= x^5 + \frac{19}{21}x^4 - \frac{59}{63}x^3 + \frac{5}{7}x^2 - \frac{8}{63}, & e_0 &= 63\,,\\
r_1 &= x^4 - \frac{6}{7}x^3 - \frac{54}{77}x^2 + \frac{5}{77}x - \frac{9}{7}, & e_1 &= -77\,,\\
r_2 &= x^3 + \frac{9144}{6185}x^2 + \frac{5682}{6185}x + \frac{10373}{6185}, & e_2 &= \frac{6185}{4851}\,,\\
r_3 &= x^2 + \frac{2338183}{8034376}x + \frac{369080899}{257100032}, & e_3 &= \frac{771300096}{420796475}\,,\\
r_4 &= x + \frac{166651173}{5236962760}, & e_4 &= -\frac{222685475860375}{258204790837504}\,,\\
r_5 &= 1, & e_5 &= \frac{156579848512133360531}{109703115798507270400}\,.
\end{aligned}$$

Before the execution of line 14 of the pseudocode the values of the variables $\gcd(a, b) = r_0, u = u_0, v = v_0$ are:

$$\begin{aligned}
\gcd(a,b) &= 1,\\
u &= \frac{2580775248128}{467729710968369}x^3 - \frac{3823697946464}{779549518280615}x^2\\
&\quad - \frac{27102209423483}{2338648554841845}x + \frac{7615669511954}{779549518280615}\,,\\
v &= \frac{703847794944}{155909903656123}x^4 + \frac{3072083769824}{779549518280615}x^3\\
&\quad - \frac{25249752472633}{2338648554841845}x^2 - \frac{301255883677}{779549518280615}x + \frac{25468935587159}{2338648554841845}\,.
\end{aligned}$$

Looking at the size of the coefficients in $\mathbb{Q}[x]$ the advantage of the normalised version is obvious, but we could not avoid the growth this way either. To get a machine architecture-dependent description and analysis of the EXTENDED-EUCLIDEAN-NORMALISED algorithm we

introduce the following notation. Let

$$\begin{aligned}
\lambda(a) &= \lfloor \log_2 |a|/w \rfloor + 1, \text{ if } a \in \mathbb{Z} \setminus \{0\}, \text{ and } \lambda(0) = 0 \ , \\
\lambda(a) &= \max\{\lambda(b), \lambda(c)\}, \text{ if } a = b/c \in \mathbb{Q},\ b, c \in \mathbb{Z},\ \gcd(b, c) = 1 \ , \\
\lambda(a) &= \max\{\lambda(b), \lambda(a_0), \ldots, \lambda(a_n)\}, \text{ if } a = \sum_{0 \le i \le n} a_i x^i / b \in \mathbb{Q}[x] \ , \\
&\quad a_i \in \mathbb{Z},\ b \in \mathbb{N}^+,\ \gcd(b, a_0, \ldots, a_n) = 1 \ ,
\end{aligned}$$

where w is the word length of the computer in bits. It is easy to verify that if $a, b \in \mathbb{Z}[x]$ and $c, d \in \mathbb{Q}$ then

$$\begin{aligned}
\lambda(c + d) &\le \lambda(c) + \lambda(d) + 1 \ , \\
\lambda(a + b) &\le \max\{\lambda(a), \lambda(b)\} + 1 \ , \\
\lambda(cd), \lambda(c/d) &\le \lambda(c) + \lambda(d) \ , \\
\lambda(ab) &\le \lambda(a) + \lambda(b) + \lambda(\min\{\deg a, \deg b\} + 1) \ .
\end{aligned}$$

We give the following theorems without proof.

Theorem 5.1 *If $a, b \in \mathbb{Z}$ and $\lambda(a) = m \ge n = \lambda(b)$ then the* Classical-Euclidean *and* Extended-Euclidean *algorithms require $O(mn)$ machine-word arithmetic operations.*

Theorem 5.2 *If F is a field and $a, b \in F[x]$, $\deg(a) = m \ge n = \deg(b)$ then the* Classical-Euclidean, Extended-Euclidean *and* Extended-Euclidean-normalised *algorithms require $O(mn)$ elementary operations in F.*

Is the growth of the coefficients perhaps due to the choice of our polynomials? Let us examine a single Euclidean division in the Extended-Euclidean-normalised algorithm. Let $a = bq + e^* r$, where

$$\begin{aligned}
a &= x^m + \frac{1}{c} \sum_{i=0}^{m-1} a_i x^i \in \mathbb{Q}[x] \ , \\
b &= x^n + \frac{1}{d} \sum_{i=0}^{n-1} b_i x^i \in \mathbb{Q}[x] \ ,
\end{aligned}$$

and $r \in \mathbb{Q}[x]$ are monic polynomials, $a_i, b_i \in \mathbb{Z}$, $e^* \in \mathbb{Q}$, $c, d \in \mathbb{N}^+$, and consider the case $n = m - 1$. Then

$$\begin{aligned}
q &= x + \frac{a_{m-1} d - b_{n-1} c}{cd} \ , \\
\lambda(q) &\le \lambda(a) + \lambda(b) + 1 \ , \\
e^* r &= a - qb = \frac{acd^2 - xbcd^2 - (a_{m-1} d - b_{n-1} c) bd}{cd^2} \ , \\
\lambda(e^* r) &\le \lambda(a) + 2\lambda(b) + 3 \ . \qquad (5.6)
\end{aligned}$$

Note that the bound (5.6) is valid for the coefficients of the remainder polynomial r as well, that is $\lambda(r) \le \lambda(a) + 2\lambda(b) + 3$. So in the case when $\lambda(a) \sim \lambda(b)$ the size of the coefficients may only grow by a factor of around three in each Euclidean division. This estimate seems

accurate for pseudorandom polynomials, the interested reader should look at problem 5-1.. The worst case estimate suggests that

$$\lambda(r_l) = O(3^l \cdot \max\{\lambda(a), \lambda(b)\}),$$

where l denotes the running time of the EXTENDED-EUCLIDEAN-NORMALISED algorithm, practically the number of times the **while** loop is executed. Luckily, this exponential growth is not achieved in each iteration of the loop and altogether the growth of the coefficients is polynomially bounded in terms of the input. Later we shall see that using modular techniques the growth can be totally eliminated.

Summarising: after computing the greatest common divisor of the polynomials $f, g \in R[x]$ (R is a field) f and g have a common root if and only if their greatest common divisor is not a constant. For, if $\gcd(f, g) = d \in R[x]$ is not a constant then the roots of d are the roots of f and g, too, since d divides f and g. On the other hand, if f and g have a root in common then their greatest common divisor cannot be a constant, since the common root is the root of it, too.

5.2.2. Primitive Euclidean algorithm

If R is a UFD (unique factorisation domain, where every non-zero non-unit element can be written as a product of irreducible elements in a unique way up to reordering and multiplication by units) but not necessarily a Euclidean domain then the situation is more complicated, since we may not have a Euclidean algorithm in $R[x]$. Luckily there are several useful methods due to: (1) unique factorisation in $R[x]$, (2) the existence of a greatest common divisor of two or more arbitrary elements.

The first possible method is performing the calculations in the field of fractions of R. The polynomial $p(x) \in R[x]$ is called a ***primitive polynomial*** if there is no prime in R that divides all coefficients of $p(x)$. A famous lemma by Gauss says that the product of primitive polynomials is again primitive, hence for the primitive polynomials f, g, $d = \gcd(f, g) \in R[x]$ if and only if $d = \gcd(f, g) \in H[x]$, where H denotes the field of fractions of R. So we can calculate greatest common divisors in $H[x]$ instead of $R[x]$. Unfortunately, this approach is not really effective because arithmetic in the field of fractions H is much more expensive than in R.

A second possibility is an algorithm similar to the Euclidean algorithm: in the ring of polynomials in one variable over an integral domain a so called ***pseudo-division*** can be defined. Using the polynomials (5.1), (5.2) if $m \geq n$ then there exist $q, r \in R[x]$, such that

$$g_n^{m-n+1} f = gq + r \ ,$$

where $r = 0$ or $\deg r < \deg g$. The polynomial q is called the ***pseudo-quotient*** of f and g and r is called ***pseudo-remainder***. The notation is $q = \mathrm{pquo}(f, g), r = \mathrm{prem}(f, g)$.

Example 5.5 Let

$$f(x) = 12x^4 - 68x^3 + 52x^2 - 92x + 56 \in \mathbb{Z}[x] \ , \tag{5.7}$$

$$g(x) = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x] \ . \tag{5.8}$$

then $\mathrm{pquo}(f, g) = -144(x + 1)$, $\mathrm{prem}(f, g) = 1152(6x^2 - 19x + 10)$.

iteration	r	c	d
–	–	$3x^4 - 17x^3 + 13x^2 - 23x + 14$	$-3x^3 + 20x^2 - 21x + 6$
1	$108x^2 - 342x + 108$	$-3x^3 + 20x^2 - 21x + 6$	$6x^2 - 19x + 10$
2	$621x - 414$	$6x^2 - 19x + 10$	$3x - 2$
3	0	$3x - 2$	0

Figure 5.3. The illustration of the operation of the PRIMITIVE-EUCLIDEAN algorithm with input $a(x) = 12x^4 - 68x^3 + 52x^2 - 92x + 56$, $b(x) = -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x]$. The first two lines of the program compute the primitive parts of the polynomials. The loop between lines 3 and 6 is executed three times, the table shows the values of r, c and d in the iterations. In line 7 the variable γ equals $\gcd(4, 4) = 4$. The PRIMITIVE-EUCLIDEAN(a, b) algorithm returns $4 \cdot (3x - 2)$ as result.

On the other hand, each polynomial $f(x) \in R[x]$ can be written in a unique form

$$f(x) = \mathrm{cont}(f) \cdot \mathrm{pp}(f)$$

up to a unit factor, where $\mathrm{cont}(f) \in R$ and $\mathrm{pp}(f) \in R[x]$ are primitive polynomials. Then $\mathrm{cont}(f)$ is called the ***content,*** $\mathrm{pp}(f)$ is called the ***primitive part*** of $f(x)$. The uniqueness of the form can be achieved by the normalisation of units. For example in the case of integers, we always choose the positive number from the equivalence classes of $\mathbb{Z}$.

The following algorithm performs a series of pseudo-divisions. The algorithm uses the function prem(), which computes the pseudo-remainder; and it assumes that we can calculate greatest common divisor in R and content and primitive part in $R[x]$. The input is $a, b \in R[x]$, where R is a UFD. The output is the polynomial $\gcd(a, b) \in R[x]$.

PRIMITIVE-EUCLIDEAN(a, b)

```
1  c ← pp(f)
2  d ← pp(g)
3  while d ≠ 0
4        do r ← prem(c, d)
5           c ← d
6           d ← pp(r)
7  γ ← gcd(cont(a), cont(b))
8  δ ← γc
9  return δ
```

The operation of the algorithm is illustrated by figure 5.3. The running time of the PRIMITIVE-EUCLIDEAN algorithm is the same as the running time of the previous versions of the Euclidean algorithm.

The PRIMITIVE-EUCLIDEAN algorithm is very important because the ring $R[x_1, x_2, \ldots, x_t]$ of multivariate polynomials is a UFD, so we apply the algorithm recursively, e.g. in $R[x_2, \ldots, x_t][x_1]$, using computations in the UFDs $R[x_2, \ldots, x_t], \ldots, R[x_t]$. In other words, the recursive view of multivariate polynomial rings leads to the recursive application of the PRIMITIVE-EUCLIDEAN algorithm in a straightforward way.

We may note that, like above, the algorithm shows a growth in the coefficients.

Let us take a detailed look at the UFD $\mathbb{Z}[x]$. The bound on the size of the coefficients

of the greatest common divisor is given by the following theorem, which we state without proof.

Theorem 5.3 (Landau-Mignotte). *Let* $a(x) = \sum_{i=0}^{m} a_i x^i$, $b(x) = \sum_{i=0}^{n} b_i x^i \in \mathbb{Z}[x]$, $a_m \neq 0 \neq b_n$, *and* $b(x) \mid a(x)$. *Then*

$$\sum_{i=1}^{n} |b_i| \leq 2^n \left|\frac{b_n}{a_m}\right| \sqrt{\sum_{i=0}^{m} a_i^2} \ .$$

Corollary 5.4 *With the notations of the previous theorem, the absolute value of any coefficient of the polynomial* $gcd(a, b) \in \mathbb{Z}[x]$ *is smaller than*

$$2^{\min\{m,n\}} \cdot gcd(a_m, b_n) \cdot \min\left\{\frac{1}{|a_m|}\sqrt{\sum_{i=1}^{m} a_i^2}, \frac{1}{|b_n|}\sqrt{\sum_{i=1}^{n} b_i^2}\right\} .$$

Proof. The greatest common divisor of a and b obviously divides a and b and its degree is at most the minimum of their degrees. Further, the leading coefficient of the greatest common divisor divides a_m and b_n, so it divides $\gcd(a_m, b_n)$, too. ∎

Example 5.6 Corollary 5.4 implies that the absolute value of the coefficients of the greatest common divisor is at most $\lfloor 32/9\sqrt{3197}\rfloor = 201$ for the polynomials (5.4), (5.5), and at most $\lfloor 32\sqrt{886}\rfloor = 952$ for the polynomials (5.7) and (5.8).

5.2.3. The resultant

The following method describes the necessary and sufficient conditions for the common roots of (5.1) and (5.2) in the most general context. As a further advantage, it can be applied to solve algebraic equation systems of higher degree.

Let R be an integral domain and H its field of fractions. Let us consider the smallest extension K of H over which both $f(x)$ of (5.1) and $g(x)$ of (5.2) splits into linear factors. Let us denote the roots (in K) of the polynomial $f(x)$ by $\alpha_1, \alpha_2, \ldots, \alpha_m$, and the roots of $g(x)$ by $\beta_1, \beta_2, \ldots, \beta_n$. Let us form the following product:

$$\begin{aligned} \operatorname{res}(f,g) &= f_m^n g_n^m (\alpha_1-\beta_1)(\alpha_1-\beta_2)\cdots(\alpha_1-\beta_n) \\ &\quad \cdot(\alpha_2-\beta_1)(\alpha_2-\beta_2)\cdots(\alpha_2-\beta_n) \\ &\quad \vdots \\ &\quad \cdot(\alpha_m-\beta_1)(\alpha_m-\beta_2)\cdots(\alpha_m-\beta_n) \\ &= f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(\alpha_i-\beta_j) \ . \end{aligned}$$

It is obvious that $\operatorname{res}(f, g)$ equals 0 if and only if for some i and j we have $\alpha_i = \beta_j$, that is f and g have a common root. We call this product $\operatorname{res}(f, g)$ the ***resultant*** of the polynomials f and g. Note that the value of the resultant depends on the order of f and g, but the resultants

obtained in the two ways can only differ in sign.

$$\begin{aligned}\operatorname{res}(g,f) &= g_n^m f_m^n \prod_{j=1}^{n}\prod_{i=1}^{m}(\beta_j-\alpha_i)\\ &= (-1)^{mn} f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(\alpha_i-\beta_j) = (-1)^{mn}\operatorname{res}(f,g)\ .\end{aligned}$$

Evidently, this form of the resultant cannot be applied in practice, since it presumes the knowledge of the roots. Let us examine the different forms of the resultant. Since

$$\begin{aligned}f(x) &= f_m(x-\alpha_1)(x-\alpha_2)\cdots(x-\alpha_m) \quad (f_m\neq 0)\ ,\\ g(x) &= g_n(x-\beta_1)(x-\beta_2)\cdots(x-\beta_n) \quad (g_n\neq 0)\ ,\end{aligned}$$

hence

$$\begin{aligned}g(\alpha_i) &= g_n(\alpha_i-\beta_1)(\alpha_i-\beta_2)\cdots(\alpha_i-\beta_n)\\ &= g_n\prod_{j=1}^{n}(\alpha_i-\beta_j)\ .\end{aligned}$$

Thus

$$\begin{aligned}\operatorname{res}(f,g) &= f_m^n\prod_{i=1}^{m}\Big(g_n\prod_{j=1}^{n}(\alpha_i-\beta_j)\Big)\\ &= f_m^n\prod_{i=1}^{m}g(\alpha_i) = (-1)^{mn}g_n^m\prod_{j=1}^{n}f(\beta_j)\ .\end{aligned}$$

Although it looks a lot more friendly, this form still requires our knowledge of the roots of at least one polynomial. We next examine how the resultant may be expressed only in terms of the coefficients of the polynomials. This leads to the Sylvester form of the resultant.

Let us presume that the polynomial f in (5.1) and the polynomial g in (5.2) have a common root. This means that there exists a number $\alpha\in K$ for which

$$\begin{aligned}f(\alpha) &= f_m\alpha^m+f_{m-1}\alpha^{m-1}+\cdots+f_1\alpha+f_0=0\ ,\\ g(\alpha) &= g_n\alpha^n+g_{n-1}\alpha^{n-1}+\cdots+g_1\alpha+g_0=0\ .\end{aligned}$$

Multiply these equations by the numbers $\alpha^{n-1},\alpha^{n-2},\ldots,\alpha,1$ and $\alpha^{m-1},\alpha^{m-2},\ldots,\alpha,1$, respectively. We get n equations from the first one and m from the second one. We consider these $m+n$ equations as a homogeneous system of linear equations in $m+n$ indeterminates. This system has the obviously non-trivial solution $\alpha^{m+n-1},\alpha^{m+n-2},\ldots,\alpha,1$. It is a well-known fact that a homogeneous system with as many equations as indeterminates has non-trivial solutions if and only if its determinant is zero. We get that f and g can only have

common roots if the determinant

$$D = \begin{vmatrix} f_m & \cdots & \cdots & \cdots & f_0 & & & \uparrow \\ & \ddots & & & & \ddots & & n \\ & & f_m & \cdots & \cdots & \cdots & f_0 & \downarrow \\ g_n & \cdots & \cdots & g_0 & & & & \uparrow \\ & \ddots & & & \ddots & & & m \\ & & \ddots & & & \ddots & & \\ & & & g_n & \cdots & \cdots & g_0 & \downarrow \end{vmatrix} \tag{5.9}$$

equals 0 (there are 0s everywhere outside the dotted areas). Thus a necessary condition for the existence of common roots is that the determinant D of order $(m+n)$ be 0. Below we prove that D equals the resultant of f and g, hence $D = 0$ is a sufficient condition for common roots, too. The determinant (5.9) is called the ***Sylvester form*** of the resultant.

