Introduction to Computer Algebra

K. Kalorkoti

School of Informatics
University of Edinburgh
Informatics Forum
10 Crichton Street
Edinburgh EH8 9AB
Scotland
E-mail: kk@inf.ed.ac.uk

January 2019

Preface

These notes are for the fourth year and MSc course in Computer Algebra. They contain more material than can fit into the time available. The intention is to allow you to browse through extra material with relative ease. If you have time, you are strongly encouraged to follow in detail at least some of the extra material. However it will *not* be assumed that you have done this and success in the course does not depend on it. The course itself will cover a selection of the topics mentioned, even so some details will be omitted. For example Gröbner bases are given quite a full treatment. In the course itself more emphasis will be placed on understanding their use, the basic algorithm and the intuitive explanation of how and why they work. For other topics we will look at the details quite closely. Broadly speaking, the examinable parts are those covered in the lectures or assigned for home reading (the lecture log at the course web page, http://www.inf.ed.ac.uk/teaching/courses/ca, will state the parts of the notes covered). A guide to revision will be handed out at the end of the course (and placed on the course web site) stating exactly which topics are examinable.

You will find a large number of exercises in the notes. While they are not part of the formal coursework assessment (separate exercise sheets will be issued for this) you are strongly encouraged to try some of them. At the very least you should read and understand each exercise. You are welcome to hand me any of your attempts for feedback; we will in any case discuss some of the exercises in lectures. As a general rule the less familiar you are with a topic the more exercises you should attempt from the relevant section. One or more of these exercises will be suggested at the end of some lectures for you to try. There will be a follow up discussion at the next lecture to provide you with feedback on your attempts. If you find any errors or obscure passages in the notes please let me know.

Parts of these notes are based on lectures given by Dr. Franz Winkler at the Summer School in Computer Algebra which was held at RISC-LINZ, Johannes Kepler University, Austria, during the first two weeks of July, 1990. I am grateful for his permission to use the material.

Note: These notes and all other handouts in the course are for your private study and must not be communicated to others in any way; breaking this requirement constitutes academic misconduct.

He liked sums,
but not the way they were taught.
— BECKETT, *Malone Dies*

Contents

1 Introduction

1.1 General introduction

Consider the expression

$$f = \frac{32x^8 - 16x^7 + 82x^6 - 40x^5 + 85x^4 - 40x^3 + 101x^2 - 48x + 6}{8x^5 - 2x^4 + 4x^3 - x^2 + 12x - 3} \tag{1}$$

Suppose that we want to differentiate f. In a sense there is no problem at all since we can use the formula

$$\frac{d(p/q)}{dx} = \frac{q\frac{dp}{dx} - p\frac{dq}{dx}}{q^2}.$$

However an attempt to apply this to f by hand leads to quite messy calculations. Even if no error happens during these calculations the answer might be needlessly complicated. Indeed a direct application of the formula to f (with expansion of the products in the numerator) yields

$$\frac{\begin{array}{r}768x^{12} - 512x^{11} + 1392x^{10} - 776x^9 + 3152x^8 - 1928x^7 + 3212x^6 \\ -1626x^5 + 2768x^4 - 1548x^3 + 1452x^2 - 594x + 72\end{array}}{(8x^5 - 2x^4 + 4x^3 - x^2 + 12x - 3)^2}.$$

It might be better to simplify f as much as possible before operating on it. Here the word 'simplify' denotes the removal of common factors from numerator and denominator. For example the expression

$$\frac{x^4 - 1}{x^2 + 1}$$

simplifies to

$$x^2 - 1$$

since $x^4 - 1 = (x^2 - 1)(x^2 + 1)$ (we discuss this point in greater detail in §4.7.8). This process is rather like that of reducing a fraction a/b to its lowest terms by cancelling out the greatest common divisor of a and b. In fact it can be seen that the numerator in the right hand side of (1) is equal to

$$(4x - 1)^2(x^2 + 2)(2x^4 + x^2 + 3)$$

and the denominator is equal to

$$(4x - 1)(2x^4 + x^2 + 3)$$

so that

$$\begin{aligned} f &= (4x - 1)(x^2 + 2) \\ &= 4x^3 - x^2 + 8x - 2 \end{aligned}$$

which yields

$$\frac{df}{dx} = 12x^2 - 2x + 8.$$

The simplification achieved here is quite considerable. The obvious question now is how can we carry out the simplification in general? Indeed do we *need* to factorize the numerator and denominator or is there an easier method? Again a comparison with the case for fractions suggests that we do not have to find the factorization (we discuss this in §4.7.6).

As another example suppose that we want to integrate

$$g = \frac{x^2 - 5}{x(x-1)^4}.$$

A little experience shows that such problems can often be solved by decomposing the expression into *partial fractions*. (We say that a fraction c/pq is decomposed into partial fractions if we can write it as $a/p + b/q$. The summands might themselves be split further into partial fractions. Note that such a decomposition is not always possible, e.g., $1/x^2$ cannot be so decomposed.) In our case we have

$$g = \frac{-5}{x} + \frac{5}{x-1} - \frac{5}{(x-1)^2} + \frac{6}{(x-1)^3} - \frac{4}{(x-1)^4}.$$

We may now proceed to integrate each of the summands and add the results to obtain the integral of g. Once again the decomposition process is rather tedious and we could easily make a mistake—the same applies to the final process of integration! It is therefore highly desirable to have machine assistance for such tasks. Indeed it would be very useful to be able to deal with more general expressions such as

$$\frac{x+a}{x(x-b)(x^2+c)}$$

where a, b, c are arbitrary unspecified constants. More ambitiously we might ask about the possibility of integrating expressions drawn from a large natural class. One of the major achievements of Computer Algebra has been the development of an algorithm that will take an expression from such a class and either return its integral or inform the user that no integral exists within the class.

There are many more applications of Computer Algebra some of which will be discussed as the course progresses. The overall aim is the development of algorithms (and systems implementing them) for Mathematical problem solving. As the examples given above illustrate, we do *not* restrict attention to numerical problems.

1.2 Features of Computer Algebra Systems

All the well developed Computer Algebra systems are designed to handle a large amount of traditional forms of mathematics such as that used in Engineering applications. Some systems can handle more abstract Mathematics as well. In this section we outline those capabilities that are found in the majority of well developed systems. The degree to which problems of a given kind (such as integration) can be solved will vary from one system to the next.

Interactive use

The user can use the system directly from the terminal and see results displayed on the screen. On old fashioned character terminals the results were shown in a basic 2-dimensional format; this can still be useful if a system is being used remotely over a slow connection. Systems such as Maple will produce a much more sophisticated display on modern terminals; Axiom does not yet have a sophisticated interface.

File handling

Expressions and commands can be read from files. This is clearly essential for the development of programs in the system's language. Output can also be sent to a file. The output can be in various formats or in a form which is suitable for input back to the system at some later stage.

Polynomial manipulation

Without this ability a system cannot be deemed to be a Computer Algebra system. Polynomials are expressions such as

$$4x^2 + 2x - 3$$

or

$$x^4 - x^3y + 5xy^3 - 20y.$$

Systems can also handle operations on rational expressions (i.e., expressions of the form p/q where p, q are both polynomials).

Elementary special functions

These include such basics as

- sine, cosine, tangent (and their inverses),
- natural logarithms and exponentials,
- hyperbolic sine, cosine, tangent (and their inverses).

Normally systems can differentiate and integrate expressions in these as well as apply certain simplifications such as $\sin(0) = 0$.

Arithmetic

Integer and rational arithmetic can be carried out to any degree of accuracy. (Some systems impose an upper limit but this is extremely large. Naturally the available physical storage will always impose a limit.) In floating point arithmetic the user can specify very high levels of accuracy. Normally systems will use exact arithmetic rather than resort to floating point approximations. For example 1/3 will *not* be converted to an approximation such as 0.33333 unless the user demands it. Similar remarks apply to such expressions as $\sqrt{9+4\sqrt{2}}$. If approximations are used then not only do we lose precision but might also obtain false results. For example consider the polynomials (taken from [21])

$$\begin{aligned} p &= x^3 - 8 \\ q &= (1/3)x^2 - (4/3) \end{aligned}$$

Now it can be seen that the polynomial

$$d = (1/3)x - (2/3)$$

divides both of these, i.e.,

$$\begin{aligned} p &= (3x^2 + 6x + 12)d \\ q &= (x + 2)d \end{aligned}$$

and moreover no polynomial of degree higher than that of d has this property, i.e., d is a gcd of p, q. Suppose however that we consider the polynomial

$$\hat{q} = 0.333333x^2 - 1.33333$$

which appears to be a good approximation to q. If we now use p and $\hat{q}$ to find an approximation $\hat{d}$ to d and use six digit accuracy in arithmetic operations (using a version of Euclid's algorithm for polynomial gcd's) then we obtain $\hat{d} = 0.000001$ which is nothing like a good approximation to d.

Most systems can also carry out complex number arithmetic.

Differentiation

This can be applied to expressions involving polynomials and special functions which are recognized by the system. Partial derivatives can also be computed.

Integration

Expressions involving polynomials and special functions can usually be integrated. The expressions can be quite complicated—the system does much more than just look the result up in a table of standard integrals. Integration is one of the most difficult tasks carried out by a Computer Algebra system. All systems have some problems with definite integration and occasionally produce wrong answers (indefinite integration is easier). It is important to understand that Computer Algebra systems treat integration from an algebraic rather than an analytic viewpoint. For most of the time there is no difference but for certain situations the answer returned is correct only in the algebraic sense (e.g., the integral of a continuous function might be returned as a non continuous one). This fact is not appreciated as widely as it ought to be.

Solving equations

Systems of simultaneous linear equations with numeric or symbolic coefficients can be solved exactly.

Arbitrary polynomial equations up to degree 4 can also be solved exactly. For higher degree polynomials no general method is possible (this is a mathematical theorem), however certain types of equations can still be handled.

First and second order differential equations can usually be solved as well.

Substitution

It is very useful to be able to substitute an expression in place of a given variable. This allows the user to build up large expressions or alter existing ones in a controlled manner. All systems have this facility.

Matrices

All systems provide basic operations on matrices. The entries of the matrices do not have to be numerical or even specified. Systems provide methods for initializing large matrices or for building special types such as symmetric matrices.

Mathematical structures

Sophisticated systems such as Axiom enable the user to work (and indeed define for themselves) mathematical structures of many kinds. Thus a user can work with groups, rings, fields, algebras etc. The underlying system supplies the appropriate operations on elements and many algorithms for examining the structure of particular examples thus making it easy to test conjectures of many kinds.

Defining new rules and procedures

Users can define their own functions for tasks which are not covered by the built-in operations. It is generally possible to define certain properties of the function such as its derivative or integral (where these make sense). Rules for simplifying expressions which involve the function can usually be specified as well.

Graphics

Most systems can produce simple plots. With increasing computational power and better displays, most systems have introduced highly developed capabilities for plotting mathematical functions in 2 or 3 dimensions (in black and white or colour and specified illumination for 3D).

1.3 Syntax of Associated Languages

These are mostly Algol-like, with more recent influences. A little experience with a language such as C provides sufficient background for a quick mastery of a Computer Algebra system's language. All the general purpose systems allow users to define their own procedures thus extending the system.

Most systems provide a notion of type but this is at times an add on to the original design and the default mode is to work without types. By contrast types are at the core of Axiom.

1.4 Data Sructures

All systems supply a large number of built in data structures, ranging rom purely mathematical ones such as polynomials, infinite series or matrices to general ones such as lists, arrays, hash tables or trees. There are over 40 kinds of aggregate data structures in Axiom.

2 Brief Introduction to Axiom

2.1 Background

The brief history of Axiom at http://www.axiom-developer.org can be summarised as follows. Axiom is a system for symbolic mathematical computation which started in 1971 as an IBM project under the name Scratchpad. In the 1990's it was sold to the Numerical Algorithms Group (NAG) and marketed under the name Axiom. However it did not become a financial success and was withdrawn from the market in October 2001. Shortly after this it was released as free software with the source being available at https://github.com/daly/axiom. For more details see http://www.axiom-developer.org and the introduction to the free book at http://www.axiom-developer.org/axiom-website/bookvol1.pdf (which you should download). In fact there are now at least two open source versions of Axiom, this course will follow the one given at the preceding links.

2.2 Using the System

Simply log on to a linux machine and type `axiom`, alternatively download and install it on your own machine. We have tested it on linux and Mac, the Axiom site also has instructions for Windows but we have not tested these.

2.3 Brief Overview of Axiom's Structure

Every object in Axiom belongs to a unique *domain of computation* or *domain* for short[1]. The name of each domain starts with a capital letter, e.g., `Integer`. Domains can themselves be parts of larger ones, so `PositiveInteger` is contained in `Integer` which is contained in `Polynomial(Integer)`. Operations names (e.g., +) can be applied in common whenever they make sense, just as in standard Mathematics. Domains themselves have types which are called *categories*[2]. The role of categories is to ensure that types make sense, so we can build matrices of integers but not of hash tables. Domains can be combined in any meaningful way to build new ones. For example

```
D:List(Record(key:String,val:DMP([x,y,z],FRAC INT)))
```

declares the variable `D` to take on values that are lists of records with two entries dereferenced by `key` (a string) and `val` (a polynomial in x, y, z with rational coefficients). So if we set

```
D:=[["parabola", (x/a)^2+(y/b)^2-1],["sphere",x^2+y^2+z^2-1]]
```

then `D.2.key` returns `"sphere"` while `D.2.val` returns $x^2 + y^2 + z^2 - 1$. Compare, by contrast, the session

```
(1) -> P:UP(x,Table(Integer,String))
P:UP(x,Table(Integer,String))
 1) ->
   UnivariatePolynomial(x,Table(Integer,String)) is not a valid type.
(1) -> P:UP(x,Integer)
P:UP(x,Integer)
                                                       Type: Void
```

Axiom detects that the first type declaration does not make sense and rejects it. The second one is meaningful so it is accepted.

For a more extended discussion see the chapter *A Technical Introduction to AXIOM* in the free book http://www.axiom-developer.org/axiom-website/bookvol1.pdf. One point to note is that the example under *1.4 Operations Can Refer To Abstract Types* should not be typed into Axiom directly; it seems to be in Axiom's language for developers but this is not made clear.

[1]These should not be confused with the special types of rings also called domains, that we will meet later on.
[2]These should not be confused with the objects of study in Category Theory, a branch of mathematics.

2.4 Basic Language Features

Like other computer algebra systems Axiom has its own programming language. Most of its constructs will be familiar from other imperative languages, e.g., loops, conditional statements, recursion. The format of loops is quite flexible allowing for convenient expression of ideas. Here are some examples:

```
(1) -> for i in 1..5 repeat print(i^i)
   1->
   4
   27
   256
   3125
                                                                    Type: Void
(2) -> L:=[1,2,3,4,5]
L:=[1,2,3,4,5]
(2) ->
   (2)  [1,2,3,4,5]
                                                  Type: List(PositiveInteger)
(3) -> for i in L repeat print(i^i)
for i in L repeat print(i^i)
   1->
   4
   27
   256
   3125
                                                                    Type: Void
(4) -> L:=[i for i in 1..10]
L:=[i for i in 1..10]
(4) ->
   (4)  [1,2,3,4,5,6,7,8,9,10]
                                                  Type: List(PositiveInteger)
(5) -> for i in L repeat print(i^i)
for i in L repeat print(i^i)
   1->
   4
   27
   256
   3125
   46656
   823543
   16777216
   387420489
   10000000000
                                                                    Type: Void
```

In the second and third examples `L` is a list which we create in two different ways (lines `(2)` and `(4)`). There are many data structures built in with supporting operations.

As can be seen from the session above Axiom is a typed language. Indeed it was an early adopter of the notion of inheritance so that operations common to various types can become more specialised as the data type does so. However the user is not forced to declare the type of everything, the system will try to deduce the information, thus freeing users to work fluently. At times the system is unable to deduce the type (either because of insufficient information or it is too complicated); if so the user is informed and has the opportunity to supply the information. Here is an example of a function definition from Axiom:

```
(1) -> f(0)==0
f(0)==0
                                                               Type: Void
(2) -> f(1)==1
f(1)==1
                                                               Type: Void
(3) -> f(n)==f(n-1)+f(n-2)
f(n)==f(n-1)+f(n-2)
                                                               Type: Void
(4) -> f(10)
f(10)
   Compiling function f with type Integer -> NonNegativeInteger
   Compiling function f as a recurrence relation.
                                                           ^
... (a lot of messages deleted [KK])
   (4)  55
                                                    Type: PositiveInteger
(5) -> f(1000)
f(1000)
(5) ->
   (5)
  434665576869374564356885276750406258025646605173717804024817290895365554179_
   490518904038798400792551692959225930803226347752096896232398733224711616429_
   96440906533187938298969649928516003704476137795166849228875
                                                    Type: PositiveInteger
(6) ->
```

Clearly `f` is the Fibonacci function. We define it piecewise just as would be done in a maths book. The sign `==` is delayed assignment, in effect it tells Axiom not to evaluate anything at this time but to use the right hand side when required (it does carry out various checks, e.g., that types make sense). On the first call Axiom gathers together the piecewise definition of `f` and tries to compile it, in this case successfully. During compilation it carries out some optimisations so that in this case computing `f` is done efficiently (a naive version would involve exponentially many recursive calls). If the function cannot be compiled (e.g., the system cannot determine the types of the arguments) it is interpreted instead.