Theorem 5.5 *With the notations above*

$$D = f_m^n \prod_{i=1}^{m} g(\alpha_i) \, .$$

Proof. We shall procede by induction on m. If $m = 0$ then $f = f_m = f_0$, so the right-hand side is f_0^n. The left hand side is a determinant of order n with f_0 everywhere in the diagonal, and 0 everywhere else. Thus $D = f_0^n$, the statement is true. In the following presume that $m > 0$ and the statement is true for $m - 1$. So if we take the polynomial

$$f^*(x) = f_m(x - \alpha_1) \cdots (x - \alpha_{m-1}) = f^*_{m-1}x^{m-1} + f^*_{m-2}x^{m-2} + \cdots + f^*_1 x + f^*_0$$

instead of f then f^* and g fulfill the condition:

$$D^* = \begin{vmatrix} f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 & & & \\ & \ddots & & & & \ddots & & \\ & & f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 & \\ g_n & \cdots & \cdots & g_0 & & & & \\ & \ddots & & & \ddots & & & \\ & & \ddots & & & \ddots & & \\ & & & g_n & \cdots & \cdots & g_0 & \end{vmatrix} = f^{*m}_{m-1} \prod_{i=1}^{m-1} g(\alpha_i) \, .$$

Since $f = f^*(x - \alpha_m)$, the coefficients of f and f^* satisfy

$$f_m = f^*_{m-1}, f_{m-1} = f^*_{m-2} - f^*_{m-1}\alpha_m, \ldots, f_1 = f^*_0 - f^*_1\alpha_m, f_0 = -f^*_0\alpha_m.$$

Thus

$$D = \begin{vmatrix} f^*_{m-1} & f^*_{m-2} - f^*_{m-1}\alpha_m & \cdots & \cdots & -f^*_0\alpha_m & & & \\ & \ddots & & & & & \ddots & \\ & & & f^*_{m-1} & \cdots & \cdots & \cdots & -f^*_0\alpha_m \\ g_n & \cdots & \cdots & g_0 & & & & \\ & \ddots & & & & \ddots & & \\ & & & \ddots & & & \ddots & \\ & & & & g_n & \cdots & \cdots & g_0 \end{vmatrix} .$$

We transform the determinant in the following way: we add α_m times the first column to the second column, then we add α_m times the new second column to the third column, etc. In this way the α_m-s disappeared from the first n lines, so the first n lines of D^* and the transformed D are identical. In the last m rows, subtract α_m times the second one from the first one, and similarly, always subtract α_m times a row from the row right above it. In the end, D becomes

$$D = \begin{vmatrix} f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 & & & & \\ & \ddots & & & & \ddots & & & \\ & & f^*_{m-1} & \cdots & \cdots & \cdots & f^*_0 & & \\ g_n & \cdots & \cdots & g_0 & & & & & \\ & \ddots & & & \ddots & & & & \\ & & \ddots & & & \ddots & & & \\ & & & g_n & \cdots & \cdots & g_0 & & \\ & & & & g_n & g_n\alpha_m + g_{n-1} & \cdots & g(\alpha_m) \end{vmatrix} .$$

Using the last row for expansion we get $D = D^* g(\alpha_m)$ which implies $D = f_m^n \prod_{i=1}^m g(\alpha_i)$ by the induction hypothesis. ■

We get that $D = \mathrm{res}(f, g)$, that is the polynomials f and g have a common root in K if and only if the determinant D vanishes.

From an algorithmic point of view, the computation of the resultant in Sylvester form for higher degree polynomials means the computation of a large determinant. The following theorem implies that pseudo-division may simplify the computation.

Theorem 5.6 *For the polynomials f of (5.1) and g of (5.2) in the case of $m \geq n > 0$*

$$\begin{cases} res(f,g) = 0, & \text{if } prem(f,g) = 0\ , \\ g_n^{(m-n)(n-1)+d} res(f,g) = (-1)^{mn} res(g,r), & \text{if } r = prem(f,g) \neq 0 \text{ and } d = \deg(r)\ . \end{cases}$$

Proof. Multiply the first line of the determinant (5.9) by g_n^{m-n+1}. Let $q = q_{m-n}x^{m-n} + \cdots + q_0 \in R[x]$ and $r = r_d x^d + \cdots + r_0 \in R[x]$ be the uniquely determined polynomials with

$$\begin{aligned} g_n^{m-n+1}(f_m x^m + \cdots + f_0) &= (q_{m-n}x^{m-n} + \cdots + q_0)(g_n x^n + \cdots + g_0) \\ &\quad + r_d x^d + \cdots + r_0\ , \end{aligned}$$

where $r = \mathrm{prem}(f, g)$. Then multiplying row $(n+1)$ of the resultant by q_{m-n}, row $(n+2)$ by

q_{m-n-1} etc. and subtracting from the first line we get the determinant

$$g_n^{m-n+1}\mathrm{res}(f,g) = \begin{vmatrix} 0 & \cdots & 0 & r_d & \cdots & \cdots & r_0 & & & & \\ & f_m & \cdots & \cdots & \cdots & \cdots & \cdots & f_0 & & & \\ & & \ddots & & & & & & \ddots & & \\ & & & f_m & \cdots & \cdots & \cdots & \cdots & \cdots & f_0 & \\ g_n & \cdots & \cdots & \cdots & g_0 & & & & & & \\ & \ddots & & & & \ddots & & & & & \\ & & \ddots & & & & \ddots & & & & \\ & & & \ddots & & & & \ddots & & & \\ & & & & \ddots & & & & \ddots & & \\ & & & & & g_n & \cdots & \cdots & \cdots & g_0 & \end{vmatrix}.$$

Here r_d is in the $(m-d+1)$th column of the first row, and r_0 is in the $(m+1)$th column of the first row.

Similarly, let us multiply the second row by g_n^{m-n+1}, then multiply rows $(n+2)$, $(n+3)$, ... by q_{m-n}, q_{m-n-1} etc., and subtract them from the second row. Continue in the same way for the third, ..., nth row. The result:

$$g_n^{n(m-n+1)}\mathrm{res}(f,g) = \begin{vmatrix} & & & r_d & \cdots & \cdots & r_0 & & & & \\ & & & & \ddots & & & \ddots & & & \\ & & & & & \ddots & & & \ddots & & \\ & & & & & & r_d & \cdots & \cdots & r_0 & \\ g_n & \cdots & \cdots & \cdots & g_0 & & & & & & \\ & \ddots & & & & \ddots & & & & & \\ & & \ddots & & & & \ddots & & & & \\ & & & \ddots & & & & \ddots & & & \\ & & & & \ddots & & & & \ddots & & \\ & & & & & g_n & \cdots & \cdots & \cdots & g_0 & \end{vmatrix}.$$

After reordering of rows

$$g_n^{n(m-n+1)}\mathrm{res}(f,g) = (-1)^{mn}\begin{vmatrix} g_n & \cdots & \cdots & \cdots & g_0 & & & & & \\ & \ddots & & & & \ddots & & & & \\ & & \ddots & & & & \ddots & & & \\ & & & g_n & \cdots & \cdots & \cdots & g_0 & & \\ & & & & \ddots & & & & \ddots & \\ & & & & & g_n & \cdots & \cdots & \cdots & g_0 \\ & & & r_d & \cdots & \cdots & r_0 & & & \\ & & & & \ddots & & & \ddots & & \\ & & & & & \ddots & & & \ddots & \\ & & & & & & r_d & \cdots & \cdots & r_0 \end{vmatrix} .$$

Note that

$$\begin{vmatrix} g_n & \cdots & \cdots & \cdots & g_0 & & \\ & \ddots & & & & \ddots & \\ & & g_n & \cdots & \cdots & \cdots & g_0 \\ r_d & \cdots & \cdots & r_0 & & & \\ & \ddots & & & \ddots & & \\ & & \ddots & & & \ddots & \\ & & & r_d & \cdots & \cdots & r_0 \end{vmatrix} = \mathrm{res}(g,r) ,$$

thus

$$g_n^{n(m-n+1)}\mathrm{res}(f,g) = (-1)^{mn} g_n^{m-d}\mathrm{res}(g,r) ,$$

and then

$$g_n^{(m-n)(n-1)+d}\mathrm{res}(f,g) = (-1)^{mn}\mathrm{res}(g,r) \tag{5.10}$$

follows. ■

Equation (5.10) describes an important relationship. Instead of computing the possibly gigantic determinant $\mathrm{res}(f,g)$ we perform a series of pseudo-divisions and apply (5.10) in every step. We calculate the resultant only when no more pseudo-division can be done. An important consequence of the theorem is

Corollary 5.7 *There exist polynomials $u, v \in R[x]$ for which $res(f,g) = fu + gv$ with* $\deg u < \deg g$, $\deg v < \deg f$.

Proof. Multiply the ith column in the determinant form of the resultant by x^{m+n-i} and add it to the last column for each $i = 1, \ldots, (m+n-1)$. The result is:

$$\mathrm{res}(f,g) = \begin{vmatrix} f_m & \cdots & \cdots & f_0 & \cdots & x^{n-1}f \\ & \ddots & & & \ddots & \vdots \\ & & f_m & \cdots & \cdots & f \\ g_n & \cdots & \cdots & g_0 & \cdots & x^{m-1}g \\ & \ddots & & & \ddots & \vdots \\ & & g_n & \cdots & \cdots & g \end{vmatrix} .$$

Using the last column for expansion and factoring out f and g we get the statement with the restrictions on the degrees. ■

The most important benefit of the resultant method, compared to the previously discussed methods, is that the input polynomials may contain symbolic coefficients as well.

Example 5.7 Let

$$\begin{aligned} f(x) &= 2x^3 - \xi x^2 + x + 3 \in \mathbb{Q}[x] , \\ g(x) &= x^2 - 5x + 6 \in \mathbb{Q}[x] . \end{aligned}$$

Then the existence of common rational roots of f and g cannot be decided by variants of the Euclidean algorithm, but we can decide it with the resultant method. Such a root exists if and only if

$$\operatorname{res}(f,g) = \begin{vmatrix} 2 & -\xi & 1 & 3 & \\ & 2 & -\xi & 1 & 3 \\ 1 & -5 & 6 & & \\ & 1 & -5 & 6 & \\ & & 1 & -5 & 6 \end{vmatrix} = 36\xi^2 - 429\xi + 1260 = 3(4\xi - 21)(3\xi - 20) = 0 ,$$

that is when $\xi = 20/3$ or $\xi = 21/4$.

The significance of the resultant is not only that we can decide the existence of common roots of polynomials, but also that using it we can reduce the solution of algebraic equation systems to solving univariate equations.

Example 5.8 Let

$$f(x,y) = x^2 + xy + 2x + y - 1 \in \mathbb{Z}[x,y] , \tag{5.11}$$

$$g(x,y) = x^2 + 3x - y^2 + 2y - 1 \in \mathbb{Z}[x,y] . \tag{5.12}$$

Consider the polynomials f and g as elements of $(\mathbb{Z}[x])[y]$. They have a common root if and only if

$$\operatorname{res}_y(f,g) = \begin{vmatrix} x+1 & x^2+2x-1 & 0 \\ 0 & x+1 & x^2+2x-1 \\ -1 & 2 & x^2+3x-1 \end{vmatrix} = -x^3 - 2x^2 + 3x = 0 .$$

Common roots in $\mathbb{Z}$ can exist for $x \in \{-3, 0, 1\}$. For each x we substitute into the equations (5.11), (5.12) (already in $\mathbb{Z}[y]$) and get that the integer solutions are $(-3, 1), (0, 1), (1, -1)$.

We note that the resultant method can be applied for the solution of polynomial equations in several variables, too, but it is not really effective. One problem is that computational space explosion occurs in the computation of the determinant. We note that computing the resultant of two univariate polynomials in determinant form with the usual Gauss-elimination requires $O((m+n)^3)$ operations, while variants of the Euclidean algorithm are quadratic. The other problem is that computational complexity depends strongly on the order of the indeterminates. *Eliminating all variables together* in a polynomial equation system is much more effective. This leads to the introduction of multivariate resultants.

5.2.4. Modular greatest common divisor

All methods considered so far for the existence and calculation of common roots of polynomials are characterised by an explosion of computational space. The natural question

arises: can we apply modular techniques? Below we examine the case $a(x), b(x) \in \mathbb{Z}[x]$ with $(a, b \neq 0)$. Let us consider the polynomials (5.4), (5.5) $\in \mathbb{Z}[x]$ and let $p = 13$ a prime number. Then the series of remainders in $\mathbb{Z}_p[x]$ in the Classical-Euclidean algorithm is the following:

$$\begin{aligned} r_0 &= 11x^5 + 5x^4 + 6x^3 + 6x^2 + 5 \,, \\ r_1 &= x^4 + x^3 + 2x^2 + 8x + 8 \,, \\ r_2 &= 3x^3 + 8x^2 + 12x + 1 \,, \\ r_3 &= x^2 + 10x + 10 \,, \\ r_4 &= 7x \,, \\ r_5 &= 10 \,. \end{aligned}$$

We get that the polynomials a and b are relatively prime in $\mathbb{Z}_p[x]$. The following theorem describes the connection between greatest common divisors in $\mathbb{Z}[x]$ and $\mathbb{Z}_p[x]$.

Theorem 5.8 *Let $a, b \in \mathbb{Z}[x], a, b \neq 0$. Let p be a prime for which $p \nmid lc(a)$ and $p \nmid lc(b)$. Let further $c = gcd(a, b) \in \mathbb{Z}[x]$, $a_p = a\,rem\,p$, $b_p = b\,rem\,p$ and $c_p = c\,rem\,p$. Then*

(1) $\deg(gcd(a_p, b_p)) \geq \deg(gcd(a, b))$,

(2) if $p \nmid res(a/c, b/c)$, then $gcd(a_p, b_p) = c_p$.

Proof. (1): Since $c_p \mid a_p$ and $c_p \mid b_p$, thus $c_p \mid \gcd(a_p, b_p)$. So

$$\deg(\gcd(a_p, b_p)) \geq \deg(\gcd(a, b) \bmod p).$$

By hypothesis $p \nmid \mathrm{lc}(\gcd(a, b))$, which implies

$$\deg(\gcd(a, b) \bmod p) = \deg(\gcd(a, b)) \,.$$

(2): Since $\gcd(a/c, b/c) = 1$, and c_p is non-trivial, so

$$\gcd(a_p, b_p) = c_p \cdot \gcd(a_p/c_p, b_p/c_p) \,. \tag{5.13}$$

If $\gcd(a_p, b_p) \neq c_p$ then the right hand side of (5.13) is non-trivial, thus $\mathrm{res}(a_p/c_p, b_p/c_p) = 0$. But the resultant is the sum of the corresponding products of the coefficients, so $p \mid \mathrm{res}(a/c, b/c)$, a contradiction. ∎

Corollary 5.9 *There are at most a finite number of primes p for which $p \nmid lc(a)$, $p \nmid lc(b)$ and* $\deg(gcd(a_p, b_p)) > \deg(gcd(a, b))$. ∎

In the case when in statement (1) of theorem 5.8 is fulfilled we call p a „lucky prime". We can outline a modular algorithm for gcd computation.

Modular-Gcd-Bigprime(a, b)

```
1  M ← the Landau-Mignotte constant (from corollary 5.4)
2  H ← {}
3  while TRUE
4        do p ← a prime with p ≥ 2M, p ∉ H, p ∤ lc(a) and p ∤ lc(b)
5           c_p ← gcd(a_p, b_p)
6           if c_p | a and c_p | b
7              then return c_p
8              else H ← H ∪ {p}
```

The first line of the algorithm requires the calculation of the Landau-Mignotte bound. The fourth line requires a „sufficiently large" prime p which does not divide the leading coefficient of a or b. The fifth line computes the greatest common divisor of the polynomials a and b modulo p (for example with the CLASSICAL-EUCLIDEAN algorithm in $\mathbb{Z}_p[x]$). We store the coefficients of the resulting polynomials with symmetrical representation. The sixth line examines whether $c_p \mid a$ and $c_p \mid b$ are fulfilled, in which case c_p is the required greatest common divisor. If this is not the case then p is an „unlucky prime", so we choose another prime. Since by theorem 5.8 there are only finitely many „unlucky primes", the algorithm will always terminate sooner or later. If the primes are chosen according to a given strategy, the set H is not needed.

The disadvantage of the MODULAR-GCD-BIGPRIME algorithm is that the Landau-Mignotte constant grows exponentially in terms of the degree of the input polynomials, so we have to work with large primes. The question is how we could modify the algorithm so that we can rather work with „many small primes". Since greatest common divisor in $\mathbb{Z}_p[x]$ is unique only up to a constant factor, we have to be careful with the coefficients of the polynomials in the new algorithm. So before applying the Chinese remainder theorem for the coefficients of the modular greatest common divisors taken modulo different primes, we have to normalise the leading coefficient of $\gcd(a_p, b_p)$. If a_m and b_n are the leading coefficients of a and b then the leading coefficient of $\gcd(a, b)$ divides $\gcd(a_m, b_n)$. Therefore, we normalise the leading coefficient of $\gcd(a_p, b_p)$ to $\gcd(a_m, b_n) \bmod p$ in the case of primitive polynomials a and b; and finally we take the primitive part of the resulting polynomial. Just like in the MODULAR-GCD-BIGPRIME algorithm, modular values are stored with symmetrical representation. These observations lead to the following modular gcd algorithm using small primes.