Finally, Axiom makes polymorphic functions available so that the same code will be compiled as appropriate for different data types. Here is a simple example:

```
(1) -> first(L)==L.1
first(L)==L.1
```

```
                                                          Type: Void
(2) -> first([1,2,3])
first([1,2,3])
   Compiling function first with type List(PositiveInteger) ->
      PositiveInteger

... (a lot of messages deleted [KK])
   (2)  1
                                               Type: PositiveInteger
(3) -> first([1,x,x^2])
first([1,x,x^2])
   Compiling function first with type List(Polynomial(Integer)) ->
      Polynomial(Integer)

... (a lot of messages deleted [KK])
   (3)  1
                                           Type: Polynomial(Integer)
```

On subsequent calls of, e.g., `first([1,2,3])` and `first([1,x,x^2])` no further compilation is necessary; the appropriate compiled code is used.

2.5 Basic Literature

The following is a short list of references together with some comments. A much longer list is given at the end of the notes.

1. B. Buchberger, G. E. Collins and R. Loos (editors), *Symbolic and Algebraic Computation. Computing Supplementum 4*, Springer (1983).

   This book is a collection of articles written by well known people in the field. Many topics are covered, some of them in great detail. It is essential reading for somebody who wants to pursue the field further although certain parts of it are now out of date.

2. J. H. Davenport, Y. Siret and E. Tournier, *Computer Algebra; systems and algorithms for algebraic computation*, Academic Press, (1988).

   This is one of the first text books on the area (and the only one at a relatively low price). The coverage is quite comprehensive and includes many advanced topics. It does not deal with implementation issues in any depth but some discussion is included. There is an appendix giving some of the algebraic background which is essential for the subject. There is also an annex which can act as a basic reference manual for the system REDUCE.

   Unfortunately there is a tendency to skip details—this is unavoidable in a book of reasonable size with such a breadth of coverage. More seriously there are quite a few misprints although most are fairly obvious.

3. K. O. Geddes, S. R. Czapor and G. Labahn, *Algorithms for Computer Algebra*, Kluwer Academic Publishers (1992).

   This book is comprehensive and is a good reference for the area. The authors are long-standing members of the Maple group, however the book itself is not Maple centred rather it is concerned with the core algorithms of the area. Unfortunately it is very expensive, however there is a copy in the library.

4. R. Zippel, *Effective Polynomial Computation*, Kluwer Academic Publishers (1993).

   This book focuses on the core polynomial operations in Computer Algebra and studies them in depth. It is a good place to go to for extra detail (both theoretical and practical). Unfortunately it has a fairly large number of misprints most of which are minor (a list is available from K. Kalorkoti).

5. D. E. Knuth, *Seminumerical Algorithms*, (Second Edition), Addison-Wesley (1981).

   Chapter 4 gives a comprehensive treatment of topics in arithmetic and some aspects of polynomial arithmetic. Although the course will not go into such staggering detail every serious student of computing should consult this book. Several copies are available from the library.

6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms.* McGraw-Hill, 2002 (now in its third edition, published September 2009).

   This book is not specifically related to Computer Algebra, however it is an excellent source for many algorithms and general background. There are also several chapters on basic Mathematics.

7. D. R. Stoutemyer, Crimes and misdemeanors in the computer algebra trade, *Notices of the AMS*, Sept. 1991, 701-705.

   This paper describes some of the ways in which computer algebra systems can produce wrong results. Ideally it (or something like it) should be read by every user of such systems. A crude but effective summing up of some of the paper is that having a computer does not give you the right to throw your brain away—quite the opposite is true. The paper concentrates on analysis rather than algebra, surprisingly the author does not discuss newer systems such as Scratchpad II (now called AXIOM) which try to be more sensible about the domain of computation. (In this course we will be concerned almost exclusively with the algebraic viewpoint.)

8. F. Winkler, *Polynomial Algorithms in Computer Algebra*, Springer (1996).

   This book is devoted to algebraic aspects of the subject, ranging from basic topics to algebraic curves.

9. J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, Cambridge University Press (1999).

   This book gives comprehensive coverage to the basics of the subject, aiming at a complete presentation of mathematical underpinnings, analysis of algorithms and development of asymptotically fast methods.

3 Computer Algebra Systems

Below is a list of some systems in addition Axiom.

- Macsyma: started at MIT under the direction of J. Moses. A very large system. A major drawback is that it has had too many implementors with no common style and there is no source code documentation. It is therefore difficult to build on the system using the source code. Macsyma was marketed by Symbolics Inc. for many years. It is now available as Maxima under the GPL license.

- REDUCE: started at the University of Utah under the direction of A. Hearn later of the Rand Corporation, Santa Monica. It has many special functions for Physics and is now available under a BSD license.

- ALDES/SAC-II (then SACLIB): under the direction of G. E. Collins, University of Wisconsin, Maddison (later at RISC, University of Linz). Originally implemented in Fortran and now in C, the user languages is called ALDES. The system is not very user friendly, it is intended for research in Computer Algebra, many basic algorithms were implemented in it for the first time. It is still being developed.

- muMath/Derive: developed specifically for micros with limited memory by Soft Warehouse later owned by Texas Instruments. Derive was menu driven and replaced muMath. It was discontinued in June 2007.

- Maple: started at the University of Waterloo, Canada, in the early 1980's. It is a powerful and compact system which is ideal for multi-user environments and is still under very active development.

- Mathematica: this is a product of Wolfram Research Inc., based in Ilinois, and is the successor to SMP. It was announced with a great deal of publicity in Summer 1988. One very striking feature at the time was its ability to produce very high quality graphics. Its programming language attempts to mix just about all possible paradigms, this does not seem like a good idea to me. It is still been developed.

- Sage: this is not strictly speaking a computer algebra system but a wrapper for various free systems. The front page of the web site (http://www.sagemath.org) states 'SageMath is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more. Access their combined power through a common, Python-based language or directly via interfaces or wrappers.'

There are various other more specialised systems, e.g., GAP and Magma for group theory (GAP is available through Sage).

4 Basic Structures and Algorithms

4.1 Algebraic Structures

We describe some abstract algebraic structures which will be useful in defining various concepts. At first they appear rather dry and daunting but they have their origins in concrete applications (for example groups and fields were shown by Galois to have a deep connection with the problem of solving polynomial equations in radicals when the concepts were still just being formed—see Stewart [57]). We shall use the following standard notation:

1. $\mathbb{Z}$, the integers,
2. $\mathbb{Q}$, the rationals,
3. $\mathbb{R}$, the reals,
4. $\mathbb{C}$, the complex numbers,
5. $\mathbb{Z}_n$ the integers modulo n where $n > 1$ is a natural number.

A *binary operation* on a set R is simply a function which takes two elements of R and returns an element of R, i.e., a function $R \times R \to R$. We normally use infix operators such as $\circ$, $\times$, $*$, $+$ for binary operations. For example addition, subtraction and multiplication are all binary operations on $\mathbb{R}$ (division is not a binary operation on $\mathbb{R}$ because it is undefined whenever the second argument is 0). A binary operation $\circ$ on R is *commutative* if

$$x \circ y = y \circ x, \quad \text{for all } x, y \in R.$$

We say that $\circ$ is *associative* if

$$(x \circ y) \circ z = x \circ (y \circ z), \quad \text{for all } x, y, z \in R.$$

Thus the addition of numbers is both commutative and associative, while subtraction is neither. On the other hand matrix multiplication is associative but *not* commutative (except in the case of 1×1 matrices with entries whose multiplication is commutative).

Associative operations make notation very easy because it can be shown that any valid way of bracketing the expression

$$x_1 \circ x_2 \circ \cdots \circ x_n$$

leads to the same result. (The proof of this makes a useful exercise in induction.)

A *ring* is a set R equipped with two binary operations $+$, $*$ called *addition* and *multiplication* with the following properties:

1. $+$ is associative,
2. $+$ is commutative,
3. there is a fixed element 0 of R such that $x + 0 = x$ for all $x \in R$,
4. for each element x of R there is an element $y \in R$ such that $x + y = 0$, (i.e., x has an *additive inverse*),

5. $*$ is associative,

6. for all $x, y, z \in R$ we have
$$x * (y + z) = x * y + x * z,$$
$$(x + y) * z = x * z + y * z,$$
i.e., $*$ is both left and right *distributive* over $+$.

It is important to bear in mind that R need not be a set of numbers and even if it is the two binary operations might be different from the usual addition and multiplication of numbers (in which case we use symbols which are different from $+$ and $*$ in order to avoid confusion). The element 0 is easily seen to be unique and is called the *zero* of the ring. We prove this claim by the usual method: suppose there are two elements 0, $0'$ that satisfy the axioms for zero. Then

$$\begin{aligned} 0' &= 0' + 0, && \text{by axiom 3} \\ &= 0 + 0', && \text{by axiom 2} \\ &= 0, && \text{since } 0' \text{ satisfies axiom 3 by assumption.} \end{aligned}$$

Thus $0' = 0$ as claimed. Moreover every x has exactly one additive inverse which is denoted by $-x$. We normally write $x - y$ instead of the more cumbersome $x + (-y)$. The axioms immediately imply certain standard identities such as $x * 0 = 0$ for all x. To see this note that $x * 0 = x * (0 + 0) = x * 0 + x * 0$. Now adding $-(x * 0)$ to both sides we obtain $0 = x * 0$ as required. Similarly $0 * x = 0$ for all x (remember that $*$ is not assumed to be commutative so this second identity does not follow immediately from the first).

It is worth noting that multiplication in rings is frequently denoted by juxtaposition. Thus we write xy rather than $x * y$.

The archetypal ring is $\mathbb{Z}$ with the usual addition and multiplication. Other examples include:

1. $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ with the usual addition and multiplication.

2. $2\,\mathbb{Z}$ the set of all even integers with the usual addition and multiplication.

3. $\mathbb{Z}_n$ the integers modulo n where $n > 1$ is a natural number. Here addition and multiplication are carried out as normal but we then take as result the remainder after division by n. Alternatively we view the elements of $\mathbb{Z}_n$ to be the integers and use ordinary addition and multiplication but interpret equality to mean that the two numbers involved have the same remainder after division by n. More accurately the elements are equivalence classes where two numbers are said to be equivalent if they have the same remainder when divided by n (this is the same as saying that their difference is divisible by n). The classes are $\{kn + r \mid k \in \mathbb{Z}\}$, for $0 \le r \le n - 1$. For simplicity we denote such a class by r. We could use any other element of each class to represent it. A simple exercise shows that doing operations based on representatives is unambiguous, that is we always get the same underlying equivalence class.

   We will see later on that this is a particular case of a more general construction that gives us a new ring from the ingredients of an existing one. Just to give a hint of this, note that the set $n\,\mathbb{Z} = \{nm \mid m \in \mathbb{Z}\}$ is itself a subring of $\mathbb{Z}$ (actually all we need is that it is a two sided ideal but this will have to wait). Note that n can be arbitrary for this construction. Now if we define the relation $\sim$ on $\mathbb{Z}$ by
$$a \sim b \Leftrightarrow a - b \in n\,\mathbb{Z}$$

we obtain an equivalence relation and use $\mathbb{Z}/n\mathbb{Z}$ to denote the equivalence classes. Let us denote the equivalence class of an integer r by $[r]$; note that by definition $[r] = \{mn+r \mid m \in \mathbb{Z}\}$.We can turn $\mathbb{Z}/n\mathbb{Z}$ into a ring by defining $+$ and $*$ on equivalence classes by

$$[r] + [s] = [r + s],$$
$$[r] * [s] = [r * s].$$

There is a subtlety here, we must show that the operations are well defined, i.e., if $[r_1] = [r_2]$ and $[s_1] = [s_2]$ then $[r_1] + [s_1] = [r_2] + [s_2]$ and $[r_1] * [s_1] = [r_2] * [s_2]$. This is quite easy to do but we will leave it for the general case (this is where we need the substructure $n\mathbb{Z}$ to be a subring, in fact just a two sided ideal).

The use of square brackets to denote equivalence classes is helpful at first but soon becomes tedious. In practice we drop the brackets and understand from the context that an equivalence class is being denoted. This is an important convention to get used to; the meaning of a piece of notation depends on the structure in which we are working. Thus if we are working over the integers then 3 denotes the familiar integer ("three") but if we are working in the integers modulo 6 then it denotes the set of integers $\{6m+3 \mid m \in \mathbb{Z}\}$. Finally, we have used $\mathbb{Z}_n$ as an abbreviation for $\mathbb{Z}/n\mathbb{Z}$. This is quite standard but in more advanced algebra the notation is not so good because it conflicts with the notation of another very useful concept (localization, since you ask).

4. Square matrices of a fixed size with integer entries. Here we use the normal operations of matrix addition and matrix multiplication.

   Again this is a particular case of a general construction, we can form a ring by considering square matrices with entries from any given ring.

5. Let S be any set and $P = \mathcal{P}(S)$ the *power set* of S (i.e., the set of all subsets of S). This forms a ring where

   - Addition is symmetric difference, i.e., $A+B$ is $A \cup B - A \cap B$ (this consists of al elements that are in one of the sets but not both).
   - Multiplication is intersection, i.e., $A * B$ is $A \cap B$.

   The empty set plays the role of 0. This is an example of a *Boolean ring*, i.e. $x * x = x$ for all x.

Note that in all our examples, except for matrices, multiplication is also commutative. A ring whose multiplication is commutative is called *commutative.* Notice further that in all our examples, except for the ring $2\mathbb{Z}$ (generally $n\mathbb{Z}$ for $n > 1$), we have a special element 1 with the property that

$$1x = x1 = x, \quad \text{for all } x \in R.$$

An element with this property is called a (multiplicative) *identity* of the ring. In fact it is easily seen that if such an identity exists then it is unique. (There is a clear analogy with 0 here which is an additive identity for the ring—however the existence of 0 in a ring is required by the axioms).

While the definition of rings is clearly motivated by the integers, certain 'strange' things can happen. For example in $\mathbb{Z}_6$ we have $2 \times 3 = 0$ even though $2 \neq 0$ and $3 \neq 0$ in $\mathbb{Z}_6$ (i.e., 6 does not divide 2 or 3). Matrix rings also exhibit such behaviour.

Pursuing the numerical analogy one level up we see that in the ring $\mathbb{Q}$ every non-zero element has a *multiplicative inverse*, i.e., for each $x \neq 0$ there is a $y \in \mathbb{Q}$ such that $xy = yx = 1$. Of course in general the notion of a multiplicative inverse only makes sense if the ring has an identity. However even if the ring does have an identity there is no guarantee that particular elements will have inverses. For example in the ring of 2×2 matrices with integer entries we observe that

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

does not have an inverse. Note that

$$\begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$$

also does not have an inverse in our ring although it does have an inverse in the ring of 2×2 matrices with entries from $\mathbb{Q}$. It is easy to show that if an element x of a ring R has an inverse then it is unique (and is normally denoted by x^{-1}).

Exercise 4.1 *Write the multiplication tables of* $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_4$, $\mathbb{Z}_5$, $\mathbb{Z}_6$ *and decide which elements have an inverse for each of these rings. Can you spot a pattern?*

Observations such as the above lead to the notion of *fields*. A field is a ring with the extra properties

1. there is a multiplicative identity which is *different* from 0,
2. multiplication is commutative,
3. every non-zero element has an inverse,

This brings the behaviour of our abstract objects much closer to that of numbers. However 'strange' things can still happen. For example it is easy to see that $\mathbb{Z}_2$ is a field but in here we have $1+1 = 0$. Certain other types of behaviour are definitely excluded, for example if $xy = 0$ then $x = 0$ or $y = 0$; for if $x \neq 0$ then it has a multiplcative inverse and so we have

$$y = (x^{-1}x)y = x^{-1}(xy) = x^{-1}0 = 0.$$

Examples of fields include:

1. $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ all with the usual operations.
2. $\mathbb{Z}_p$ when p is a prime.

Exercise 4.2 *Show that* $\mathbb{Z}_n$ *is not a field whenever* n *is not a prime number. Treat* $n = 1$ *as a special case and then consider composite* n *for* $n > 1$. *(Hint: in a field there cannot be non-zero elements* x, y *such that* $xy = 0$, *as shown above.)*

Exercise 4.3 *We have seen that in certain fields it is possible to obtain 0 by adding 1 to itself sufficiently many times. The* characteristic *of a field is defined to be 0 if we can never obtain 0 in the way described and otherwise it is the least number of times we have to add 1 to itself in order to obtain 0. Show that every non-zero characteristic is a prime number.*

Fields bring us closer to number systems such as $\mathbb{Q}$ and $\mathbb{R}$ rather than $\mathbb{Z}$. One reason for this is because in a field every non-zero element must have an inverse. We can define other types of rings which don't go this far. An *integral domain* (or *ID*) is a commutative ring with identity (different from zero) with the property that if $xy = 0$ then $x = 0$ or $y = 0$. (Of course every field is an ID but not conversely.) A still more specialized class of rings consists of the *unique factorization domains* (or *UFD's*). We shall not give a formal definition of these but explain the idea behind them. It is well known that every non-zero integer n can be factorized as a product of prime numbers:

$$n = p_1 p_2 \cdots p_s.$$

(Here we allow the possibility of negative 'primes' such as -3.) Moreover if we have another factorization of n into primes

$$n = q_1 q_2 \cdots q_t,$$

then

1. $s = t$, and

2. there is a permutation π of $1, 2, 3, \ldots, s$ such that $p_i = \epsilon_i q_{\pi(i)}$ for $1 \leq i \leq n$ where $\epsilon_i = \pm 1$.

Note that ± 1 are the only invertible elements of $\mathbb{Z}$. The notion of a prime or rather *irreducible* element can be defined for ID's: a non-zero element a of an ID is *irreducible* if it does not have an inverse and whenever $a = bc$ then either b or c has a multiplicative inverse.