MODULAR-GCD-SMALLPRIMES(a, b)

1 $d \leftarrow \gcd(\operatorname{lc}(a), \operatorname{lc}(b))$
2 $p \leftarrow$ a prime for which $p \nmid d$
3 $H \leftarrow \{p\}$

```
 4  P ← p
 5  c_p ← gcd(a_p, b_p)
 6  g_p ← (d mod p) · c_p
 7  (n, i, j) ← (3, 1, 1)
 8  while TRUE
 9        do if j = 1
10              then if deg g_p = 0
11                        then return 1
12              (g, j, P) ← (g_p, 0, p)
13           if n ≤ i
14              then g ← pp(g)
15                   if g | a and g | b
16                      then return g
17           p ← a prime for which p ∤ d and p ∉ H
18           H ← H ∪ {p}
19           c_p ← gcd(a_p, b_p)
20           g_p ← (d mod p) · c_p
21           if deg g_p < deg g
22              then (i, j) ← (1, 1)
23           if j = 0
24              then if deg g_p = deg g
25                        then g_1 = Coeff-build(g, g_p, P, p)
26                             if g_1 = g
27                                then i ← i + 1
28                                else i ← 1
29                             P ← P · p
30                             g ← g_1
```

Coeff-Build(a, b, m_1, m_2)

```
1  p ← 0
2  c ← 1/m_1 mod m_2
3  for i ← deg a downto 0
4      do r ← a_i mod m_1
5         s ← (b_i − r) mod m_2
6         p ← p + (r + s · m_1)x^i
7  return p
```

We may note that the algorithm Modular-Gcd-Smallprimes does not require as many small primes as the Landau-Mignotte bound tells us. When the value of the polynomial g does not change during a few iterations, we test in lines 13–16 if g is really a greatest common divisor. The number of these iterations is stored in the variable n of line six. We note that the value of n could vary according to the input polynomial. The primes used in the algorithms could preferably be chosen from a (architecture-dependent) prestored list containing primes that fit in a machine word, so the use of the set H becomes unnecessary. Corollary 5.9

implies that the MODULAR-GCD-SMALLPRIMES algorithm always comes to an end.

The COEFF-BUILD algorithm computes the solution of the equation system obtained by taking congruence relations modulo m_1 and m_2 for the coefficients of identical degree in the input polynomials a and b. This is done according to the Chinese remainder theorem. It is very important to store the results in symmetrical modular representation form.

Example 5.9 Let us examine the operation of the MODULAR-GCD-SMALLPRIMES algorithm for the previously seen polynomials (5.4), (5.5). For comprehensivity we calculate with small primes. Recall that

$$\begin{aligned} a(x) &= 63x^5 + 57x^4 - 59x^3 + 45x^2 - 8 \in \mathbb{Z}[x] , \\ b(x) &= -77x^4 + 66x^3 + 54x^2 - 5x + 99 \in \mathbb{Z}[x] . \end{aligned}$$

After the execution of the first six lines of the algorithm with $p = 5$ we have $d = 7, c_p = x^2 + 3x + 2$ and $g_p = 2x^2 + x - 1$. Since $j = 1$ due to line 7, the lines 10–12 are executed. The polynomial g_p is not zero so $g = 2x^2 + x - 1$, $j = 0$, and $P = 5$ will be the values after execution. The condition in line 13 is not fulfilled so we choose another prime. The prime $p = 7$ is a bad choice but $p = 11$ is allowed. According to lines 19–20 then $c_p = 1$, $g_p = -4$. Since $\deg g_p < \deg g$, we have $j = 1$ and lines 25–30 are not executed. The polynomial g_p is constant so the return value in line 11 is 1, which means that the polynomials a and b are relatively prime.

Example 5.10 In our second example consider the already discussed polynomials

$$\begin{aligned} a(x) &= 12x^4 - 68x^3 + 52x^2 - 92x + 56 \in \mathbb{Z}[x] , \\ b(x) &= -12x^3 + 80x^2 - 84x + 24 \in \mathbb{Z}[x] . \end{aligned}$$

Let again $p = 5$. After the first six lines of the polynomials $d = 12, c_p = x + 1, g_p = 2x + 2$. After the execution of lines 10–12 we have $P = 5, g = 2x + 2$. Let the next prime be $p = 7$. So the new values are $c_p = x + 4, g_p = -2x - 1$. Since $\deg g_p = \deg g$, after lines 25–30 $P = 35$ and the new value of g is $12x - 8$. The value of the variable i is still 1. Let the next prime be 11. Then $c_p = g_p = x + 3$. The polynomials g_p and g have the same degree, so we modify the coefficients of g. Then $g_1 = 12x - 8$ and since $g = g_1$, we get $i = 2$ and $P = 385$. Let the new prime be 13. Then $c_p = x + 8, g_p = -x + 5$. The degrees of g_p and g are still equal thus lines 25–30 are executed and the variables become $g = 12x - 8, P = 4654, i = 3$.

After the execution of lines 17–18 it turns out that $g \mid a$ and $g \mid b$ so $g = 12x - 8$ is the greatest common divisor.

We give the following theorem without proof.

Theorem 5.10 *The* MODULAR-GCD-SMALLPRIMES *algorithm works correctly. The computational complexity of the algorithm is* $O(m^3(\lg m + \lambda(K))^2)$ *in machine word operations, where* $m = \min\{\deg a, \deg b\}$*, and* K *is the Landau-Mignotte bound for the polynomials* a *and* b*.*

Exercises

5.2-1 Let R be a commutative ring with identity element, $a = \sum_{i=0}^{m} a_i x^i \in R[x]$, $b = \sum_{i=0}^{n} b_i x^i \in R[x]$, furthermore b_n a unit, $m \geq n \geq 0$. The following algorithm performs Euclidean division for a and b and outputs the polynomials $q, r \in R[x]$ for which $a = qb + r$ and $\deg r < n$ or $r = 0$ holds.

EUCLIDEAN-DIVISION-UNIVARIATE-POLYNOMIALS(a, b)

```
1  r ← a
2  for i ← m − n downto 0
3      do if deg r = n + i
4           then q_i ← lc(r)/b_n
5                r ← r − q_i x^i b
6           else q_i ← 0
7  q ← Σ_{i=0}^{m−n} q_i x^i and r
8  return q
```

Prove that the algorithm uses at most

$$(2 \deg b + 1)(\deg q + 1) = O(m^2)$$

operations in R.

5.2-2 What is the difference between the algorithms EXTENDED-EUCLIDEAN and EXTENDED-EUCLIDEAN-NORMALISED in $\mathbb{Z}[x]$?

5.2-3 Prove that $\operatorname{res}(f \cdot g, h) = \operatorname{res}(f, h) \cdot \operatorname{res}(g, h)$.

5.2-4 The ***discriminant*** of the polynomial $f(x) \in R[x]$ $(\deg f = m, \operatorname{lc}(f) = f_m)$ is the element

$$\operatorname{discr} f = \frac{(-1)^{\frac{m(m-1)}{2}}}{f_m} \operatorname{res}(f, f') \in R$$

where f' denotes the derivative of f with respect to x. The polynomial f has a multiple root if and only if its discriminant is 0. Compute $(\operatorname{discr} f)$ for general polynomials of second and third degree.

5.3. Gröbner basis

Let F be a field and $R = F[x_1, x_2, \ldots, x_n]$ be a multivariate polynomial ring in n variables over F. Let $f_1, f_2, \ldots, f_s \in R$. First we determine a necessary and sufficient condition having the polynomials $f_1, f_2, \ldots, f_s$ common roots in R. We can see that the problem is a kind of generalisation of the case $s = 2$ from the previous subsection. Let

$$I = \langle f_1, \ldots, f_s \rangle = \left\{ \sum_{1 \le i \le s} q_i f_i : q_i \in R \right\}$$

denote the ideal ***generated*** by the polynomials $f_1, \ldots, f_s$. Then the polynomials $f_1, \ldots, f_s$ form a ***basis*** of the ideal I. The ***variety*** of an ideal I is the set

$$V(I) = \left\{ u \in F^n : f(u) = 0 \text{ for all } f \in I \right\}.$$

The knowledge of the variety $V(I)$ means that we also know the common roots of $f_1, \ldots, f_s$. The most important questions about the variety and the ideal I are as follows.

- $V(I) \neq \emptyset$?

- How „big" is $V(I)$?
- Given $f \in R$, in which case $f \in I$?
- $I = R$?

Fortunately, in a special basis of the ideal I, in the so called Gröbner basis these questions are easy to answer. First let us study the case $n = 1$. Since $F[x]$ is a Euclidean ring, therefore

$$\langle f_1, \dots, f_s \rangle = \langle \gcd(f_1, \dots, f_s) \rangle \ . \tag{5.14}$$

We may assume that $s = 2$. Let $f, g \in F[x]$ and divide f by g with remainder. Then there exist unique polynomials $q, r \in F[x]$ with $f = gq + r$ and $\deg r < \deg g$. Hence

$$f \in \langle g \rangle \Leftrightarrow r = 0 \ .$$

Moreover, $V(g) = \{u_1, \dots, u_d\}$ if $x - u_1, \dots, x - u_d$ are the distinct linear factors of $g \in F[x]$. Unfortunately, the equality (5.14) is not true in the case of two or more variables. Indeed, a multivariate polynomial ring over an arbitrary field is not necessary Euclidean, therefore we have to give a new interpretation of division with remainder. We proceed in this direction.

5.3.1. Monomial order

Recall that a partial order $\rho \subseteq S \times S$ is a total order (or simply order) if either $a \rho b$ or $b \rho a$ for all $a, b \in S$. The total order '$\preceq$' $\subseteq \mathbb{N}^n$ is ***allowable*** if

(i) $(0, \dots, 0) \preceq v$ for all $v \in \mathbb{N}^n$,
(ii) $v_1 \preceq v_2 \Rightarrow v_1 + v \preceq v_2 + v$ for all $v_1, v_2, v \in \mathbb{N}^n$.

It is easy to prove that any allowable order on $\mathbb{N}^n$ is a well-order (namely, every nonempty subset of $\mathbb{N}^n$ has a least element). With the notation already adopted consider the set

$$T = \{x_1^{i_1} \cdots x_n^{i_n} \mid i_1, \dots, i_n \in \mathbb{N}\}.$$

The elements of T are called ***monomials***. Observe that T is closed under multiplication on $F[x_1, \dots, x_n]$ constituting a commutative monoid. The map $\mathbb{N}^n \to T$, $(i_1, \dots, i_n) \mapsto x_1^{i_1} \cdots x_n^{i_n}$ is an isomorphism, therefore for an allowable total order $\preceq$ on T we have that

(i) $1 \preceq t$ for all $t \in T$,
(ii) $\forall\ t_1, t_2, t \in T \ \ t_1 \prec t_2 \Rightarrow t_1 t \prec t_2 t$.

The allowable orders on T are called ***monomial orders***. If $n = 1$, the natural order is a monomial order, and the corresponding univariate monomials are ordered by their degree. Let us see some standard examples of higher degree monomial orders. Let

$$\alpha = x_1^{i_1} \cdots x_n^{i_n},\ \beta = x_1^{j_1} \cdots x_n^{j_n} \in T \ ,$$

where the variables are ordered as $x_1 \succ x_2 \succ \cdots \succ x_{n-1} \succ x_n$.

- ***Pure lexicographic order.***
  $\alpha \prec_{plex} \beta \Leftrightarrow \exists l \in \{1, \dots, n\}\ i_l < j_l$ and $i_1 = j_1, \dots, i_{l-1} = j_{l-1}$.

- ***Graded lexicographic order.***
  $\alpha \prec_{grlex} \beta \Leftrightarrow i_1 + \cdots + i_n < j_1 + \cdots + j_n$ or $(i_1 + \cdots + i_n = j_1 + \cdots + j_n$ and $\alpha \prec_{plex} \beta)$.
- ***Graded reverse lexicographic order.***
  $\alpha \prec_{grevlex} \beta \Leftrightarrow i_1 + \cdots + i_n < j_1 + \cdots + j_n$ or $(i_1 + \cdots + i_n = j_1 + \cdots + j_n$ and $\exists l \in \{1, \ldots, n\}$ $i_l > j_l$ and $i_{l+1} = j_{l+1}, \ldots, i_n = j_n)$.

To prove that these orders are really monomial orders we leave as an exercise. Observe that if $n = 1$ then $\prec_{plex} = \prec_{grlex} = \prec_{grevlex}$. We call the graded reverse lexicographic order often as a total degree order and we denote it by $\prec_{tdeg}$.

Example 5.11
Let $\prec = \prec_{plex}$ and let $z \prec y \prec x$. Then

$$\begin{aligned} 1 &\prec z \prec z^2 \prec \cdots \prec y \prec yz \prec yz^2 \prec \cdots \\ &\prec y^2 \prec y^2z \prec y^2z^2 \prec \cdots x \prec xz \prec xz^2 \prec \cdots \\ &\prec xy \prec xy^2 \prec \cdots \prec x^2 \prec \cdots \ . \end{aligned}$$

Let $\prec = \prec_{tdeg}$ and again, $z \prec y \prec x$. Then

$$\begin{aligned} 1 &\prec z \prec y \prec x \\ &\prec z^2 \prec yz \prec xz \prec y^2 \prec xy \prec x^2 \\ &\prec z^3 \prec yz^2 \prec xz^2 \prec y^2z \prec xyz \\ &\prec x^2z \prec y^3 \prec xy^2 \prec x^2y \prec x^3 \prec \cdots \ . \end{aligned}$$

Let a monomial order $\prec$ be given. Further we identify the vector $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$ with the monomial $x^\alpha = x_1^{\alpha_1} \cdots x_n^{\alpha_n} \in R$. Let $f = \sum_{\alpha \in \mathbb{N}^n} c_\alpha x^\alpha \in R$ be a non-zero polynomial, $c_\alpha \in F$. Then $c_\alpha x^\alpha$ $(c_\alpha \neq 0)$ are the ***terms*** of the polynomial f, $\mathrm{mdeg}(f) = \max\{\alpha \in \mathbb{N}^n : c_\alpha \neq 0\}$ is the ***multidegree*** of the polynomial (where the maximum is with respect to the monomial order), $\mathrm{lc}(f) = c_{\mathrm{mdeg}(f)} \in F \setminus \{0\}$ is the ***leading coefficient*** of f, $\mathrm{lm}(f) = x^{\mathrm{mdeg}(f)} \in R$ is the ***leading monomial*** of f, and $\mathrm{lt}(f) = \mathrm{lc}(f) \cdot \mathrm{lm}(f) \in R$ is the ***leading term*** of f. Let $\mathrm{lt}(0) = \mathrm{lc}(0) = \mathrm{lm}(0) = 0$ and $\mathrm{mdeg}(0) = -\infty$.

Example 5.12 Consider the polynomial $f(x, y, z) = 2xyz^2 - 3x^3 + 4y^4 - 5xy^2z \in \mathbb{Q}[x, y, z]$. Let $\prec = \prec_{plex}$ and $z \prec y \prec x$. Then

$$\mathrm{mdeg}(f) = (3, 0, 0),\ \mathrm{lt}(f) = -3x^3,\ \mathrm{lm}(f) = x^3,\ \mathrm{lc}(f) = -3 \ .$$

If $\prec = \prec_{tdeg}$ and $z \prec y \prec x$ then

$$\mathrm{mdeg}(f) = (0, 4, 0),\ \mathrm{lt}(f) = 4y^4,\ \mathrm{lm}(f) = y^4,\ \mathrm{lc}(f) = 4 \ .$$

5.3.2. Multivariate division with remainder

In this subsection our aim is to give an algorithm for division with remainder in R. Given multivariate polynomials $f, f_1, \ldots, f_s \in R$ and monomial order $\prec$ we want to compute the polynomials $q_1, \ldots, q_s \in R$ and $r \in R$ such that $f = q_1 f_1 + \cdots + q_s f_s + r$ and no monomial in r is divisible by any of $\mathrm{lt}(f_1), \ldots, \mathrm{lt}(f_s)$.

MULTIVARIATE-DIVISION-WITH-REMAINDER$(f, f_1, \dots, f_s)$

```
1  r ← 0
2  p ← f
3  for i ← 1 to s
4      do q_i ← 0
5  while p ≠ 0
6        do if lt(f_i) divides lt(p) for some i ∈ {1, ..., s}
7              then choose such an i and q_i ← q_i + lt(p)/lt · (f_i)
8                   p ← p − lt(p)/lt(f_i) · f_i
9              else r ← r + lt(p)
10                  p ← p − lt(p)
11 return q_1, ..., q_s and r
```

The correctness of the algorithm follows from the fact that in every iteration of the **while** cycle of lines 5–10 the following invariants hold:

(i) $\mathrm{mdeg}(p) \le \mathrm{mdeg}(f)$ and $f = p + q_1 f_1 + \cdots + q_s f_s + r$,

(ii) $q_i \neq 0 \Rightarrow \mathrm{mdeg}(q_i f_i) \le \mathrm{mdeg}(f)$ for all $1 \le i \le s$,

(iii) no monomial in r is divisible by any of $\mathrm{lt}(f_i)$.

The algorithm has a weakness, namely the multivariate division with remainder is not deterministic. In line 7 we can choose arbitrarily from the appropriate values of i.

Example 5.13 Let $f = x^2y + xy^2 + y^2 \in \mathbb{Q}[x, y]$, $f_1 = xy - 1$, $f_2 = y^2 - 1$, the monomial order $\prec_{plex}$, $y \prec_{plex} x$, and in line 7 we always choose the smallest possible i. Then the result of the algorithm is $q_1 = x + y$, $q_2 = 1$, $r = x + y + 1$. But if change the order of the functions f_1 and f_2 (that is $f_1 = y^2 - 1$ and $f_2 = xy - 1$) then the output of the algorithm is $q_1 = x + 1$, $q_2 = x$ and $r = 2x + 1$.

As we have seen in the previous example we can make the algorithm deterministic by always choosing the smallest possible i in line 7. In this case the ***quotients*** $q_1, \dots, q_s$ and the ***remainder*** r are unique, which we can express as $r = f \text{ rem } (f_1, \dots, f_s)$.

Observe that if $s = 1$ then the algorithm gives the answer to the ideal membership problem: $f \in \langle f_1 \rangle$ if and only if the remainder is zero. Unfortunately, if $s \ge 2$ then this is not true any more. For example, with the monomial order $\prec_{plex}$

$$xy^2 - x \text{ rem } (xy + 1, y^2 - 1) = -x - y,$$

and the quotients are $q_1 = y$, $q_2 = 0$. On the other hand, $xy^2 - x = x \cdot (y^2 - 1) + 0$, which shows that $xy^2 - x \in \langle xy + 1, y^2 - 1 \rangle$.

5.3.3. Monomial ideals and Hilbert's basis theorem

Our next goal is to find a special basis for an arbitrary polynomial ideal such that the remainder on division by that basis is unique, which gives the answer to the ideal membership problem. But such a basis exists at all? And if exists, is it finite?

The ideal $I \subseteq R$ is a ***monomial ideal*** if there exists a subset $A \subseteq \mathbb{N}^n$, for which

$$I = \langle x^A \rangle = \langle \{x^\alpha \in T : \alpha \in A\} \rangle ,$$

that is, the ideal I is generated by monomials.

Lemma 5.11 *Let $I = \langle x^A \rangle \subseteq R$ be a monomial ideal, and $\beta \in \mathbb{N}^n$. Then*

$$x^\beta \in I \Leftrightarrow \exists \alpha \in A \quad x^\alpha \mid x^\beta \ .$$

Proof. The $\Leftarrow$ direction is obvious. Conversely, let $\alpha_1, \ldots, \alpha_s \in A$ and $q_1, \ldots, q_s \in R$ such that $x^\beta = \sum_i q_i x^{\alpha_i}$. Then the sum has at least one member $q_i x^{\alpha_i}$ which contains x^β, therefore $x^{\alpha_i} \mid x^\beta$. ∎

The most important corollary of the lemma is that two monomial ideals are identical if and only if they contain the same monomials.

Lemma 5.12 (Dickson-lemma). *Every monomial ideal is finitely generated, namely for every $A \subseteq \mathbb{N}^n$ there exists a finite subset $B \subseteq A$ such that $\langle x^A \rangle = \langle x^B \rangle$.*

Lemma 5.13 *Let I be an ideal in $R = F[x_1, \ldots, x_n]$. If $G \subseteq I$ is a finite subset such that $\langle lt(G) \rangle = \langle lt(I) \rangle$, then $\langle G \rangle = I$.*

Proof. Let $G = \{g_1, \ldots, g_s\}$. If $f \in I$ is an arbitrary polynomial, then division with remainder gives $f = q_1 g_1 + \cdots + q_s g_s + r$, with $q_1, \ldots, q_s, r \in R$, such that either $r = 0$ or no term of r is divisible by the leading term of any g_i. But $r = f - q_1 g_1 - \cdots - q_s g_s \in I$, hence $\mathrm{lt}(r) \in \mathrm{lt}(I) \subseteq \langle \mathrm{lt}(g_1), \ldots, \mathrm{lt}(g_s) \rangle$. This together with lemma (5.11) implies that $r = 0$, therefore $f \in \langle g_1, \ldots, g_s \rangle = \langle G \rangle$. ∎

Together with Dickson's lemma applied to $\langle \mathrm{lt}(I) \rangle$, and the fact that the zero polynomial generates the zero ideal, we obtain the following famous result.

Theorem 5.14 (Hilbert's basis theorem). *Every ideal $I \subseteq R = F[x_1, \ldots, x_n]$ is finitely generated, namely there exists a finite subset $G \subseteq I$ such that $\langle G \rangle = I$ and $\langle lt(G) \rangle = \langle lt(I) \rangle$.*

Corollary 5.15 (ascending chain condition). *Let $I_1 \subseteq I_2 \subseteq \cdots$ be an ascending chain of ideals in R. Then there exists an $n \in \mathbb{N}$ such that $I_n = I_{n+1} = \cdots$.*

Proof. Let $I = \cup_{j \geq 1} I_j$. Then I is an ideal, which is finitely generated by Hilbert's basis theorem. Let $I = \langle g_1, \ldots, g_s \rangle$. With $n = \min\{j \geq 1 : g_1, \ldots, g_s \in I_j\}$, we then have $I_n = I_{n+1} = \cdots = I$. ∎

A ring satisfying the ascending chain condition is called ***Noetherian***. Specifically, if F is a field, then $F[x_1, \ldots, x_n]$ is Noetherian.

Let $\prec$ be a monomial order on R and $I \subseteq R$ an ideal. A finite set $G \subseteq I$ is a ***Gröbner basis*** for the ideal I with respect to $\prec$ if $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$. Hilbert's basis theorem implies the following

Corollary 5.16 *Every ideal I in $R = F[x_1, \ldots, x_n]$ has a Gröbner basis.*

It is easy to show that the remainder on division by Gröbner basis G does not depend on the order of the elements of G. Therefore we can use the notation $f \text{ rem } G = r \in R$. Using the Gröbner basis we can easily answer the ideal membership problem.

Theorem 5.17 *Let G be a Gröbner basis for the ideal $I \subseteq R$ with respect to a monomial order $\prec$ and let $f \in R$. Then $f \in I \Leftrightarrow f \text{ rem } G = 0$.*

Proof. We prove that there exists a unique $r \in R$ such that (1) $f - r \in I$, (2) no term of r is divisible by any monomial of $\mathrm{lt}(G)$. The existence of such an r comes from the division with remainder. For the uniqueness let $f = h_1 + r_1 = h_2 + r_2$ for arbitrary $h_1, h_2 \in I$ and no term of r_1 or r_2 is divisible by any monomial of $\mathrm{lt}(G)$. Then $r_1 - r_2 = h_2 - h_1 \in I$, and by the lemma 5.11 $\mathrm{lt}(r_1 - r_2)$ is divisible by some $\mathrm{lt}(g)$, $g \in G$. It means that $r_1 - r_2 = 0$. ■

Thus, if G is a Gröbner basis of R, then for all $f, g, h \in R$

$$g = f \text{ rem } G \text{ and } h = f \text{ rem } G \Rightarrow g = h.$$

5.3.4. Buchberger's algorithm

Unfortunately, the Hilbert's basis theorem is not constructive, since it does not tell us how to compute a Gröbner basis for an ideal I and basis G. In the following we investigate how the finite set G can fail to be a Gröbner basis for an ideal I.

Let $g, h \in R$ be nonzero polynomials, $\alpha = (\alpha_1, \ldots, \alpha_n) = \mathrm{mdeg}(g)$, $\beta = (\beta_1, \ldots, \beta_n) = \mathrm{mdeg}(h)$, and $\gamma = (\max\{\alpha_1, \beta_1\}, \ldots, \max\{\alpha_n, \beta_n\})$. The ***S-polynomial*** of g and h is

$$S(g,h) = \frac{x^\gamma}{\mathrm{lt}(g)} g - \frac{x^\gamma}{\mathrm{lt}(h)} h \in R.$$

It is easy to see that $S(g,h) = -S(h,g)$, moreover, since $x^\gamma/\mathrm{lt}(g), x^\gamma/\mathrm{lt}(h) \in R$, therefore $S(g,h) \in \langle g, h \rangle$.

The following theorem yields an easy method to test that a given set G is a Gröbner basis of the ideal $\langle G \rangle$.

Theorem 5.18 *The set $G = \{g_1, \ldots, g_s\} \subseteq R$ is the Gröbner basis if the ideal $\langle G \rangle$ if and only if*

$$S(g_i, g_j) \text{ rem } (g_1, \ldots, g_s) = 0 \text{ for all } i \ (1 \le i < j \le s) .$$

Using the S-polynomials it is easy to give an algorithm for constructing the Gröbner basis. We present a simplified version of the Buchberger's method (1965): given monomial order $\prec$ and polynomials $f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$ the algorithm yields a $G \subseteq R$ Gröbner basis of the ideal $I = \langle f_1, \ldots, f_s \rangle$.

GRÖBNER-BASIS$(f_1, \ldots, f_s)$

```
1  G ← {f_1, ..., f_s}
2  P ← {(f_i, f_j) | f_i, f_j ∈ G, i < j, f_i ≠ f_j}
3  while P ≠ ∅
4        do (f, g) ← an arbitrary pair from P
5           P ← P \ (f, g)
6           r ← S(f, g) rem G
7           if r ≠ 0
8              then G ← G ∪ {r}
9                   P ← P ∪ {(f, r) | f ∈ G}
10 return G
```

First we show the correctness of the GRÖBNER-BASIS algorithm assuming that the procedure terminates. At any stage of the algorithm the set G is a basis of the ideal I, since initially it is, and at any other steps only those elements are added to G which are the remainders of the S-polynomials on division by G. If the algorithm terminates the remainders of all the S-polynomials on division by G are zero and by the theorem (5.18) G is a Gröbner basis.

Next we show that the algorithm terminates. Let G and G^* be the sets corresponding to successive iterations of the **while** cycle $3-9$. Clearly, $G \subseteq G^*$ and $\langle \mathrm{lt}(G)\rangle \subseteq \langle \mathrm{lc}(G^*)\rangle$. Hence the ideals $\langle \mathrm{lt}(G)\rangle$ in successive iteration steps form an ascending chain, which stabilizes by the corollary (5.15). Thus, after a finite number of steps we have that $\langle \mathrm{lt}(G)\rangle = \langle \mathrm{lc}(G^*)\rangle$. We state that then $G = G^*$. Let $f, g \in G$ and $r = S(f,g)$ rem G. Then $r \in G^*$ and either $r = 0$ or $\mathrm{lt}(r) \in \langle \mathrm{lt}(G^*)\rangle = \langle \mathrm{lt}(G)\rangle$, and from the definition of the remainder we conclude that $r = 0$.

Example 5.14 Let $F = \mathbb{Q}$, $<=<_{plex}$, $z < y < x$, $f_1 = x - y - z$, $f_2 = x + y - z^2$, $f_3 = x^2 + y^2 - 1$.

By step one $G = \{f_1, f_2, f_3\}$, and by step two $P = \{(f_1, f_2), (f_1, f_3), (f_2, f_3)\}$.

At the first iteration of the **while** cycle let us chose the pair (f_1, f_2). Then $P = \{(f_1, f_3), (f_2, f_3)\}$, $S(f_1, f_2) = -2y - z + z^2$ and $r = f_4 = S(f_1, f_2)$ rem $G = -2y - z + z^2$. Therefore $G = \{f_1, f_2, f_3, f_4\}$ and $P = \{(f_1, f_3), (f_2, f_3), (f_1, f_4), (f_2, f_4), (f_3, f_4)\}$.

At the second iteration of the **while** cycle let us chose the pair (f_1, f_3). Then $P = P \setminus \{f_1, f_3\}$, $S(f_1, f_3) = -xy - xz - y^2 + 1$, $r = f_5 = S(f_1, f_3)$ rem $G = -1/2z^4 - 1/2z^2 + 1$, hence $G = \{f_i \mid 1 \le i \le 5\}$ and $P = \{(f_2, f_3), (f_1, f_4), \ldots, (f_3, f_4), (f_1, f_5), \ldots, (f_4, f_5)\}$.

At the third iteration of the **while** cycle let us chose the pair (f_2, f_3). Then $P = P \setminus \{(f_2, f_3)\}$, $S(f_2, f_3) = xy - xz^2 - y^2 + 1$, $r = S(f_2, f_3)$ rem $G = 0$.

At the fourth iteration let us chose the pair (f_1, f_4). Then $P = P \setminus \{f_1, f_4\}$, $S(f_1, f_4) = 2y^2 + 2yz + xz - xz^2$, $r = S(f_1, f_4)$ rem $G = 0$.

In the same way, the remainders of the S-polynomials on division by G of all the remaining pairs are zero, hence the algorithm returns with $G = \{x - y - z, x + y - z^2, x^2 + y^2 - 1, -2y - z + z^2, -1/2z^4 - 1/2z^2 + 1\}$ which constitutes a Gröbner basis.

5.3.5. Reduced Gröbner basis

In general, the Gröbner basis computed by Buchberger's algorithm is neither unique nor minimal. Fortunately, both can be achieved by a little finesse.

Lemma 5.19 *If G is a Gröbner basis if $I \subseteq R$, $g \in G$ and $lt(g) \in \langle lt(G \setminus \{g\})\rangle$ then $G \setminus \{g\}$ is a Gröbner basis of I as well.*

We say that the set $G \subseteq R$ is a ***minimal*** Gröbner basis for the ideal $I = \langle G\rangle$ if it is a Gröbner basis and for all $g \in G$

- $\mathrm{lc}(g) = 1$,
- $\mathrm{lt}(g) \notin \langle \mathrm{lt}(G \setminus \{g\})\rangle$.

An element $g \in G$ of a Gröbner basis G is said to be ***reduced*** with respect to G if no monomial of g is in the ideal $\langle \mathrm{lt}(G \setminus \{g\})\rangle$. A minimal Gröbner basis G for $I \subseteq R$ is reduced if all its elements are reduced with respect to G.

Theorem 5.20 *Every ideal has a unique reduced Gröbner basis.*

Example 5.15 In example 5.14. not only G but $G' = \{x - y - z, -2y - z + z^2, -1/2z^4 - 1/2z^2 + 1\}$ is a Gröbner basis as well. It is not hard to show that $G_r = \{x - 1/2z^2 - 1/2z, y - 1/2z^2 - 1/2z, z^4 + z^2 - z\}$ is a reduced Gröbner basis.

5.3.6. The complexity of computing Gröbner bases

The last forty years (from Buchberger's dissertation) was not enough to clear up entirely the algorithmic complexity of Gröbner basis computation. Implementation experiments show that we are faced with the intermediate expression swell phenomenon. Starting with a few polynomials of low degree and small coefficients the algorithm produce large number of polynomials with huge degrees and enormous coefficients. Moreover, in contrast to the variants of the Euclidean algorithm the explosion can not be kept under control. Kühnle and Mayr in 1996 gave an exponential space algorithm for computing a reduced Gröbner basis. The polynomial ideal membership problem over $\mathbb{Q}$ is *EXPSPACE*-complete.

Let $f, f_1, \ldots, f_s \in F[x_1, \ldots, x_n]$ be polynomials over a field F with $\deg f_i \leq d$ ($\prec = \prec_{ideg}$). If $f \in \langle f_1, f_2, \ldots f_s \rangle$ then

$$f = f_1 g_1 + \cdots + f_s g_s$$

for polynomials $g_1, \ldots, g_s \in F[x_1, \ldots, x_n]$ for which their degrees are bounded by $\beta = \beta(n, d) = (2d)^{2^n}$. The double exponential bound essentially unavoidable which is shown by several examples. Unfortunately, for the case $F = \mathbb{Q}$ the ideal membership problem falls into this category. Fortunately, in special cases better results are available. If $f = 1$ (Hilbert's famous Nullstellensatz) then for the case $d = 2$ the bound is $\beta = 2^{n+1}$, while for the other cases $d > 2$ the bound is $\beta = d^n$. But the variety $V(f_1, \ldots, f_s)$ is empty if and only if $1 \in \langle f_1, f_2, \ldots f_s \rangle$, therefore the solvability problem of a polynomial system is in *PSPACE*. Several result states that under specific circumstances the (general) ideal membership problem is again in *PSPACE*. Such a criteria is for example that $\langle f_1, f_2, \ldots f_s \rangle$ being zero-dimensional (contains finitely many isolated points).

In spite of the exponential complexity there are many successful story for the application of Gröbner bases: geometric theorem proving, robot kinematics and motion planning, solving polynomial system of equations are the most widespread application areas. In the following we enumerate some topics where the Gröbner basis strategy has been applied successfully.

- *Equivalence of polynomial equations.* Two sets of polynomials generate the same ideal if and only if their Gröbner bases are equal with arbitrary monomial order.
- *Solvability of polynomial equations.* The polynomial system of equations $f_i(x_1, \ldots, x_n) = 0,\ 1 \leq i \leq s$ is solvable if and only if $1 \notin \langle f_1, \ldots, f_s \rangle$.
- *Finite solvability of polynomial equations.* The polynomial system of equations $f_i(x_1, \ldots, x_n) = 0,\ 1 \leq i \leq s$ has a finite number of solutions if and only if in any Gröbner basis of $\langle f_1, \ldots, f_s \rangle$ for every variable x_i there is a polynomial such that its leading term with respect to the chosen monomial order is a power of x_i.
- *Counting of finite solutions.* Suppose that the system of polynomial equations $f_i(x_1, \ldots, x_n) = 0,\ 1 \leq i \leq s$ has a finite number of solutions. Then the number of solutions counted with multiplicities is equal to the cardinality of the set of monomials that are no multiples of the leading monomials of the polynomials in the Gröbner basis,

where any monomial order can be chosen.