Note that if u is an invertible element of a ring and a is any element of the ring then we have $a = u(u^{-1}a)$ which is a trivial 'factorization.' Such a 'factorization' tells us nothing about a. For example writing $6 = -1 \times -6$ contains no information about 6. Writing $6 = 2 \times 3$ gives us genuine information about 6 since neither factor is invertible in $\mathbb{Z}$ and so the factorization is not automatic. This discussion also shows that factorization is not an interesting concept for fields; every non-zero element is invertible. You have known this for a long time though perhaps not in this abstract setting; you would never consider if a rational or a real number has a factorization.

A UFD is an ID in which every non-zero element has a factorization in finitely many irreducible elements and this factorization is unique in a sense similar to that for integer factorization. Obviously $\mathbb{Z}$ is a UFD. Also every field is a UFD for the trivial reason that in a field there are no primes (every non-zero element has an inverse) so that each non-zero element factorizes uniquely as its (invertible) self times the empty product of irreducible elements! In a UFD we have a very important property. Let us say that b *divides* a, written as $b \mid a$, if $a = bc$ for some c. It can be shown that if p is irreducible and $p \mid ab$ then either $p \mid a$ or $p \mid b$ (in fact this property is usually taken as the definition of a *prime* element in a ring—the observation then says that in a UFD irreducible elements are primes, see Exercise 4.4 for an example of a ring in which irreducible elements need not be primes).

Exercise 4.4 *This exercise shows that unique factorization really is a special property which can fail even in rings which seem quite innocuous. Let* $\mathbb{Z}[\sqrt{-5}]$ *be the set of all elements*

$$\{\, a + b\sqrt{-5} \mid a, b \in \mathbb{Z} \,\}.$$

It is clear that $\mathbb{Z}[\sqrt{-5}]$ *becomes a ring under the usual arithmetic operations—indeed it is an ID since it just consists of numbers with the usual operations.*

1. *Show that* 3, $2+\sqrt{-5}$ *and* $2-\sqrt{-5}$ *are all irreducible elements of* $\mathbb{Z}[\sqrt{-5}]$.

   So for this part you need to show that, e.g., if $3 = (a_1 + b_1\sqrt{-5})(a_2 + b_2\sqrt{-5})$ *then one of the factors is invertible. You can make life easier by noting two things: firstly we have* $(a+b\sqrt{-5})(a-b\sqrt{-5}) = a^2+5b^2$. *Secondly* $a_1 + b_1\sqrt{-5} = a_2 + b_2\sqrt{-5}$ *if and only if* $a_1 = a_2$ *and* $b_1 = b_2$, *this follows from the fact that* $\sqrt{-5}$ *is not a real number (all we need is that it is not rational).*

2. *Observe that* $3 \times 3 = (2+\sqrt{-5})(2-\sqrt{-5})$ *so that* $3 \mid (2+\sqrt{-5})(2-\sqrt{-5})$. *Show that 3 does not divide either of* $2+\sqrt{-5}$ *or* $2-\sqrt{-5}$ *(remember that everything takes place inside* $\mathbb{Z}[\sqrt{-5}]$*).*

3. *Deduce that the ID* $\mathbb{Z}[\sqrt{-5}]$ *is not a UFD.*

4.2 Greatest Common Divisors

Let a, b be elements of an integral domain D. We call $d \in D$ a *common divisor* of a, b if $d \mid a$ and $d \mid b$. Such a common divisor is called a *greatest common divisor* (gcd) if whenever c is also a common divisor of a, b then $c \mid d$. (Intuitively speaking a gcd contains all the common factors of the two elements concerned, so long as at least one of them is non-zero.)

Note that the preceding definition makes sense even if $a = b = 0$. In this case everything (including 0) divides a and b so is a candidate to be a gcd. However only 0 can actually be a gcd for if $c \neq 0$ then $0 \nmid c$: that is there is no $c' \in D$ such that $c = c' \cdot 0$. Hence the second condition for a gcd is satisfied only by 0 and this is the unique gcd in this case. Moreover if $a \neq 0$ or $b \neq 0$ then 0 cannot be a gcd for then either $0 \nmid a$ or $0 \nmid b$. Hence the only time 0 is a gcd is when both inputs are themselves 0. It is worth noting how this definition is more general than the standard one for the gcd of two integers: there we normally say that the gcd is the largest integer that divides both inputs, but if they are both 0 there is no such largest integer. In the case when at least one input is not 0 the two definitions agree (provided we insist the gcd is positive). Clearly the problem of finding the gcd is trivial when both inputs are 0 so in developing algorithms we will assume that at least one input is not zero.

Suppose that d is a gcd of a, b and u is any invertible element of D. Then it is easy to see that ud is also a gcd of a, b (we have $a = rd$, $b = sd$ for some $r, s \in D$ and so $a = (ru^{-1})(ud)$, $b = (su^{-1})(ud)$). In fact this level of ambiguity is all there is. First of all if the gcd is 0 then this is the only candidate so the claim is clearly true. If 0 is not the gcd, assume that d_1 and d_2 are two gcd's of a, b then there are $u_1, u_2 \in D$ such that $d_1 = u_2 d_2$ and $d_2 = u_1 d_1$, since $d_1 \mid d_2$ and $d_2 \mid d_1$. Now we have $d_1 = u_2 d_2 = u_2 u_1 d_1$ and since D is an integral domain and $d_1 \neq 0$ we may deduce that $u_1 u_2 = 1$, i.e., u_1 and u_2 are invertible elements of D.

In general there is no guarantee that gcd's exist. However if D is a UFD then we can be certain of their existence. In such cases we use $\gcd(a, b)$ to denote a gcd of a, b bearing in mind that in general this does not denote a unique element (but all the possibilities are related to each other as discussed above). This is not normally a problem because when the notation is used we are willing to accept any of the possible gcd's. In some cases we can pick a particular candidate, by insisting on some extra property, so that $\gcd(a, b)$ denotes a unique element: for example in the case of the integers there are only two candidates one of which is positive and the other negative (recall that the only invertible elements of $\mathbb{Z}$ are 1 and -1). Here we make $\gcd(a, b)$ unique by choosing the positive possibility. One convention that we shall always use is that whenever the gcd of two elements is an invertible element of D then we will normalize it to be 1, the multiplicative identity (of course this is fine since if u is a gcd of a, b and u is invertible then $uu^{-1} = 1$ is also a gcd of a, b).

In general we say that a, b are *coprime* (or *relatively prime*) if $\gcd(a, b) = 1$. It is easy to show that if a, b are coprime and $b \mid ac$ then $b \mid c$. Under the same assumption, if $a \mid c$ and $b \mid c$ then $ab \mid c$.

We shall discuss the notion of gcd's in more detail for those cases of particular interest to us. In each of these cases it is possible to give an alternative equivalent definition but it is useful to know that a unified treatment is possible.

4.3 Canonical and Normal Representations

A representation in which there is a one to one correspondence between objects and their representations is often termed *canonical*. When using a canonical representation, equality of objects is the same as equality of representations. Clearly this is a desirable feature. Unfortunately it is not always possible to find canonical representations in an effective way. This might be because the problem is computationally unsolvable (remember the halting problem) or computationally intractable.

The basic objects of Computer Algebra come from Mathematical structures in which we have a notion of a zero object and of subtraction. Under these conditions we can look for a weaker form of representation—one in which the object 0 has exactly one representation while other objects might have several. Such representations are called *normal*. If we have normal forms then we can test equality of two objects a, b by computing the representation of $a - b$ and testing it for equality (as a representation) with the unique form for 0.

4.4 Integers and Rationals

The great nineteenth century mathematician Kronecker stated 'God made the integers: all the rest is the work of man'. Some people would question Kronecker's assertion that God exists while others would claim that God's mistake was to add man to the integers. In any case computer algebra systems must be able to deal with true integers rather than the minute subrange offered by many languages.

4.5 Integers

In representing non-negative integers we fix a base and then hold the digits either in a linked list or an array (for the latter option we need a language such as C that enables us to grow arrays at runtime). Leading zeros are suppressed (i.e., 011 loses the 0) in order to have a canonical representation (we need to have a special case for 0 itself). The base chosen is usually as large as possible subject to being able to fit the digits into a word of memory and leave one bit to indicate a carry in basic operations (otherwise the carry has to be retrieved in ways that are too machine dependent). Usually the base is a power of 2 or of 10. Powers of 2 make some operations more efficient while powers of 10 make the input and output of integers more efficient.