- *Simplification of expressions.*

We show an example for the last item.

Example 5.16 Let $a, b, c \in \mathbb{R}$ be given such that

$$a + b + c = 3, \;\; a^2 + b^2 + c^2 = 9, \;\; a^3 + b^3 + c^3 = 24 \; .$$

Simplify the expression $a^4 + b^4 + c^4$. So let $f_1 = a + b + c - 3$, $f_2 = a^2 + b^2 + c^2 - 9$ and $a^3 + b^3 + c^3 - 24$ be elements of $\mathbb{R}[a, b, c]$ and let $\prec = \prec_{plex}$, $c \prec b \prec a$. Then the Gröbner basis of $\langle f_1, f_2, f_3 \rangle$ is

$$G = \{a + b + c - 3, b^2 + c^2 - 3b - 3c + bc, 1 - 3c^2 + c^3\} \; .$$

Since $a^4 + b^4 + c^4 \operatorname{rem} G = 69$, the answer to the question is at hand.

Exercises

5.3-1 Prove that the orders $\prec_{plex}$, $\prec_{grlex}$ and $\prec_{tdeg}$ are really monomial orders.

5.3-2 Let $\prec$ be a monomial order on R, $f, g \in R \setminus \{0\}$. Prove the following:

a. $\operatorname{mdeg}(fg) = \operatorname{mdeg}(f) + \operatorname{mdeg}(g)$,

b. if $f + g \neq 0$ then $\operatorname{mdeg}(f + g) \preceq \max\{\operatorname{mdeg}(f), \operatorname{mdeg}(g)\}$, with equality if $\operatorname{mdeg}(f) \neq \operatorname{mdeg}(g)$.

5.3-3 Let $f = 2x^4y^2z - 3x^4yz^2 + 4xy^4z^2 - 5xy^2z^4 + 6x^2y^4z - 7x^2yz^4 \in \mathbb{Q}[x, y, z]$.

a. Determine the order of the monomials in f for the monomial orders $\prec_{plex}$, $\prec_{grlex}$ and $\prec_{tdeg}$ with $z \prec y \prec x$ in all cases.

b. For each of the three monomial orders from (*a.*) determine $\operatorname{mdeg}(f)$, $\operatorname{lc}(f)$, $\operatorname{lm}(f)$ and $\operatorname{lt}(f)$.

5.3-4⋆ Prove the Dickson's lemma.

5.3-5 Compute the Gröbner basis and the reduced Gröbner basis of the ideal $I = \langle x^2 + y - 1, xy - x \rangle \subseteq \mathbb{Q}[x, y]$ using the monomial order $\prec = \prec_{lex}$, where $y \prec x$. Which of the following polynomials belong to I: $f_1 = x^2 + y^2 - y$, $f_2 = 3xy^2 - 4xy + x + 1$?

5.4. Symbolic integration

The problem of indefinite integration is the following: given a function f, find a function g the derivative of which is f, that is, $g'(x) = f(x)$; for this relationship the $\int f(x)\,dx = g(x)$ notation is also used. In introductory calculus courses one tries to solve indefinite integration problems by different methods, among which one tries to choose in a heuristic way: substitution, trigonometric substitution, integration by parts, etc. Only the integration of rational functions is usually solved by an algorithmic method.

It can be shown that indefinite integration in the general case is algorithmically unsolvable. So we only have the possibility to look for a reasonably large part that can be solved using algorithms.

The first step is the algebraization of the problem: we discard every analytical concept and consider differentiation as a new (unary) algebraic operation connected to addition

and multiplication in a given way, and we try to find the „inverse" of this operation. This approach leads to the introduction of the concept of differential algebra.

The integration routines of computer algebra systems (e.g. MAPLE), similarly to us, first try a few heuristic methods. Integrals of polynomials (or a bit more generally, finite Laurent-series) are easy to determine. This is followed by a simple table lookup process (e.g. in the case of MAPLE 35 basic integrals are used). One can of course use integral tables entered from books as well. Next we may look for special cases where appropriate methods are known. For example, for integrands of the form

$$e^{ax+b}\sin(cx+d)\cdot p(x)\ ,$$

where p is a polynomial, the integral can be determined using integration by parts. When the above methods fail, a form of substitution called the „derivative-divides" method is tried: if the integrand is a composite expression then for every sub-expression $f(x)$ we divide by the derivative of f, and we check if x vanishes from the result after the substitution $u = f(x)$. With the use of these simple methods we can determine a surprisingly large number of integrals. To their great advantage they can solve simple problems very quickly. If they do not succeed we try algorithmic methods. The first one is for the integration of rational functions. As we shall see, there is significant difference between the version used in computer algebra systems and the version used in hand computations, since the aims are short running times even in complicated cases, and the simplest possible form of the result. The Risch algorithm for integrating elementary functions is based on the algorithm for the integration of rational functions. We describe the Risch algorithm, but not in full detail. In most cases we only outline the proofs.

5.4.1. Integration of Rational Functions

In this subsection we introduce the notion of differential field and differential extension field, then we describe Hermite's method.

Differential fields

Let K be a field of characteristic 0, with a mapping $f \mapsto f'$ of K into itself satisfying:

(1) $(f+g)' = f' + g'$ (additivity);

(2) $(fg)' = f'g + g'f$ (Leibniz-rule).

The mapping $f \mapsto f'$ is called a ***differential operator***, ***differentiation*** or ***derivation***, and K is called a ***differential field***. The set $C = \{c \in K : c' = 0\}$ is the ***field of constants*** or ***constant subfield*** in K. If $f' = g$ we also write $f = \int g$. Obviously, for any constant $c \in C$ we have $f + c = \int g$. The ***logarithmic derivative*** of an element $0 \neq f \in K$ is defined as f'/f. (Formally this is the „derivative of $\log(f)$".)

Theorem 5.21 *With the notations of the previous definition the usual rules of derivation hold:*

(1) $0' = 1' = (-1)' = 0$;

(2) *derivation is C-linear:* $(af+bg)' = af' + bg'$, *if* $f, g \in K$, $a, b \in C$;

(3) *if* $g \neq 0$, f *is arbitrary, then* $(f/g)' = (f'g - g'f)/g^2$;

(4) $(f^n)' = nf'f^{n-1}$, *if* $0 \neq f \in K$ *and* $n \in \mathbb{Z}$;

(5) $\int fg' = fg - \int gf'$, *if* $f, g \in K$ (***integration by parts***).

Example 5.17 (1) With the notations of the previous definition the mapping $f \mapsto 0$ on K is the *trivial derivation*, for this we have $C = K$.

(2) Let $K = \mathbb{Q}(x)$. There exists a single differential operator on $\mathbb{Q}(x)$ with $x' = 1$, it is the usual differentiation. For this the constants are the elements of $\mathbb{Q}$. Indeed, $n' = 0$ for $n \in \mathbb{N}$ by induction so the elements of $\mathbb{Z}$ and $\mathbb{Q}$ are constants, too. We have by induction that the derivative of power functions is the usual one, thus by linearity, this is true for polynomials, and by the differentiation rule of quotients, we get the statement. It is not difficult to calculate that for the usual differentiation the constants are the elements of $\mathbb{Q}$.

(3) If $K = C(x)$, where C is an arbitrary field of characteristic 0, then there exists a single differential operator on K with constant subfield C and $x' = 1$, it is the usual differentiation; this statement is obtained similarly to the previous one.

If C is an arbitrary field of characteristic 0, and $K = C(x)$ with the usual differentiation then $1/x$ is not the derivative of anything. (The proof of the statement is very much like the proof of the irrationality of $\sqrt{2}$, but we have to work with divisibility by x rather than by 2.)

The example shows that for the integration of $1/x$ and other similar functions we have to extend the differential field. In order to integrate rational functions an extension by logarithms will be sufficient.

Extensions of differential fields

Let L be a differential field and $K \subset L$ a subfield of L. If differentiation doesn't lead out of K, then we say that K is a ***differential subfield*** of L, and L is a ***differential extension field*** of K. If for some $f, g \in L$ we have $f' = g'/g$, that is the derivative of f is the logarithmic derivative of g, then we write $f = \log g$. (We note that log, just like $\int$ is a relation rather than a function. In other words, log is an abstract concept here and not a logarithm function to a given base.) If we can choose $g \in K$ we say that f is ***logarithmic over*** K**.**

Example 5.18 (1) Let $g = x \in K = \mathbb{Q}(x)$, $L = \mathbb{Q}(x, f)$, where f is a new indeterminate, and let $f' = g'/g = 1/x$, that is $f = \log(x)$. Then $\int 1/x\,dx = \log(x)$.

(2) Analogically,

$$\int \frac{1}{x^2 - 2} = \frac{\sqrt{2}}{4}\log(x - \sqrt{2}) - \frac{\sqrt{2}}{4}\log(x + \sqrt{2})$$

is in the differential field $\mathbb{Q}(\sqrt{2})(x, \log(x - \sqrt{2}), \log(x + \sqrt{2}))$.

(3) Since

$$\int \frac{1}{x^3 + x} = \log(x) - \frac{1}{2}\log(x + i) - \frac{1}{2}\log(x - i) = \log(x) - \frac{1}{2}\log(x^2 + 1)\,,$$

the integral can be considered as an element of

$$\mathbb{Q}(x, \log(x), \log(x^2 + 1))$$

or an element of

$$\mathbb{Q}(i)(x, \log(x), \log(x - i), \log(x + i))$$

as well. Obviously, it is more reasonable to choose the first possibility, because in this case there is no need to extend the base field.

Hermite's method

Let K be a field of characteristic 0, $f, g \in K[x]$ non-zero relatively prime polynomials. To compute the integral $\int f/g$ using Hermite's method, we can find polynomials $a, b, c, d \in K[x]$ with

$$\int \frac{f}{g} = \frac{c}{d} + \int \frac{a}{b}\,, \tag{5.15}$$

where $\deg a < \deg b$ and b is monic and square-free. The rational function c/d is called the ***rational part,*** the expression $\int a/b$ is called the ***logarithmic part*** of the integral. The method avoids the factorisation of g into linear factors (in a factor field or some larger field), and even its decomposition into irreducible factors over K.

Trivially, we may assume that g is monic. By Euclidean division we have $f = pg + h$, where $\deg h < \deg g$, thus $f/g = p + h/g$. The integration of the polynomial part p is trivial. Let us determine the square-free factorisation of g, that is find monic and pairwise relatively prime polynomials $g_1, \ldots, g_m \in K[x]$ for which $g_m \neq 1$ and $g = g_1 g_2^2 \cdots g_m^m$. Let us construct the partial fraction decomposition (this can be achieved by the Euclidean algorithm):

$$\frac{h}{g} = \sum_{i=1}^{m} \sum_{j=1}^{i} \frac{h_{i,j}}{g_i^j}\,,$$

where every $h_{i,j}$ has smaller degree than g_i.

The Hermite-reduction is the iteration of the following step: if $j > 1$ then the integral $\int h_{i,j}/g_i^j$ is reduced to the sum of a rational function and an integral similar to the original one, with j reduced by 1. Using that g_i is square-free we get $\gcd(g_i, g_i') = 1$, thus we can can obtain polynomials $s, t \in K[x]$ by the application of the extended Euclidean algorithm for which $sg_i + tg_i' = h_{i,j}$ and $\deg s, \deg t < \deg g_i$. Hence, using integration by parts

$$\begin{aligned}
\int \frac{h_{i,j}}{g_i^j} &= \int \frac{t \cdot g_i'}{g_i^j} + \int \frac{s}{g_i^{j-1}} \\
&= \frac{-t}{(j-1)g_i^{j-1}} + \int \frac{t'}{(j-1)g_i^{j-1}} + \int \frac{s}{g_i^{j-1}} \\
&= \frac{-t}{(j-1)g_i^{j-1}} + \int \frac{s + t'/(j-1)}{g_i^{j-1}}\,.
\end{aligned}$$

It can be shown that using fast algorithms, if $\deg f, \deg g < n$, then the procedure requires $O(M(n)\log n)$ operations in the field K, where $M(n)$ is a bound on the number of operations needed to multiply to polynomials of degree at most n.

Hermite's method has a variant that avoids the partial fraction decomposition of h/g. If $m = 1$ then g is square-free. If $m > 1$ then let

$$g^* = g_1 g_2^2 \cdots g_{m-1}^{m-1} = \frac{g}{g_m^m}\,.$$

Since $\gcd(g_m, g^* g_m') = 1$, there exist polynomials $s, t \in K[x]$ for which

$$sg_m + tg^* g_m' = h\,.$$

Dividing both sides by $g = g^* g_m^m$ and integrating by parts

$$\int \frac{h}{g} = \frac{-t}{(m-1)g_m^{m-1}} + \int \frac{s + g^* t'/(m-1)}{g^* g_m^{m-1}} ,$$

thus m is reduced by one.

We note that a and c can be determined by the method of undetermined coefficients (Horowitz' method). After division we may assume $\deg f < \deg g$. As it can be seen from the algorithm, we can choose $d = g_2 g_3^2 \cdots g_m^{m-1}$ and $b = g_1 g_2 \cdots g_m$. Differentiating (5.15) we get a system of linear equations on $\deg b$ coefficients of a and $degd$ coefficients of c, altogether n coefficients. This method in general is not as fast as Hermite's method.

The algorithm below performs the Hermite-reduction for a rational function f/g of the variable x.

HERMITE-REDUCTION(f, g)

```
 1  p ← quo(f, g)
 2  h ← rem(f, g)
 3  (g[1], ..., g[m]) ← SQUARE-FREE(g)
 4  construct partial fraction decomposition of (h/g), compute counters h[i, j] belonging to g[i]^j
 5  rac ← 0
 6  int ← 0
 7  for i ← 1 to m
 8      do int ← int + h[i, 1]/g[i]
 9      for j ← 2 to i
10          do n ← j
11          while n > 1
12                  do determine s and t from the equation s · g[i] + t · g[i]' = h[i, j]
13                  n ← n − 1
14                  rac ← rac − (t/n)/g[i]^n
15                  h[i, n] ← s + t'/n
16          int ← int + h[i, 1]/g[i]
17  red ← rac + ∫ p + ∫ int
18  return red
```

If for some field K of characteristic 0 we want to compute the integral $\int a/b$, where $a, b \in K[x]$ are non-zero relatively prime polynomials with $\deg a < \deg b$, b square-free and monic, we can proceed by decomposing the polynomial b into linear factors in its factor field L: $b = \prod_{k=1}^{n}(x - \text{ł}a_k)$, then constructing the partial fraction decomposition over L: $a/b = \sum_{k=1}^{n} c_k/(x - \text{ł}a_k)$, finally integrating:

$$\int \frac{a}{b} = \sum_{k=1}^{n} c_k \log(x - \text{ł}a_k) \in L(x, \log(x - \text{ł}a_1), \ldots, \log(x - \text{ł}a_n)) .$$

The disadvantage of this method, as we have seen in the example of the function $1/(x^3 + x)$, is that the degree of the field extension L can be too large. An extension degree as large as $n!$ can occur, which leads to totally unmanageable cases. On the other hand, it is not clear either if a field extension is needed at all: for example in the case of the function $1/(x^2 - 2)$ can't

we compute the integral without extending the base field? The following theorem enables us to choose the degree of the field extension as small as possible.

Theorem 5.22 (Rothstein–Trager integration algorithm). *Let K be a field of characteristic 0, $a, b \in K[x]$ non-zero relatively prime polynomials, $\deg a < \deg b$, b square-free and monic. If L is an algebraic extension of K, $c_1, \ldots, c_k \in L \setminus K$ are square-free and pairwise relatively prime monic polynomials then the following statements are equivalent:*

(1) $$\int \frac{a}{b} = \sum_{i=1}^{k} c_i \log v_i \ ,$$

(2) *The polynomial $r = res_x(b, a - yb') \in K[y]$ can be decomposed into linear factors over L, $c_1, \ldots, c_k$ are exactly the distinct of r, and $v_i = gcd(b, a - c_i b')$, if $i = 1, \ldots, k$. Here, res_x is the resultant taken in the indeterminate x.*

Example 5.19 Let us consider again the problem of computing the integral $\int 1/(x^3 + x)\,dx$. In this case

$$r = \mathrm{res}_x(x^3 + x, 1 - y(3x^2 + 1)) = -4z^3 + 3y + 1 = -(2y + 1)^2(y - 1) \ ,$$

the roots of which are $c_1 = 1$ and $c_2 = -1/2$. Thus

$$\begin{aligned} v_1 &= \gcd(x^3 + x, 1 - (3x^2 + 1)) = x \ , \\ v_2 &= \gcd(x^3 + x, 1 + \frac{1}{2}(3x^2 + 1)) = x^2 + 1 \ . \end{aligned}$$

The algorithm which is easily given based on the previous theorem can be slightly improved: instead of computing $v_i = \gcd(b, a - c_i b')$ (by calculations over the field L), v_i can be computed over K as well, applying the Extended-Euclidean-normalised algorithm. This was discovered independently by Trager, , respectively Lazard and Rioboo. It is not difficult to show that the running time of the complete integration algorithm obtained this way is $O(nM(n) \lg n)$, if $\deg f, \deg g < n$.

Theorem 5.23 (Lazard–Rioboo–Trager-formula). *With the notations of the previous theorem, let e denote the multiplicity of c_i as a root of the polynomial $r = res_x(b, a - yb')$. Then*

(1) $\deg v_i = e$;

(2) *if $w(x, y) \in K(y)[x]$ denotes the remainder of degree e in the* Extended-Euclidean-normalised-*algorithm performed in $K(y)[x]$ on b and $a - yb'$, then $v_i = w(x, c_i)$.*

The algorithm below is the improved Lazard–Rioboo–Trager version of the Rothstein–Trager-method. We compute $\int a/b$ for the rational function a/b of the indeterminate x, where b is square-free and monic and $\deg a < \deg b$.