Knuth [39] gives a great deal of detail on algorithms for the basic arithmetic operations on large integers. Here we look only at multiplication. The classical method uses around n^2 multiplications of digits and n additions so its cost is proportional to n^2. We can do better than this by using Karatsuba's algorithm [36]. For simplicity we consider two integers x, y represented in base B

both of length n and put

$$x = aB^{n/2} + b,$$
$$y = cB^{n/2} + d,$$

(with appropriate adjustment if n is odd). Then

$$xy = acB^n + (bc + ad)B^{n/2} + bd.$$

Instead of doing one multiplication of integers of length n (costing $\sim n^2$ time we do four multiplications of integers of length $n/2$ (which costs $\sim 4(n/2)^2 = n^2$). *But* we don't have to compute the four products since

$$bc + ad = (a + b)(c + d) - ac - bd$$

and so we only need three products of integers of length $n/2$ (or possibly $n/2 + 1$) and then some shifts and additions (costing $\sim n$ time). Using the algorithm recursively we find that the time $T(n)$ taken to multiply two integers of length n is given by the recurrence

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ 3T(n/2) + \Theta(n), & \text{if } n > 1. \end{cases}$$

The solution to this is

$$T(n) = \Theta(n^{\log_2 3}).$$

Note that $\log_2 3 \approx 1.67$. We therefore have

Theorem 4.1 *Multiplication of integers of length n can be done in time proportional to $n^{\log_2 3}$.*

Knuth [39] shows that the method can be refined to prove

Theorem 4.2 *For each $\epsilon > 0$ there is an algorithm for multiplying integers of length n in time proportional to $n^{1+\epsilon}$.*

Many Computer Algebra systems use the basic Karatsuba algorithm (the refined version is not really practical). This pays off for numbers of sufficiently many digits (this is implementation dependent) and so the classical method is used for smaller numbers. (Bear in mind that what constitutes a 'digit' depends on the base.) An asymptotically faster method is due to Schönhage and Strassen [56] which takes time proportional to $n \log n \log\log n$. Unfortunately the constant involved is large and so Computer Algebra systems do not use this method.

4.6 Fractions

These can be represented in any structure that can hold two integer structures (or pointers to them). Thus a fraction a/b could be denoted generically as $\langle a, b\rangle$. We normally insist on the following conditions:

1. The second integer is positive so the sign of the fraction is determined by that of a.
2. The gcd of the two integers is 1.

The gcd of a, b is undefined if both a, b are 0 and is the largest positive integer d that divides both a and b otherwise (d always exists—why?). This method gives a canonical form and has the advantage of keeping the size of the two integers as small as possible. We discuss the computation of gcd's in §4.6.1.

Carrying out arithmetic on rational numbers is straightforward but even here a little thought can improve efficiency. Let a/b and c/d be fractions in their lowest terms with $b, c > 0$. Then

$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd} = \frac{ac/\gcd(ac, bd)}{bd/\gcd(ac, bd)}$$

and the last expression is in canonical form. However the fact that $\gcd(a, b) = \gcd(c, d) = 1$ means that

$$\gcd(ac, bd) = \gcd(a, d)\gcd(b, c).$$

If we put

$$\begin{aligned} d_1 &= \gcd(a, d) \\ d_2 &= \gcd(b, c) \end{aligned}$$

then

$$\frac{(a/d_1)(c/d_2)}{(b/d_2)(d/d_1)}$$

gives us the canonical form for a/b. This method requires two gcd computations. However this is not slower than the previous method. We will see in §4.6.1 that the number of 'basic steps' of the usual algorithm for computing gcd's is essentially proportional to the logarithm of the largest of its inputs. Thus the number of basic steps needed to compute d_1 and d_2 is roughly the same as is required to compute $\gcd(ac, bd)$. Each basic step operates on numbers whose size decreases with each step. It follows that the second method is potentially faster because it works with smaller numbers than the first.

Division is essentially identical to multiplication.

For addition and subtraction let

$$\frac{a}{b} \pm \frac{c}{d} = \frac{p}{q}$$

where p/q is in canonical form. Generally it is best to compute p', q' by

$$\begin{aligned} p' &= a\frac{d}{\gcd(b, d)} + c\frac{b}{\gcd(b, d)} \\ q' &= \frac{bd}{\gcd(b, d)} \end{aligned}$$

and then

$$p = p'/\gcd(p', q'), \qquad q = q'/\gcd(p', q').$$

Exercise 4.5 *Computing q' by $u := bd$, $v := \gcd(b, d)$, $q' = u/v$ is a bad idea. Suggest a better method (we assume that $\gcd(b, d)$ is already known).*

4.6.1 Euclid's Algorithm for the Integers

Recall that the gcd of a, b is 0 if both a, b are 0 and is the largest positive integer d that divides both a and b otherwise. We will now assume that at least one of a, b is non-zero. We note the following obvious properties of the gcd function.

1. $\gcd(a, b) = \gcd(b, a)$.
2. $\gcd(a, b) = \gcd(|a|, |b|)$.
3. $\gcd(0, b) = |b|$.
4. $\gcd(a, b) = \gcd(a - b, b)$.

Exercise 4.6 *Prove each of the preceding properties of* gcd*'s.*

From now on we shall assume that a, b are non-negative. The various equations suggest the following (impractical) method of finding $\gcd(a, b)$.

1. If $a = 0$ then return b.
2. If $a < b$ then return $\gcd(b, a)$.
3. Otherwise return $\gcd(a - b, b)$.

Note that this process always halts with $a = 0$ (why?). There is one obvious improvement that is suggested by the fact that

$$\gcd(a, b) = \gcd(a - b, b) = \gcd(a - 2b, b) = \ldots = \gcd(a - qb, b)$$

for any integer q. In particular if $b > 0$ and we put

$$a = qb + r$$

where $0 \leq r < b$ and q is an integer then

$$\gcd(a, b) = \gcd(b, r).$$

We usually call q the *quotient* of a, b and r the *remainder* or *residue*. This frequently reduces the number of subtractions we carry out at the expense of introducing an integer division. Thus we may find $\gcd(9, 24) = 3$ by

$$\begin{aligned} 24 &= 2 \times 9 + 6 \\ 9 &= 1 \times 6 + 3 \\ 6 &= 2 \times 3 + 0 \end{aligned}$$

In general we put $r_0 = a$, $r_1 = b$ and write

$$\begin{aligned} r_0 &= q_1 r_1 + r_2 \\ r_1 &= q_2 r_2 + r_3 \\ r_2 &= q_3 r_3 + r_4 \\ &\vdots \\ r_{s-2} &= q_{s-1} r_{s-1} + r_s \\ r_{s-1} &= q_s r_s + r_{s+1} \end{aligned}$$

where $r_{s+1} = 0$ and $0 \leq r_i < r_{i-1}$ for $1 \leq i \leq s+1$. Note that we must eventually have $r_i = 0$ for some i since $r_0 > r_1 > \ldots > r_s > r_{s+1} \geq 0$. Furthermore since

$$\begin{aligned} \gcd(a,b) &= \gcd(r_0, r_1) \\ &= \gcd(r_1, r_2) \\ &\vdots \\ &= \gcd(r_{s-2}, r_{s-1}) \\ &= \gcd(r_{s-1}, r_s) \\ &= \gcd(r_s, r_{s+1}) \\ &= \gcd(r_s, 0) \\ &= r_s \end{aligned}$$

it follows that $\gcd(a,b)$ is just the last non-zero remainder. This is the most common form of *Euclid's algorithm* as this process is known[3]. (If $b = 0$ then the algorithm stops straight away and $\gcd(a,b) = r_0 = a$.)

Exercise 4.7 *What happens if we use Euclid's algorithm with $a < b$? Try an example.*

Exercise 4.8 *Assume that $b > 0$. Prove that if $a = qb + r$ where $0 \leq r < b$ and q is an integer then $\gcd(a,b) = \gcd(b,r)$. (Hint: Show that the set of divisors of a and b is exactly the same as the set of divisors of b and r.)*

Suppose we apply Euclid's algorithm and arrive at the last non-zero remainder r_s. We may then rewrite the last step as

$$r_s = r_{s-2} - q_{s-1} r_{s-1}.$$

The remainder r_{s-1} can be written as

$$r_{s-1} = r_{s-3} - q_{s-2} r_{s-2}$$

so that

$$r_s = -q_{s-1} r_{s-3} + (1 + q_{s-1} q_{s-2}) r_{s-2}.$$

This process can be continued until we obtain

$$r_s = u r_0 + v r_1$$

where u, v are *integers*. What we have shown is that if $d = \gcd(a,b)$ then there are integers u, v such that

$$d = ua + vb.$$

Moreover Euclid's algorithm enables us to compute u, v. The process of back-substitution is unattractive from a computational point of view, but it is easy to compute u, v in a 'forwards' direction (i.e., along with the computation of the gcd). This version of Euclid's Algorithm is usually called the *Extended Euclidean Algorithm*. Note that u, v are not unique, e.g., $(-1) \cdot 2 + 1 \cdot 3 = (-4) \cdot 2 + 3 \cdot 3 = 1$. The Extended Euclidean Algorithm simply produces one possible pair of values. Finally note that if $a = b = 0$ then the equation holds trivially since the gcd is 0.