INTEGRATE-LOGARITHMIC-PART(a, b, x)

```
1  Let r(y) ← res_x(b, a − yb′) by the subresultant algorithm, furthermore
2  w_e(x, y) the remainder of degree e during the computation
3  (r_1(y), ..., r_k(y)) ←SQUARE-FREE(r(y))
4  int ← 0
5  for i ← 1 to k
6      do if r_i(y) ≠ 1
7           then w(y) ← the gcd of the coefficients of w_i(x, y)
8                (l(y), s(y), t(y)) ←EXTENDED-EUCLIDEAN-NORMALIZED(w(y), r_i(y))
9                w_i(x, y) ←PRIMITIVE-PART(rem(s(y) · w_i(x, y), r_i(y)))
10               (r_{i,1}, ..., r_{i,k}) ←FACTORS(r_i)
11               for j ← 1 to k
12                    do d ← deg(r_{i,j})
13                       c ←SOLVE(r_{i,j}(y) = 0, y)
14                       if d = 1
15                          then int ← int + c · log(w_i(x, c))
16                          else for n ← 1 to d
17                                 do int ← int + c[n] · log(w_i(x, c[n]))
18 return int
```

Example 5.20 Let us consider again the problem of computing the integral $\int 1/(x^2-2)\,dx$. In this case

$$r = \mathrm{res}_x(x^2 - 2, 1 - y \cdot 2x) = -8y^2 + 1 \,.$$

The polynomial is irreducible in $\mathbb{Q}[x]$, thus we cannot avoid the extension of $\mathbb{Q}$. The roots of r are $\pm 1/\sqrt{8}$. From the EXTENDED-EUCLIDEAN-NORMALIZED-algorithm over $\mathbb{Q}(y)$ $w_1(x, y) = x - 1/(2y)$, thus the integral

$$\int \frac{1}{x^2-2}\,dx = \frac{1}{\sqrt{8}}\log(x - \sqrt{2}) - \frac{1}{\sqrt{8}}\log(x + \sqrt{2}) \,.$$

5.4.2. The Risch integration algorithm

Surprisingly, the methods found for the integration of rational functions can be generalised for the integration of expressions containing common functions (sin, exp etc.) and their inverse. Computer algebra systems can compute the integral of remarkably complicated functions, but sometimes they fail in seemingly very simple cases, for example the expression $\int x/(1+e^x)\,dx$ is returned unevaluated or else the result contains a special non-elementary function, for example the logarithmic integral. This is due to the fact that in such cases the integral cannot be given in „closed form".

Although the basic results for integration in „closed form" had been discovered by Liouville in 1833, the corresponding algorithmic methods were only developed by Risch in 1968.

Elementary functions

The functions usually referred to as functions in „closed form" are the ones composed of

rational functions, exponential functions, logarithms, trigonometric and hyperbolic functions, their inverses and n-th roots (or more generally „inverses" of polynomial functions, that is solutions of polynomial equations); that is any nesting of the above functions is again a function in „closed form".

One might note that while $\int 1/(1+x^2)\,dx$ is usually given in the form $\operatorname{arctg}(x)$, the algorithm for the integration of rational functions returns

$$\int \frac{1}{1+x^2}\,dx = \frac{i}{2}\log(x+i) - \frac{i}{2}\log(x-i)$$

as solution. Since trigonometric, hyperbolic functions and their inverses in $\mathbb{C}$ can be expressed in terms of exponentials and logarithms, we can restrict our attention to exponentials and logarithms. Surprisingly, it turns out that the only extensions needed are logarithms (besides algebraic numbers) in the general case, too.

Exponential elements

Let L be a differential extension field of the differential field K. If for a $\theta \in L$ there exists a $u \in K$ such that $\theta'/\theta = u'$, that is the logarithmic derivative of θ equals the derivative of an element of K then we say that θ is ***exponential*** over K and we write $\theta = \exp(u)$. If only the following is true: for an element $\theta \in L$ there is a $u \in K$ such that $\theta'/\theta = u$, that is the logarithmic derivative of θ is an element of K then we call θ ***hyperexponential*** over K.

Logarithmic, exponential or hyperexponential elements may be algebraic or transcendent over K.

Elementary extensions

Let L be a differential extension field of the differential field K. If

$$L = K(\theta_1, \theta_2, \ldots, \theta_n)\ ,$$

where θ_j for $j = 1, 2, \ldots, n$ is logarithmic, exponential or algebraic over the field

$$K_{j-1} = K(\theta_1, \ldots, \theta_{j-1})$$

($K_0 = K$) then L is called an ***elementary extension*** of K. If for $j = 1, 2, \ldots, n$ θ_j is either transcendental and logarithmic, or transcendental exponential over K_{j-1}, then L is a ***transcendental elementary extension*** of K.

Let $C(x)$ be the differential field of rational functions with the usual differentiation and constant subfield $\mathbb{C}$. An elementary extension of $C(x)$ is called a field of elementary functions, a transcendental elementary extension of $C(x)$ is called a field of transcendental elementary functions.

Example 5.21 The function $f = \exp(x)+\exp(2x)+\exp(x/2)$ can be written in the form $f = \theta_1+\theta_2+\theta_3 \in \mathbb{Q}(x, \theta_1, \theta_2, \theta_3)$, where $\theta_1 = \exp(x)$, $\theta_2 = \exp(2x)$, $\theta_3 = \exp(x/2)$. Trivially, θ_1 is exponential over $\mathbb{Q}(x)$, θ_2 is exponential over $\mathbb{Q}(x, \theta_1)$ and θ_3 is exponential over $\mathbb{Q}(x, \theta_1, \theta_2)$. Since $\theta_2 = \theta_1^2$ and $\mathbb{Q}(x, \theta_1, \theta_2) = \mathbb{Q}(\theta_1)$, f can be written in the simpler form $f = \theta_1 + \theta_1^2 + \theta_3$. The function θ_3 is not only exponential but also algebraic over $\mathbb{Q}(x, \theta_1)$, since $\theta_3^2 - \theta_1 = 0$, that is $\theta_3 = \theta_1^{1/2}$. So $f = \theta_1 + \theta_1^2 + \theta_1^{1/2} \in \mathbb{Q}(x, \theta_1, \theta_1^{1/2})$. But f can be put in an even simpler form:

$$f = \theta_3^2 + \theta_3^4 + \theta_3 \in \mathbb{Q}(x, \theta_3)\ .$$

Example 5.22 The function

$$f = \sqrt{\log(x^2 + 3x + 2)(\log(x + 1) + \log(x + 2))}$$

can be written in form $f = \theta_4 \in \mathbb{Q}(x, \theta_1, \theta_2, \theta_3, \theta_4)$, where $\theta_1 = \log(x^2 + 3x + 1)$, $\theta_2 = \log(x + 1)$, $\theta_3 = \log(x+2)$, and θ_4 satisfies the algebraic equation $\theta_4^2 - \theta_1(\theta_2 + \theta_3) = 0$; but also in the much simpler form $f = \theta_1 \in \mathbb{Q}(x, \theta_1)$.

Example 5.23 The function $f = \exp(\log(x)/2)$ can be written in the form $f = \theta_2 \in \mathbb{Q}(x, \theta_1, \theta_2)$, where $\theta_1 = \log(x)$ and $\theta_2 = \exp(\theta_1/2)$, so θ_1 is logarithmic over $\mathbb{Q}(x)$, and θ_2 is exponential over $\mathbb{Q}(x, \theta_1)$. But $\theta_2^2 - x = 0$, so θ_2 is algebraic over $\mathbb{Q}(x)$, and $f(x) = x^{1/2}$.

The integration of elementary functions

The integral of an element of a field of elementary functions will be completely characterised by Liouville's Principle in the case when it is an elementary function. Algebraic extensions however – if not only the constant field is extended – cause us great difficulty.

Here we only deal with the integration of elements of fields of transcendental elementary functions by the Risch integration algorithm.

In practice, this means an element of the field of transcendental elementary functions $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)(x, \theta_1, \ldots, \theta_n)$, where $\alpha_1, \ldots, \alpha_k$ are algebraic over $\mathbb{Q}$ and the integral is an element of the field

$$\mathbb{Q}(\alpha_1, \ldots, \alpha_k, \ldots, \alpha_{k+h})(x, \theta_1, \ldots, \theta_n, \ldots, \theta_{n+m})$$

of elementary functions. In principle it would be simpler to choose $\mathbb{C}$ as constant subfield but, as we have seen in the case of rational functions, this is impossible, for we can only compute in an exact way in special fields like algebraic number fields; and we even have to keep the number and degrees of $\alpha_{k+1}, \ldots, \alpha_{k+h}$ as small as possible. Nevertheless, we will deal with algebraic extensions of the constant subfield dynamically: we can imagine that necessary extensions have already been made, while in practice we only perform extensions when they become necessary.

After the conversion of trigonometric and hyperbolic functions (and their inverses) to exponentials (respectively, logarithms), the integrand becomes an element of a field of elementary functions. Examples 5.21. and 5.22. show that there are functions that do not seem to be elements of a transcendental elementary extension „at first sight", and yet they are; and example 5.23. shows that there are functions that seem to be elements of such an extension „at first sight", and yet they are not. The first step is to represent the integrand as an element of a field of transcendental elementary functions using the algebraic relationships between the different exponential and logarithmic functions. We will not consider how this can be done. If we succeeded in doing so can be verified by the following structure theorem by Risch. We omit the proof of the theorem. We will need a definition.

An element θ is ***monomial*** over a differential field K if K and $K(\theta)$ have the same constant field, θ is transcendental over K and it is either exponential or logarithmic over K.

Theorem 5.24 (Structure Theorem). *Let K the field of constants and $K_n = K(x, \theta_1, \ldots, \theta_n)$ a differential extension field of $K(x)$, which has constant field K. Let us assume that for all j either θ_j is algebraic over $K_{j-1} = K(x, \theta_1, \ldots, \theta_{j-1})$, or $\theta_j = w_j$, with $w_j = \log(u_j)$ and $u_j \in K_{j-1}$, or $\theta_j = u_j$, with $u_j = exp(w_j)$ and $w_j \in K_{j-1}$. Then*

1. *$g = \log(f)$, where $f \in K_n \setminus K$, is monomial over K_n if and only if there is no product*

$$f^k \cdot \prod u_j^{k_j}, \quad k, k_j \in \mathbb{Z}, k \neq 0$$

which is an element of K;

2. *$g = exp(f)$, where $f \in K_n \setminus K$, is monomial over K_n if and only if there is no linear combination*

$$f + \sum c_j w_j, \quad c_j \in \mathbb{Q}$$

which is an element of K.

Product and summation is only taken for logarithmic and exponential steps.

The most important classic result of the entire theory is the following theorem.

Theorem 5.25 (Liouville's Principle). *Let K be a differential field with constant field C. Let L be a differential extension field of K with the same constant field. Let us assume that $g' = f \in K$. Then there exist constants $c_1, \ldots, c_m \in C$ and elements $v_0, v_1, \ldots, v_m \in K$, such that*

$$f = v_0' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} ,$$

that is

$$g = \int f = v_0 + \sum_{j=1}^{m} c_j \log(v_j) .$$

Note that the situation is similar to the case of rational functions.

We shall not prove this theorem. Although the proof is lengthy, the idea of the proof is easy to describe. First we show that a transcendental exponential extension cannot be „eliminated", that is differentiating a rational function of it, the new element does not vanish. This is due to the fact that differentiating a polynomial of an element of the transcendental exponential extension we get a polynomial of the same degree, and the original polynomial does not divide the derivative except in case when the original polynomial is monomial. Next we show that no algebraic extension is needed to express the integral. This is essentially due to the fact that substituting an element of an algebraic extension into its minimal polynomial we get zero, and differentiating this equation we get the derivative of the extending element as a rational function of the element. Finally we have to examine extensions by transcendental logarithmic elements. We show that such an extending element can be eliminated by differentiation if and only if it appears in a linear polynomial with constant leading coefficient. This is due to the fact that differentiating a polynomial of such an extending element we get a polynomial the degree of which is either the same or is reduced by one, the latter case only being possible when the leading coefficient is constant.

The Risch algorithm
Let K be an algebraic number field over $\mathbb{Q}$, and $K_n = K(x, \theta_1, \ldots, \theta_n)$ a field of transcendental elementary functions. The algorithm is recursive in n: using the notation $\theta = \theta_n$ we will integrate a function $f(\theta)/g(\theta) \in K_n = K_{n-1}(\theta)$, where $K_{n-1} = K(x, \theta_1, \ldots, \theta_{n-1})$. (The case $n = 0$ is the integration of rational functions.) We may assume f and g relatively prime and g monic. Besides differentiation with respect to x we will also use differentiation with respect to θ which we denote by $d/d\theta$. In the following we shall only present the algorithms.

Risch algorithm: logarithmic case
With the notations of the previous paragraph we first presume that θ is transcendental and logarithmic, $\theta' = u'/u$, $u \in K_{n-1}$. By Euclidean division $f(\theta) = p(\theta)g(\theta) + h(\theta)$, hence

$$\int \frac{f(\theta)}{g(\theta)} = \int p(\theta) + \int \frac{h(\theta)}{g(\theta)} \; .$$

Unlike the integration of rational functions, here the integration of the polynomial part is more difficult. Therefore we begin with the integration of the rational part.

Logarithmic case, rational part
Let $g(\theta) = g_1(\theta)g_2^2(\theta)\cdots g_m^m(\theta)$ the square-free factorisation of $g(\theta)$. Then

$$\gcd\left(g_j(\theta), \frac{d}{d\theta}g_j(\theta)\right) = 1$$

is obvious. It can be shown that the much stronger condition $\gcd(g_j(\theta), g_j(\theta)') = 1$ is also fulfilled. By partial fraction decomposition

$$\frac{h(\theta)}{g(\theta)} = \sum_{i=1}^{m} \sum_{j=1}^{i} \frac{h_{i,j}(\theta)}{g_i(\theta)^j} \; .$$

We use Hermite-reduction: using the extended Euclidean algorithm we get polynomials $s(\theta), t(\theta) \in K_{n-1}[\theta]$ which satisfy $s(\theta)g_i(\theta) + t(\theta)g_i(\theta)' = h_{i,j}(\theta)$ and $\deg s(\theta), \deg t(\theta) < \deg g_i(\theta)$. Using integration by parts

$$\begin{aligned}
\int \frac{h_{i,j}(\theta)}{g_i^j(\theta)} &= \int \frac{t(\theta)\cdot g_i(\theta)'}{g_i(\theta)^j} + \int \frac{s(\theta)}{g_i(\theta)^{j-1}} \\
&= \frac{-t(\theta)}{(j-1)g_i(\theta)^{j-1}} + \int \frac{t(\theta)'}{(j-1)g_i(\theta)^{j-1}} + \int \frac{s(\theta)}{g_i(\theta)^{j-1}} \\
&= \frac{-t(\theta)}{(j-1)g_i(\theta)^{j-1}} + \int \frac{s(\theta) + t(\theta)'/(j-1)}{g_i(\theta)^{j-1}} \; .
\end{aligned}$$

Continuing this procedure while $j > 1$ we get

$$\int \frac{h(\theta)}{g(\theta)} = \frac{c(\theta)}{d(\theta)} + \int \frac{a(\theta)}{b(\theta)} \; ,$$

where $a(\theta), b(\theta), c(\theta), d(\theta) \in K_{n-1}[\theta]$, $\deg a(\theta) < \deg b(\theta)$ and $b(\theta)$ is a square-free and monic polynomial.

It can be shown that the Rothstein–Trager-method can be applied to compute the integral $\int a(\theta)/b(\theta)$. Let us calculate the resultant

$$r(y) = \mathrm{res}_\theta(b(\theta), a(\theta) - y \cdot b(\theta)') \ .$$

It can be shown that the integral is elementary if and only if $r(y)$ is of form $r(y) = \overline{r}(y)s$ where $\overline{r}(y) \in K[y]$ and $s \in K_{n-1}$. If we compute the primitive part of $r(y)$, choose it as $\overline{r}(y)$ and any coefficient of $\overline{r}(y)$ is not a constant then there is no elementary integral. Otherwise, let $c_1, \ldots, c_k$ be the distinct roots of $\overline{r}(y)$ in its factor field and let

$$v_i(\theta) = \gcd(b(\theta), a(\theta) - c_i b(\theta)') \in K_{n-1}(c_1, \ldots, c_k)[\theta] \ ,$$

for $i = 1, \ldots, k$. It can be shown that

$$\int \frac{a(\theta)}{b(\theta)} = \sum_{i=1}^{k} c_i \log(v_i(\theta)) \ .$$

Let us consider a few examples.

Example 5.24 The integrand of the integral $\int 1/\log(x)$ is $1/\theta \in \mathbb{Q}(x, \theta)$, where $\theta = \log(x)$. Since

$$r(y) = \mathrm{res}_\theta(\theta, 1 - y/x) = 1 - y/x \in \mathbb{Q}(x)[y]$$

is a primitive polynomial and it has a coefficient that is not constant, the integral is not elementary.