[3]Given by Euclid in his *Elements*, Book 7, Propositions 1 and 2. This dates from about 300 B.C.

Exercise 4.9 *Show how to compute u, v along with the* gcd *by updating appropriate variables.*

Exercise 4.10 *Let $a_1, \ldots, a_n \in \mathbb{Z}$ not all 0.*

1. *Define $\gcd(a_1, \ldots, a_n)$ to be the largest positive integer d that divides each a_i. Prove that $d = \gcd(a_1, \gcd(a_2, \ldots, \gcd(a_{n-1}, a_n) \cdots)$. Deduce that there are integers $u_1, \ldots, u_n$ such that $d = u_1 a_1 + \cdots u_n a_n$.*

2. *Define $(a_1, \ldots, a_n) = \{\, u_1 a_1 + \cdots + u_n a_n \mid u_1, \ldots, u_n \in \mathbb{Z} \,\}$ (we will see later on that this is an example of an ideal of a ring, in this case $\mathbb{Z}$). Prove that $(a_1, \ldots, a_n) = (d)$ where $d = \gcd(a_1, \ldots, a_n)$. (Here you are proving that two sets are equal so show that each is contained in the other.)*

Lemma 4.1 *$\mathbb{Z}_n$ is a field if and only if n is a prime.*

Proof First we show that if n is not a prime then $\mathbb{Z}_n$ is not a field. If n is not a prime then it is either 1 or a composite number. If $n = 1$ then in $\mathbb{Z}_n$ we have $0 = 1$ which is not allowed in a field. If n is composite then we have $n = rs$ for some integers r, s with $1 < r, s < n$. Thus $r \neq 0$ and $s \neq 0$ in $\mathbb{Z}_n$ but $rs = 0$ in $\mathbb{Z}_n$ and this cannot happen in a field.

Now we show that if n is a prime then $\mathbb{Z}_n$ is a field. We know that $\mathbb{Z}_n$ is a commutative ring with identity for all n. Moreover when n is a prime then $0 \neq 1$ in $\mathbb{Z}_n$ (since $n > 2$ so that 0 and 1 are distinct remainders modulo n). Thus we need only show that every nonzero element of $\mathbb{Z}_n$ has a multiplicative inverse. So suppose that a is a non-zero member of $\mathbb{Z}_n$. It followls that n does not divide a and so $\gcd(a, n) = 1$ (since n is a prime the only possible divisors are 1 and n). Thus by the Extended Euclidean Algorithm there are integers u, v such that $ua + vn = 1$. Thus $ua = 1$ in $\mathbb{Z}_n$ and so u is the required multiplicative inverse. □

Exercise 4.11 *Prove that the invertible elements of $\mathbb{Z}_n$ are precisely those a such that $\gcd(a, n) = 1$.*

Exercise 4.12 *Consider the equation*

$$ax + by = c$$

where a, b, c are all integers. Suppose that we wish to solve this equation for integers x, y. Of course it is possible that no such solution exists. For example the equation $2x + 4y = 3$ has no solution in integers x, y since the left hand side is always even while the right hand side is odd.

State a necessary and sufficient condition for the equation to have a solution in integers.

Exercise 4.13 *The process of back-substitution can be formalized as follows. Put*

$$Q_n(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if } n = 0; \\ x_1, & \text{if } n = 1; \\ x_1 Q_{n-1}(x_2, \ldots, x_n) + Q_{n-2}(x_3, \ldots, x_n), & \text{if } n > 1. \end{cases}$$

Thus $Q_2(x_1, x_2) = x_1 x_2 + 1$, $Q_3(x_1, x_2, x_3) = x_1 x_2 x_3 + x_1 + x_3$, etc.

1. *It was noted by the great eighteenth century mathematician L. Euler that $Q_n(x_1, x_2, \ldots, x_n)$ is the sum of all terms obtainable by starting with $x_1 x_2 \cdots x_n$ and deleting zero or more non-overlapping product pairs $x_i x_{i+1}$. Prove this and deduce that*

$$Q_n(x_1, x_2, \ldots, x_n) = Q_n(x_n, \ldots, x_2, x_1)$$

so that for $n > 1$

$$Q_n(x_1, x_2, \ldots, x_n) = x_n Q_{n-1}(x_1, \ldots, x_{n-1}) + Q_{n-2}(x_1, \ldots, x_{n-2})$$

2. *Prove, by induction on n, that*

   $$r_n = (-1)^n \big(Q_{n-2}(q_2, \ldots, q_{n-1}) r_0 - Q_{n-1}(q_1, \ldots, q_{n-1}) r_1\big)$$

   for all $n \geq 2$. (Here the q_i are the quotients of Euclid's algorithm applied to r_0, r_1 and the r_i are the remainders.)

3. *Express $Q_n(q_1, \ldots, q_n)$ and $Q_{n-1}(q_2, \ldots, q_n)$ in terms of a, b and $\gcd(a, b)$*

Exercise 4.14 *Let $/x_1, x_2, \ldots, x_n/$ denote the continued fraction*

$$\cfrac{1}{x_1 + \cfrac{1}{x_2 + \cfrac{1}{\cfrac{\cdots}{x_{n-1} + \cfrac{1}{x_n}}}}}$$

Show that

$$\frac{b}{a} = /q_1, q_2, \ldots, q_n/$$

where the q_i are all the quotients obtained by applying Euclid's algorithm with $r_0 = a$, $r_1 = b$, e.g., $15/39 = /2, 1, 1, 2/$.

(Can you see the relationship between the continued fraction and the polynomials Q_i defined in the preceding exercise?)

See Knuth [39] for more material related to the last two exercises (and Euclid's algorithm in general).

4.6.2 Worst-Case Runtime of Euclid's Algorithm

Suppose we carry out Euclid's algorithm with $r_0 = a$, $r_1 = b$. It is clear that all numbers involved are no larger than $\max(a, b)$ so that we never have to carry out any arithmetic on numbers that are bigger than $\max(a, b)$. It follows that if we know the worst possible number of steps of the form

$$r_i = q_{i+1} r_{i+1} + r_{i+2}$$

then we shall have a good idea of the worst-case runtime of Euclid's algorithm. To be precise the ith step involves us in computing q_{i+1} and r_{i+2} which consists of an integer division and a remainder. We shall give an outline of the analysis, which turns out to be a little surprising.

Let $F_0, F_1, F_2, \ldots$ denote the Fibonacci sequence defined by

$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{otherwise.} \end{cases}$$

i.e., the sequence $0, 1, 1, 2, 3, 5, \ldots$. These numbers have many interesting properties, e.g.,

$$\gcd(F_m, F_n) = F_{\gcd(m,n)}.$$

Now suppose we apply Euclid's algorithm to F_{n+2}, F_{n+1}. We observe the following behaviour

$$\begin{aligned} F_{n+2} &= 1 \times F_{n+1} + F_n \\ F_{n+1} &= 1 \times F_n + F_{n-1} \\ &\vdots \\ 5 &= 1 \times 3 + 2 \\ 3 &= 1 \times 2 + 1 \\ 2 &= 2 \times 1 \end{aligned}$$

and there are exactly n steps. Note how the quotients are all 1 except for the last one which is 2 (the last quotient can never be 1 in any application of Euclid's algorithm with $r_0 \neq r_1$, can you see why?). In other words we are making as little progress as is possible. The following result confirms an obvious conjecture.

Theorem 4.3 (G. Lamé, 1845) *For $n \geq 1$, let a, b be integers with $a > b > 0$ such that Euclid's algorithm applied to a, b requires exactly n steps and such that a is as small as possible satisfying these conditions. Then*

$$a = F_{n+2}, \qquad b = F_{n+1}.$$

Corollary 4.1 *If $0 \leq a, b < N$, where N is an integer, then the number of steps required by Euclid's algorithm when it is applied to a, b is at most $\lceil \log_\phi(\sqrt{5}N) \rceil - 2$ where $\phi = \frac{1}{2}(1 + \sqrt{5})$, the so-called golden ratio.*

Proof Let n be the number of steps taken by Euclid's algorithm applied to a, b. Note that the worst possibility is to have $a < b$ since then the first step is 'wasted' in swapping a and b. The algorithm then proceeds as though it had been applied to b, a where $b > a$ and takes $n - 1$ steps. Now according to the theorem we have $a = F_n$, $b = F_{n+1}$. So we want to find the largest n that satisfies

$$F_{n+1} < N \tag{2}$$

Now it can be shown that

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n) \tag{3}$$

where $\hat{\phi} = 1 - \phi = \frac{1}{2}(1 - \sqrt{5})$ (see Exercise 4.15 or §4.7.2). Note that $\hat{\phi} = -0.6183\ldots$ and $\hat{\phi}^n$ approaches 0 very rapidly. In fact $\hat{\phi}^n/\sqrt{5}$ is always small enough so that

$$F_{n+1} = \frac{\phi^{n+1}}{\sqrt{5}} \quad \text{rounded to the nearest integer.}$$

Combining this with equation (2) we obtain $n + 1 < \log_\phi(\sqrt{5}N)$ which completes the proof. □

It is worth noting that $\log_\phi(\sqrt{5}N)$ is approximately $4.785 \log_{10} N + 1.672$. For more details see Knuth [39] or Cormen, Leiserson and Rivest [18].