Example 5.25 The integrand of the integral $\int 1/(x\log(x))$ $1/(x\theta) \in \mathbb{Q}(x, \theta)$, where $\theta = \log(x)$. Here

$$r(y) = \mathrm{res}_\theta(\theta, 1/x - y/x) = 1/x - y/x \in \mathbb{Q}(x)[y],$$

which has primitive part $1-y$. Every coefficient of this is constant, so the integral is elementary, $c_1 = 1$, $v_1(\theta) = \gcd(\theta, 1/x - 1/x) = \theta$, so

$$\int \frac{1}{x\log(x)} = c_1 \log(v_1(\theta)) = \log(\log(x)) \ .$$

Logarithmic case, polynomial part

The remaining problem is the integration of the polynomial part

$$p(\theta) = p_k\theta^k + p_{k-1}\theta^{k-1} + \cdots + p_0 \in K_{n-1}[\theta] \ .$$

According to Liouville's Principle $\int p(\theta)$ is elementary if and only if

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{k} c_j \frac{v_j(\theta)'}{v_j(\theta)} \ ,$$

where $c_j \in \overline{K}$ and $v_i \in \overline{K}_{n-1}(\theta)$ for $j = 0, 1, \ldots, m$, and $\overline{K}_{\mathbb{C}}$ is an extension of K and $\overline{K}_{n-1} = \overline{K}(x, \theta_1, \ldots, \theta_{n-1})$. We will show that $\overline{K}$ can be an algebraic extension of K. A similar

reasoning to the proof of Liouville's Principle shows that $v_0(\theta) \in \overline{K}_{n-1}[\theta]$ and $v_j(\theta) \in \overline{K}_{n-1}$ (that is independent of θ) for $j = 1, 2, \ldots, m$. Thus

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ .$$

We also get by the reasoning used in the proof of Liouville's Principle that the degree of $v_0(\theta)$ is at most $k+1$. So if $v_0(\theta) = q_{k+1}\theta^{k+1} + q_k\theta^k + \cdots + q_0$, then

$$p_k\theta^k + p_{k-1}\theta^{k-1} + \cdots + p_0 = (q_{k+1}\theta^{k+1} + q_k\theta^k + \cdots + q_0)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ .$$

Hence we get the following system of equations:

$$\begin{aligned} 0 &= q_{k+1}' \ , \\ p_k &= (k+1)q_{k+1}\theta' + q_k' \ , \\ p_{k-1} &= kq_k\theta' + q_{k-1}' \ , \\ &\vdots \\ p_1 &= 2q_2\theta' + q_1' \ , \\ p_0 &= q_1\theta' + \overline{q}_0' \ , \end{aligned}$$

where in the last equation $\overline{q}_0 = q_0 + \sum_{j=1}^{m} c_j \log(v_j)$. The solution of the first equation is simply a constant b_{k+1}. Substituting this into the next equation and integrating both sides we get

$$\int p_k = (k+1)b_{k+1} \cdot \theta + q_k \ .$$

Applying the integration procedure recursively the integral of $p_k \in K_{n-1}$ can be computed, but this equation can only be solved if the integral is elementary and it uses at most one logarithmic extension and it is exactly $\theta = \log(u)$. If this is not fulfilled then $\int p(\theta)$ cannot be elementary. If it is fulfilled then $\int p_k = c_k\theta + d_k$ for some $c_k \in \overline{K}$ and $d_k \in \overline{K}_{n-1}$, hence $b_{k+1} = c_{k+1}/(k+1) \in \overline{K}$ and $q_k = d_k + b_k$ with an arbitrary integration constant b_k. Substituting for q_k into the next equation and rearranging we get

$$p_{k-1} - kd_k\theta' = kb_k\theta' + q_{k-1}' \ ,$$

so we have

$$\int \left(p_{k-1} - kd_k \frac{u'}{u}\right) = kb_k\theta + q_{k-1}$$

after integration. The integrand on the right hand side is in K_{n-1} so we can call the integration procedure in a recursive way. Just like above, the equation can only be solved when the integral is elementary and it uses at most one logarithmic extension and it is exactly $\theta = \log(u)$. Let us assume that this is the case and

$$\int \left(p_{k-1} - kd_k \frac{u'}{u}\right) = c_{k-1}\theta + d_{k-1} \ ,$$

where $c_{k-1} \in \overline{K}$ and $d_{k-1} \in \overline{K}_{n-1}$. Then the solution is $b_k = c_{k-1}/k \in \overline{K}$ and $q_{k-1} = d_{k-1} + b_{k-1}$, where b_{k-1} is an arbitrary integration constant. Continuing the procedure, the solution of the

penultimate equation is $b_2 = c_1/2 \in \overline{K}$ and $q_1 = d_1 + b_1$ with an integration constant b_1. Substituting for q_1 into the last equation, after rearrangement and integration

$$\int \left(p_0 - d_1 \frac{u'}{u}\right) = b_1\theta + \overline{q}_0 \ .$$

This time the only condition is that the integral should be an elementary function. If it is elementary, say

$$\int \left(p_0 - d_1 \frac{u'}{u}\right) = d_0 \in \overline{K}_{n-1} \ ,$$

then $b_1 \in \overline{K}$ is the coefficient of $\theta = \log(u)$ in d_0-ban and $\overline{q}_0 = d_0 - b_1 \log(u)$, and the result is

$$\int p(\theta) = b_{k+1}\theta^{k+1} + q_k\theta^k + \cdots + q_1\theta + \overline{q}_0 \ .$$

Let us consider a few examples.

Example 5.26 The integrand of the integral $\int \log(x)$ is $\theta \in \mathbb{Q}(x, \theta)$, where $\theta = \log(x)$. If the integral is elementary then

$$\int \theta = b_2\theta^2 + q_1\theta + \overline{q}_0$$

and $0 = b_2'$, $1 = 2b_2\theta' + q_1'$, $0 = q_1\theta' + \overline{q}_0'$. With the unknown constant b_2 from the second equation $\int 1 = 2b_2\theta + q_1$. Since $\int 1 = x + b_1$, we get $b_2 = 0$, $q_1 = x + b_1$. From the third equation $-x\theta' = b_1\theta' + \overline{q}_0'$. Since $\theta' = 1/x$, after integration $\int -1 = b_1\theta + \overline{q}_0$ and $\int -1 = -x$, we get $b_1 = 0$, $\overline{q}_0 = -x$, hence $\int \log(x) = x\log(x) - x$.

Example 5.27 The integrand of the integral $\int \log(\log(x))$ is $\theta_2 \in \mathbb{Q}(x, \theta_1, \theta_2)$, where $\theta_1 = \log(x)$ and $\theta_2 = \log(\theta_1)$. If the integral is elementary then

$$\int \theta_2 = b_2\theta_2^2 + q_1\theta_2 + \overline{q}_0$$

and $0 = b_2'$, $1 = 2b_2\theta_2' + q_1'$, $0 = q_1\theta_2' + \overline{q}_0'$. With the unknown constant b_2 from the second equation $\int 1 = 2b_2\theta + q_1$. Since $\int 1 = x + b_1$, we get $b_2 = 0$, $q_1 = x + b_1$. From the third equation $-x\theta_2' = b_1\theta_2' + \overline{q}_0'$. Since $\theta_2' = \theta_1'/\theta_1 = 1/(x\log(x))$, the equation

$$\int \frac{-1}{\log(x)} = b_1\theta_2 + \overline{q}_0$$

must hold but we know from example 5.24. that the integral on the left hand side is not elementary.

Risch algorithm: exponential case

Now we assume that θ is transcendental and exponential, $\theta'/\theta = u'$, $u \in K_{n-1}$. By Euclidean division $f(\theta) = q(\theta)g(\theta) + h(\theta)$, whence

$$\int \frac{f(\theta)}{g(\theta)} = \int q(\theta) + \int \frac{h(\theta)}{g(\theta)} \ .$$

We plan using Hermite's methodfor the rational part. But we have to face an unpleasant surprise: although for the square-free factors $g_j(\theta)$

$$\gcd\left(g_j(\theta), \frac{d}{d\theta} g_j(\theta)\right) = 1$$

is obviously satisfied, the much stronger condition $\gcd(g_j(\theta), g_j(\theta)') = 1$ is not. For example if $g_j(\theta) = \theta$ then

$$\gcd(g_j(\theta), g_j(\theta)') = \gcd(\theta, u'\theta) = \theta \ .$$

It can be shown, however, that this unpleasant phenomenon does not appear if $\theta \nmid g_j(\theta)$, in such cases $\gcd(g_j(\theta), g_j(\theta)') = 1$. Thus it will be sufficient to eliminate θ from the denominator. Let $g(\theta) = \theta^\ell \overline{g}(\theta)$, where $\theta \nmid \overline{g}(\theta)$ already and let us look for polynomials $\overline{h}(\theta), s(\theta) \in K_{n-1}[\theta]$ with $\overline{h}(\theta)\theta^\ell + t(\theta)\overline{g}(\theta) = h(\theta)$, $\deg \overline{h}(\theta) < \deg \overline{g}(\theta)$ and $\deg s(\theta) < \ell$. Dividing both sides by $g(\theta)$ we get

$$\frac{f(\theta($}{g(\theta)} = q(\theta) + \frac{t(\theta)}{\theta^l} + \frac{\overline{h}(\theta)}{\overline{g}(\theta)} \ .$$

With the notation $p(\theta) = q(\theta) + t(\theta)/\theta^l$, $p(\theta)$ is a finite Laurent-series the integration of which will be no harder than the integration of a polynomial. This is not surprising if we note $\theta^{-1} = \exp(-u)$. Even so, the integration of the „polynomial part" is more difficult here too. We start with the other one.

Exponential case, rational part

Let $\overline{g}(\theta) = g_1(\theta)g_2^2(\theta)\cdots g_m^m(\theta)$ be the square-free factorisation of $\overline{g}(\theta)$. Then because of $\theta \nmid g_j(\theta)$, $\gcd(g_j(\theta), g_j(\theta)') = 1$. Using partial fraction decomposition

$$\frac{\overline{h}(\theta)}{\overline{g}(\theta)} = \sum_{i=1}^{m} \sum_{j=1}^{i} \frac{h_{i,j}(\theta)}{g_i(\theta)^j} \ .$$

Hermite-reduction goes the same way as in the logarithmic case. We get

$$\int \frac{\overline{h}(\theta)}{\overline{g}(\theta)} = \frac{c(\theta)}{d(\theta)} + \int \frac{a(\theta)}{b(\theta)} \ ,$$

where $a(\theta), b(\theta), c(\theta), d(\theta) \in K_{n-1}[\theta]$, $\deg a(\theta) < \deg b(\theta)$ and $b(\theta)$ is a square-free and monic polynomial, $\theta \nmid b(\theta)$.

It can be shown that the Rothstein–Trager-method can be applied to compute the integral $\int a(\theta)/b(\theta)$ Let us calculate the resultant

$$r(y) = \mathrm{res}_\theta(b(\theta), a(\theta) - y \cdot b(\theta)') \ .$$

It can be shown that the integral is elementary if and only if $r(y)$ is of form $r(y) = \overline{r}(y)s$ where $\overline{r}(y) \in K[y]$ and $s \in K_{n-1}$. If we compute the primitive part of $r(y)$, choose it as $\overline{r}(y)$ and any coefficient of $\overline{r}(y)$ is not a constant then there is no elementary integral. Otherwise, let $c_1, \ldots, c_k$ be the distinct roots of $\overline{r}(y)$ in its factor field and let

$$v_i(\theta) = \gcd(b(\theta), a(\theta) - c_i b(\theta)') \in K_{n-1}(c_1, \ldots, c_k)[\theta] \ ,$$

for $i = 1, \ldots, k$. It can be shown that

$$\int \frac{a(\theta)}{b(\theta)} = -\left(\sum_{i=1}^{k} c_i \deg v_i(\theta)\right) + \sum_{i=1}^{k} c_i \log(v_i(\theta)) \ .$$

Let us consider a few examples.

Example 5.28 The integrand of the integral $\int 1/(1+\exp(x))$ is $1/(1+\theta) \in \mathbb{Q}(x,\theta)$, where $\theta = \exp(x)$. Since

$$r(y) = \operatorname{res}_\theta(\theta+1, 1-y\theta) = -1-y \in \mathbb{Q}(x)[y]$$

is a primitive polynomial which has only constant coefficients, the integral is elementary, $c_1 = -1$, $v_1(\theta) = \gcd(\theta+1, 1+\theta) = 1+\theta$, thus

$$\int \frac{1}{1+\exp(x)} = -c_1 x \deg v_1(\theta) + c_1 \log(v_1(\theta)) = x - \log(\exp(x)+1) \ .$$

Example 5.29 The integrand of the integral $\int x/(1+\exp(x))$ is $x/(1+\theta) \in \mathbb{Q}(x,\theta)$, where $\theta = \exp(x)$. Since

$$r(y) = \operatorname{res}_\theta(\theta+1, x-y\theta) = -x-y \in \mathbb{Q}(x)[y]$$

is a primitive polynomial that has a non-constant coefficient, the integral is not elementary.

Exponential case, polynomial part

The remaining problem is the integration of the „polynomial part"

$$p(\theta) = \sum_{i=-\ell}^{k} p_i \theta^i \in K_{n-1}(\theta) \ .$$

According to Liouville's Principle $\int p(\theta)$ is elementary if and only if

$$p(\theta) = v_0(\theta)' + \sum_{j=1}^{m} c_j \frac{v_j(\theta)'}{v_j(\theta)} \ ,$$

where $c_j \in \overline{K}$ and $v_j \in \overline{K}_{n-1}(\theta)$ for $j = 0, 1, \ldots, m$, and $\overline{K}_{\mathbb{C}}$ is an extension of K and $\overline{K}_{n-1} = \overline{K}(x, \theta_1, \ldots, \theta_{n-1})$. It can be shown that $\overline{K}$ can be an algebraic extension of K. A similar reasoning to the proof of Liouville's Principle shows that we may assume without breaking generality: $v_j(\theta)$ is either an element of $\overline{K}_{n-1}$ (that is independent of θ), or it is monic and irreducible in $\overline{K}_{n-1}[\theta]$ for $j = 1, 2, \ldots, m$. Furthermore, it can be shown that there can be no non-monomial factor in the denominator of $v_0(\theta)$, since such a factor would be present in the derivative, too. Similarly, the denominator of $v_j(\theta)$ $(j = 1, 2, \ldots, m)$ can have no non-monomial factor either. So we get that either $v_j(\theta) \in K_{n-1}$, or $v_j(\theta) = \theta$, since this is the only irreducible monic monomial. But if $v_j(\theta) = \theta$ then the corresponding member of the sum is $c_j v_j'(\theta)/v_j(\theta) = c_j u'$, which can be incorporated into $v_0(\theta)'$. Hence we get that if $p(\theta)$ has an elementary integral then

$$p(\theta) = \left(\sum_{j=-\ell}^{k} q_j \theta^j \right)' + \sum_{j=1}^{m} c_j \frac{v_j'}{v_j} \ ,$$

where $q_j, v_j \in \overline{K}_{n-1}$ and $c_j \in \overline{K}$; that the summation should be taken over the same range as in $p(\theta)$ follows from

$$(q_j \theta^j)' = (q_j' + j u' g_j)\theta^j \ .$$

Comparing the coefficients we get the system

$$\begin{aligned} p_j &= q_j' + ju'q_j, \quad \text{ha } -\ell \le j \le k,\ j \ne 0\ , \\ p_0 &= \overline{q}_0'\ , \end{aligned}$$

where $\overline{q}_0 = q_0 + \sum_{j=1}^{m} c_j \log(v_j)$. The solution of the equation $p_0 = \overline{q}_0'$ is simply $\overline{q}_0 = \int p_0$; if this integral is not elementary then $\int p(\theta)$ cannot be elementary either, but if it is then we have determined $\overline{q}_0$. In the case $j \ne 0$ we have to solve a differential equation called ***Risch differential equation*** to determine q_j. The differential equation is of form $y' + fy = g$ where the given functions f, g are elements of K_{n-1} and we are looking for solutions in K_{n-1}. At first sight it looks as if we had replaced the problem of integration with a more difficult problem, but the linearity of the equations and that the solution has to be in K_{n-1} means a great facility. If any Risch differential equation fails to have a solution in K_{n-1} then $\int p(\theta)$ is not elementary, otherwise

$$\int p(\theta) = \sum_{j \ne 0} q_j \theta^j + \overline{q}_0\ .$$

The Risch differential equation can be solved algorithmically, but we will not go into details. Let us consider a few examples

Example 5.30 The integrand of the integral $\int \exp(-x^2)$ is $\theta \in \mathbb{Q}(x, \theta)$, where $\theta = \exp(-x^2)$. If the integral is elementary then $\int \theta = q_1\theta$, where $q_1 \in \mathbb{C}(x)$. It is not difficult to show that the differential equation has no rational solutions so $\int \exp(-x^2)$ is not elementary.

Example 5.31 The integrand of the integral $\int x^x$ is $\exp((x\log(x)) = \theta_2 \in \mathbb{Q}(x, \theta_1, \theta_2)$, where $\theta_1 = \log(x)$ and $\theta_2 = \exp(x\theta_1)$. If the integral is elementary then $\int \theta_2 = q_1\theta_2$, where $q_1 \in \mathbb{C}(x, \theta_1)$. Differentiating both sides $\theta_2 = q_1'\theta_2 + q_1(\theta_1 + 1)\theta_2$, thus $1 = q_1' + (\theta_1 + 1)q_1$. Since θ_1 is transcendental over $\mathbb{C}(x)$, by comparing the coefficients $1 = q_1' + q_1$ and $0 = q_1$, which has no solutions. Therefore $\int x^x$ is not elementary.

Example 5.32 The integrand of the integral

$$\int \frac{(4x^2 + 4x - 1)(\exp(x^2) + 1)(\exp(x^2) - 1)}{(x+1)^2}$$

is

$$f(\theta) = \frac{4x^2 + 4x - 1}{(x+1)^2}(\theta^2 - 1) \in \mathbb{Q}(x, \theta)\ ,$$

where $\theta = \exp(x^2)$. If the integral is elementary then it is of the form $\int f(\theta) = q_2\theta^2 + \overline{q}_0$, where

$$\begin{aligned} q_2' + 4xq_2 &= \frac{4x^2 + 4x - 1}{(x+1)^2}\ , \\ \overline{q}_0' &= -\frac{4x^2 + 4x - 1}{(x+1)^2}\ . \end{aligned}$$

The second equation can be integrated and $\overline{q}_0$ is elementary. The solution of the first equation is $q_2 = 1/(1 + x)$. Hence

$$\int f(\theta) = \frac{1}{x+1}\exp^2(x^2) - \frac{(2x+1)^2}{x+1} + 4\log(x+1)\ .$$

Exercises

5.4-1 Apply Hermite-reduction to the following function $f(x) \in \mathbb{Q}(x)$:

$$f(x) = \frac{441x^7 + 780x^6 - 286x^5 + 4085x^4 + 769x^3 + 3713x^2 - 43253x + 24500}{9x^6 + 6x^5 - 65x^4 + 20x^3 + 135x^2 - 154x + 49} .$$

5.4-2 Compute the integral $\int f$, where

$$f(x) = \frac{36x^6 + 126x^5 + 183x^4 + 13807/6x^3 - 407x^2 - 3242/5x + 3044/15}{(x^2 + 7/6x + 1/3)^2(x - 2/5)^3} \in \mathbb{Q}(x) .$$

5.4-3 Apply the Risch integration algorithm to compute the following integral:

$$\int \frac{x(x+1)\{(x^2e^{2x^2} - \log(x+1)^2) + 2xe^{3x^2}(x - (2x^3 + 2x^2 + x + 1)\log(x+1))\}}{((x+1)\log^2(x+1) - (x^3 + x^2)e^{2x^2})^2} dx .$$

5.5. Theory and practice

So far in the chapter we tried to illustrate the algorithm design problems of computer algebra through the presentation of a few important symbolic algorithms. Below the interested reader will find an overview of the wider universe of the research of symbolic algorithms.

5.5.1. Other symbolic algorithms

Besides the resultant method and the theory of Gröbner-bases presented in this chapter there exists algorithms for finding *real* symbolic roots of non-linear equations and inequalities, too (Collins).

There are some remarkable algorithms in the area of symbolic solution of differential equations. There exists a decision procedure similar to the Risch algorithm for the computation of solutions in closed form of a homogeneous ordinary differential equation of second degree with rational function coefficients. In the case of higher degree linear equations Abramov's procedure gives closed rational solutions of an equation with polynomial coefficients, while Bronstein's algorithm gives solutions of the form $\exp(\int f(x)dx)$. In the case of partial differential equations Lie's symmetry methods can be used. There also exists an algorithm for the factorisation of linear differential operators over formal power series and rational functions.

Procedures based on factorisation are of great importance in the research of computer algebra algorithms. Such an importance that many consider that the entire research field was born with Berlekamp's publication on an effective algorithm for the factorization of polynomials of one variable over finite fields of small characteristic p. Later, Berlekamp extended his results for larger characteristic, too. In order to have similarly good running

times he introduced probabilistic elements into the algorithm. Today's computer algebra systems use Berlekamp's procedure even for large finite fields as a routine, perhaps without most of the users knowing about the probabilistic origin of the algorithm. The method will be presented in another chapter of the book. We note that there are numerous algorithms for the factorization of polynomials over finite fields.

Not much time after polynomial factorization over finite fields was solved, Zassenhaus, taking van der Waerden's book *Moderne Algebra* from 1936 as a base, used the so called Hensel's lemma for the arithmetic of p-adic numbers to extend factorization. „Hensel-lifting" – as his procedure is called now – is a general approach for the reconstruction of factors from their modular images. Unlike interpolation, which needs multiple points from the image, Hensel-lifing only needs one point from the image. The Berlekamp–Zassenhaus-algorithm for the factorization of polynomials with integer coefficients is of fundamental importance but it has two hidden pitfalls. First, for a certain kind of polynomials the running time is exponential. Unfortunately, many „bad" polynomials appear in the case of factorization over algebraic number fields. Second, for multivariable polynomials a representation problem appears, similar to what we have encountered at the Gauss-elimination of sparse matrices. The first problem was solved by a diophantine optimisation based on the geometry of numbers, a so called lattice reduction algorithm by Lenstra–Lenstra–Lovász; it is used together with Berlekamp's method. This polynomial algorithm is completed by a procedure which ensures that the Hensel-lifting will start from a „good" modular image and that it will end „in time". Solutions have been found for the mentioned representation problem of the factorization of multivariable polynomials, as well. This is the second area where randomisation plays a crucial role in the design of effective algorithms. We note that in practice the Berlekamp–Zassenhaus–Hensel-algorithm proves more effective than the Lenstra–Lenstra–Lovász-procedure. As a contrast, the problem of polynomial factorization can be solved in polynomial time, while the best proved algorithmic bound for the factorization of the integer N is $\widetilde{O}(N^{1/4})$ (Pollard and Strassen) in the deterministic case and $L(N)^{1+o(1)}$ (Lenstra and Pomerance) in the probabilistic case, where $L(N) = e^{\sqrt{\ln N \ln\ln N}}$.

In fact, a new theory of heuristic or probabilistic methods in computer algebra is being born to avoid computational or memory explosion and to make algorithms with deterministically large running times more effective. In the case of probabilistic algorithms the probability of inappropriate operation can be positive, this may result in an incorrect answer (Monte Carlo algorithms) or – although we always get the correct answer (Las Vegas algorithms) – we may not get anything in polynomial time. Beside the examples above, nice results have been achieved in testing polynomial identity, irreducibility of polynomials, determining matrix normal forms (Frobenius, Hilbert, Smith), etc. Their role is likely to increase in the future.

So far in the chapter we gave an overview of the most important symbolic algorithms. We mentioned in the introduction that most computer algebra systems are able to perform numeric computations as well: unlike traditional systems, the precision can be set by the user. In many cases it is useful to combine symbolic and numeric computation. Let us consider for example the symbolically computed power series solution of a differential equation. After truncation evaluating the power series with the usual floating point arithmetics in certain points we get a numerical approximation of the solution. When the problem is an approximation of a physical problem the attractivity of symbolic computation is often lost; simply because they are too complicated or too slow and they are not necessary or useful

since we are looking for a numerical solution. In other cases, when the problem cannot be dealt with using symbolic computation, the only way is numerical approximation. This may be the case when the existing symbolic algorithm does not find a closed solution (e.g. the integral of non-elementary functions, etc.), or when a symbolic algorithm for the specified problem does not exist. Although more and more numerical algorithms have symbolic equivalents, numerical procedures play an important role in computer algebra. Let us think of differentiation and integration: sometimes traditional algorithms – integral transformation, power series approximation, perturbation methods – can be the most effective.

In the design of computer algebra algorithms parallel architectures will play an increasing role in the future. Although many existing algorithms can be parallelised „by the look", it is not obvious that good sequential algorithms will perform optimally on parallel architectures as well: the optimal performance might be achieved by a completely different method.

5.5.2. An overview of computer algebra systems

The development of computer algebra systems is linked with the development of computer science and algorithmic mathematics. In the early period of computers researchers of different fields began the development of the first computer algebra systems to facilitate and accelerate their symbolic computations; these systems, reconstructed and continuously updated are present in their manieth versions today. ***General purpose*** computer algebra systems appeared in the seventies, these provide a wide range of built-in data structures, mathematical functions and algorithms, trying to cover a wide area of users. Because of their large need of computational resources their expansion became explosive in the beginning of the eighties when microprocessor-based workstations appeared. Better hardware environments, more effective resource management, the use of system-independent high-level languages and last but not least social-economic demands gradually transformed general purpose computer algebra systems into market products, which also resulted in a better user interface and document preparation.

Below we list the most widely known general and special purpose computer algebra systems and libraries.

- General purpose computer algebra systems: Axiom, Derive, Form, GNU-calc, Jacal, Macsyma, Maxima, Maple, Distributed Maple, MathCAD, Matlab Symbolic Math Toolbox, Scilab, MAS, Mathematica, Mathview, Mock-Mma, MuPAD, Reduce, Risa.
- Algebra and number theory: Bergman, CoCoA, Felix, Fermat, GRB, Kan, Macaulay, Magma, Numbers, Pari, Simath, Singular.
- Algebraic geometry: Casa, Ganith.
- Group theory: Gap, LiE, Magma, Schur.
- Tensor analysis: Cartan, FeynCalc, GRG, GRTensor, MathTensor, RedTen, Ricci, TTC.
- Computer algebra libraries: Apfloat, BigNum, GNU MP, Kant, LiDiA, NTL, Saclib, Ubasic, Weyl, Zen.

Most general purpose computer algebra systems are characterised by

- interactivity,

- knowledge of mathematical facts,
- a high-level, declarative[2] programming language with the possibility of functional programming and the knowledge of mathematical objects,
- expansibility towards the operational system and other programs,
- integration of symbolic and numeric computations,
- automatic (optimised) C and Fortran code generation,
- graphical user interface,
- 2 and 3 dimensional graphics and animation,
- possibility of editing text and automatic LaTeX conversion,
- on-line help.

Computer algebra systems are also called ***mathematical expert systems***. Today we can see an tonishing development of general purpose computer algebra systems, mainly because their knowledge and wide application area. But it would be a mistake to underestimate special systems, which play a very important role in many research fields, besides, in many cases are easier to use and more effective due to their system of notation and the low-level programming language implementation of their algorithms. It is essential that we choose the most appropriate computer algebra system to solve a specified problem.

Problems

5-1. The length of coefficients in the series of remainders in a Euclidean division
Generate two pseudorandom polynomials of degree $n = 10$ in $\mathbb{Z}[x]$ with coefficients with $l = 10$ decimal digits. Perform a single Euclidean division in (in $\mathbb{Q}[x]$) and compute the ratio of the maximal coefficients of the remainder and the original polynomial (determined by the function λ). Repeat the computation $t = 20$ times and compute the average. What is the result? Repeat the experiment with $l = 100, 500, 1000$.

5-2. Simulation of the* MODULAR-GCD-SMALLPRIMES *algorithm
Using simulation, give an estimation for the optimal value of the variable n in the MODULAR-GCD-SMALLPRIMES algorithm. Use random polynomials for different degrees and coefficient magnitudes.

5-3. Modified pseudo-euclidean division
Let $f, g \in \mathbb{Z}[x]$, $\deg f = m \geq n = \deg g$. Modify the pseudo-euclidean division in such a way that in the equation

$$g_n^s f = gq + r$$

instead of the exponent $s = m - n + 1$ put the smallest value $s \in \mathbb{N}$, for which $q, r \in \mathbb{Z}[x]$. Replace the procedures pquo() and prem() in the PRIMITIVE-EUCLIDEAN algorithm by the obtained procedures spquo() and sprem(). Compare the amount of memory space required by the algorithms.

5-4. Construction of reduced Gröbner basis
Design an algorithm that from a given Gröbner-basis G computes a reduced Gröbner basis.

[2]Declarative programming languages specify the desired result unlike imperative languages, which describe how to get the result.

5-5. Implementation of Hermite-reduction
Implement Hermite-reduction in a chosen computer algebra language

5-6. Integration of rational functions
Write a program for the integration of rational functions.

Chapter notes

The algorithms CLASSICAL-EUCLIDEAN and EXTENDED-EUCLIDEAN for non-negative integers are described in [7]. A natural continuation of the theory of resultants leads to subresultants, which can help in reducing the growth of the coefficients in the EXTENDED-EUCLIDEAN algorithm (see e.g. [9, 10]).

Gröbner bases were introduced by B. Buchberger in 1965 [2]. Several authors examined polynomial ideals before this. The most well-known is perhaps Hironaka, who used bases of ideals of power series to resolve singularities over $\mathbb{C}$. He was rewarded a Fields-medal for his work. His method was not constructive, however. Gröbner bases have been generalised for many algebraic structures in the last two decades.

The bases of differential algebra have been layed by J. F. Ritt in 1948 [24]. The square-free factorization algorithm used in symbolic integration can be found e.g. in the books [9, 10]. The importance of choosing the smallest possible extension degree in the Hermite-reduction is illustrated by example 11.11 in [10], where the decomposition field has very large degree but the integral can be expressed in an extension of degree 2. The proof of the Rothstein–Trage integration algorithm can be found in [9] (theorem 22.8.). We note that the algorithm was found independently by Rothstein and Trager. The proof of the correctness Lazard–Rioboo–Trager formula, the analysis of the running time of the INTEGRATE-LOGARITHMIC-PART algorithm, an overview of the procedures that deal with the difficulties of algebraic extension steps, the determination of the hyperexponential integral (if exists) of a hyperexponential element over $C(x)$, the proof of Liouville's principle and the proofs of the statements connected to the Risch algorithm can be found in the book [9].

There are many books and publications available on computer algebra and related topics. The interested reader will find mathematical description in the following general works: Caviness [3], Davenport et al. [8], von zur Gathen et al. [9], Geddes et al. [10], Knuth [16, 17, 18], Mignotte [20], Mishra [21], Pavelle et al. [23], Winkler [26].

The computer-oriented reader will find further information on computer algebra in Christensen [4], Gonnet and Gruntz [11], Harper et al. [13], and on the world wide web.

A wide range of books and articles deal with applications, e.g. Akritas [1], Cohen et al. (ed.) [5, 6], Grossman (ed.) [12], Hearn (ed.) [14], Kovács [19] and Odlyzko [22].

For the role of computer algebra systems in education see e.g. the works of Karian [15] and Uhl [25].

Conference proceedings: AAECC, DISCO, EUROCAL, EUROSAM, ISSAC and SYMSAC.

Computer algebra journals: *Journal of Symbolic Computation* – Academic Press, *Applicable Algebra in Engineering, Communication and Computing* – Springer-Verlag, SIGSAM *Bulletin* – ACM Press.

The Department of Computer Algebra of the University Eötvös Loránd, Budapest takes the works [9, 10, 20, 26] as a base in the education.

Bibliography

[1] A. G. Akritas. *Elements of Computer Algebra with Applications*. John Wiley & Sons, 1989. 255

[2] B. Buchberger. Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal, 1965. PhD disszertáció, Leopold-Franzens-Universität, Innsbruck. 255

[3] B. F. Caviness. Computer algebra: past and future. *Journal of Symbolic Computations*, 2:217–263, 1986. 255

[4] S. M. Christensen. Resources for computer algebra. *Computers in Physics*, 8:308–315, 1994. 255

[5] A. M. Cohen, L. van Gasten, S. Lunel (szerkesztők). *Computer Algebra for Industry 2, Problem Solving in Practice*. John Wiley & Sons, 1995. 255

[6] A. M. Cohen (szerkesztő). *Computer Algebra for Industry: Problem Solving in Practice*. John Wiley & Sons, 1993. 255

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. The MIT Press/McGraw-Hill, 2004 (Fifth corrected printing of 2. edition. 255

[8] J. Davenport, Y. Siret, E. Tournier, E.. *Computer Algebra: Systems and Algorithms for Algebraic Computation*. Academic Press, 2000. 255

[9] J. Gathen, von zur. *Modern Computer Algebra*. Cambridge University Press, 2003. 255

[10] K. O. Geddes S. Czapor, G. Labahhn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992. 255

[11] G. Gonnet, D. Gruntz, L. Bernardin Computer algebra systems. In A. Ralston, E. D. Reilly, D. Hemmendinger (szerkesztők), *Encyclopedia of Computer Science*, 287–301. o. Nature Publishing Group, 4. kiadás, 2000. 255

[12] R. Grossman. *Symbolic Computation: Applications to Scientific Computing*, *Frontiers in Applied Mathematics* 5. kötete. SIAM, 1989. 255

[13] D. Harper, C. Wooff D. Hodginson. *A Guide to Computer Algebra Systems*. John Wiley & Sons, 1991. 255

[14] A. C. Hearn. *Future Directions for Research in Symbolic Computation*. SIAM Reports on Issues in the Mathematical Sciences. SIAM, 1990. 255

[15] Z. Karian, A. Starrett. Use of symbolic computation in probability and statistics. In Z. Karian (szerkesztő), *Symbolic Computation in Undergraduate Mathematics Education*, number24 in Notes of Mathematical Association of America. Mathematical Association of America, 1992. 255

[16] D. E. Knuth. *Fundamental Algorithms*, *The Art of Computer Programming* 1. kötete. Addison-Wesley, 1968 (3. updated edition). 255

[17] D. E. Knuth. *Seminumerical Algorithms*, 2. kötet. Addison-Wesley, 1969 (3. corrected edition). 255

[18] D. E. Knuth. *Sorting and Searching*, *The Art of Computer Programming* 3. kötete. Addison-Wesley, 1973 (3. corrected edition). 255

[19] A. Kovács. Komputer algebra a tudományokban és a gyakorlatban (Computer algebra in science and practice). *Alkalmazott Matematikai Lapok*, 18:181–202, 1994-98. 255

[20] M. E. Mignotte. *Mathematics for Computer Algebra*. Springer, 1992. 255

[21] B. E. Mishra. *Algorithmic Algebra*. Springer, 1993. 255

[22] A. Odlyzko. *Applications of Symbolic Mathematics to Mathematics*. Kluwer Academic Publishers, 1985. 255

[23] R. Pavelle, M. Rothstein. Computer algebra. *Scientific American*, 245(12):102–113, 1981. 255

[24] J. Ritt. *Integration in Finite Terms*. Columbia University Press, 1948. 255

[25] J. J. Uhl. MATHEMATICA and Me. *Notices of AMS*, 35:1345–1345, 1988. 255

[26] F. Winkler. *Polynomial Algorithms in Computer Algebra*. Springer-Verlag, 1990. 255

Subject Index

Name index

Contents