Exercise 4.15 *Let*

$$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

1. *Show, by induction on n, that*

   $$F^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

   for all $n > 0$.

2. *Find a matrix S such that*

   $$SFS^{-1} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

   for some real numbers λ_1, λ_2. Note that

   $$(SFS^{-1})^n = S^{-1}F^nS,$$
   $$\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}^n = \begin{pmatrix} \lambda_1^n & 0 \\ 0 & \lambda_2^n \end{pmatrix}.$$

   Use these two facts to deduce the formula (3).

4.7 Polynomials

We start by giving a fairly formal definition of polynomials. This will be helpful in understanding the *symbolic* nature of Computer Algebra systems. (We have already used polynomials informally and it is very likely that you are familiar with them. Rest assured that this section simply gives one way of defining them. However there is a widespread misconception about polynomials—see §4.7.1—so take care!)

Let R be a commutative ring with identity, for us it will usually be one of $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ or $\mathbb{Z}_n$. Let x be a new symbol (i.e., x does not appear in R). Then a *polynomial* in the *indeterminate* x with *coefficients* from R is an object of form

$$a_0 + a_1x + a_2x^2 + \cdots + a_nx^n + \cdots$$

where each $a_i \in R$ and all but finitely many of the a_i are 0. Note that the + which appears above does not denote addition in R (after all x is not an element of R). Indeed we could just regard polynomials as infinite tuples

$$(a_0, a_1, a_2, \ldots)$$

but the first notation is much more convenient as we shall see[4]. We call a_0 the *constant* term of the polynomial and a_i the *coefficient* of x^i (we regard a_0 as the coefficient of x^0). The set of all such polynomials is denoted by $R[x]$.

[4]If you are curious here is bit more detail about the connection between the the two approaches. We define operations on the sequence notation that give us a ring. We then denote the sequence $(0, 1, 0, 0, \ldots)$ by x. From the definition of multiplication it follows that $x^i = (0, \ldots, 0, 1, 0, \ldots)$ where the 1 is in position i. From the definitions we also have that $(0, \ldots, 0, a, 0, \ldots) = a(0, \ldots, 0, 1, 0, \ldots) = ax^i$. Addition is component-wise and so it follows that $(a_0, a_1, a_2, \ldots) = a_0 + a_1x + a_2x^2 + \cdots$, so we have now got back to our familiar notation. Strictly speaking this approach is much more sound as it uses standard notions and then justifies the new notation.

A convenient abbreviation is the familiar sigma notation

$$\sum_{i=0}^{\infty} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n + \cdots$$

Two polynomials are *equal* if and only if the coefficients of corresponding powers of x are equal as elements of R. Thus

$$\sum_{i=0}^{\infty} a_i x^i = \sum_{i=0}^{\infty} b_i x^i$$

if and only if $a_0 = b_0$, $a_1 = b_1$, $a_2 = b_2$ etc. When writing polynomials we normally leave out any terms whose coefficient is 0. Thus we write

$$2 + 5x^3 - 3x^5$$

rather than

$$2 + 0x + 0x^2 + 5x^3 + 0x^4 - 3x^5 + 0x^6 + \cdots$$

Moreover we can also denote the same polynomial by $5x^3 - 3x^5 + 2$ or by $-3x^5 + 5x^3 + 2$. Thus if $a_i = 0$ for all $i > 0$ then we denote the polynomial by just a_0 (such a polynomial is called *constant*). The *zero* polynomial has 0 for all of its coefficients and is denoted by 0.

The highest power of x that has a non-zero coefficient in a polynomial p, is called the *degree* of p and is denoted by $\deg(p)$. The corresponding coefficient is called the *leading coefficient* of p and is sometimes denoted by $\mathrm{lc}(p)$. (These notions do not make sense for the zero polynomial. We leave $\deg(0)$ and $\mathrm{lc}(0)$ undefined. For example

$$\begin{aligned} \deg(x + 3x^3 - 6x^{10}) &= 10, \\ \mathrm{lc}(x + 3x^3 - 6x^{10}) &= -6. \end{aligned}$$

We define the *sum*, *difference* and *product* of polynomials in the obvious way.

$$\begin{aligned} \sum_{i=0}^{\infty} a_i x^i \pm \sum_{i=0}^{\infty} b_i x^i &= \sum_{i=0}^{\infty} (a_i \pm b_i) x^i, \\ \left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{\infty} b_i x^i\right) &= \sum_{i=0}^{\infty}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i. \end{aligned}$$

If m, n are the respective degrees of the two polynomials then we can write these definitions as

$$\begin{aligned} \sum_{i=0}^{m} a_i x^i \pm \sum_{i=0}^{n} b_i x^i &= \sum_{i=0}^{\max(m,n)} (a_i \pm b_i) x^i, \\ \left(\sum_{i=0}^{m} a_i x^i\right)\left(\sum_{i=0}^{n} b_i x^i\right) &= \sum_{i=0}^{m+n}\left(\sum_{j=0}^{i} a_j b_{i-j}\right) x^i \end{aligned}$$

where it is understood that a_i is 0 if $i > m$ and similarly b_j is 0 if $j > n$. Note that the definition above yields $ax = xa$ for all $a \in R$ from which it follows that every element of R commutes with every polynomial. It is quite easy to show that, with these definitions, the three operations on polynomials obey various well known (and usually tacitly assumed) rules of arithmetic such as

$$\begin{aligned}
p+q &= q+p,\\
pq &= qp,\\
(p+q)+r &= p+(q+r),\\
(pq)r &= p(qr),\\
p(q+r) &= pq+pr.
\end{aligned}$$

Indeed $R[x]$ is a commutative ring with identity under the operations given above. We have

$$\begin{aligned}
\deg(p \pm q) &\le \max\bigl(\deg(p), \deg(q)\bigr),\\
\deg(pq) &\le \deg(p)+\deg(q),\\
\deg(pq) &= \deg(p)+\deg(q), \quad \text{if } \operatorname{lc}(p)\operatorname{lc}(q) \neq 0,
\end{aligned}$$

whenever both sides are defined. For example

$$\begin{aligned}
(2+3x-4x^3)+(-3x+2x^2) &= (2+0)+(3-3)x+(0+2)x^2+(-4+0)x^3\\
&= 2+2x^2-4x^3
\end{aligned}$$

while

$$\begin{aligned}
(2+3x-4x^3)\times(-3x+2x^2) = {} & (2\cdot 0)\\
&+(2\cdot(-3)+3\cdot 0)x\\
&+(2\cdot 2+3\cdot(-3)+0\cdot 0)x^2\\
&+(2\cdot 0+3\cdot 2+0\cdot(-3)+(-4)\cdot 0)x^3\\
&+(2\cdot 0+3\cdot 0+0\cdot 2+(-4)\cdot(-3)+0\cdot 0)x^4\\
&+(2\cdot 0+3\cdot 0+0\cdot 0+(-4)\cdot 2+0\cdot(-3)+0\cdot 0)x^5\\
= {} & -6x-5x^2+6x^3+12x^4-8x^5.
\end{aligned}$$

If the coefficients come from $\mathbb{Z}$ then the degree of the product is equal to the sum of the degrees. On the other hand if the coefficients come from $\mathbb{Z}_8$ then the product polynomial is equal to $2x+3x^2+6x^3+4x^4$ and the degree of the product is less than the sum of the degrees of the two factors.

The preceding example makes rather heavy weather of the process of multiplying polynomials. An alternative approach is to multiply each term of the first polynomial with each one of the second and then collect coefficients of equal powers of x. we can write this out in tabular form as follows

$$\begin{array}{ccccccccccc}
2 & + & 3x & & & - & 4x^3 & & & & \\
 & - & 3x & + & 2x^2 & & & & & & \\
\hline
 & - & 6x & - & 9x^2 & & & + & 12x^4 & & \\
 & & & + & 4x^2 & + & 6x^3 & & & - & 8x^5 \\
\hline\hline
 & - & 6x & - & 5x^2 & + & 6x^3 & + & 12x^4 & - & 8x^5
\end{array}$$

Of course this is just the same process as the familiar school method of multiplying integers, with the difference that we do not need to concern ourselves with carries. Most computer algebra systems multiply polynomials by an algorithm based on this process. (It is interesting that the implementation of such a simple algorithm still presents a subtlety about the amount of memory used—see [21], p. 66.)

4.7.1 Polynomial Functions

Given a polynomial

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

we can define a corresponding function $\hat{p} : R \to R$ by putting

$$\hat{p}(\alpha) = a_0 + a_1 \alpha + \cdots + a_n \alpha^n$$

for each $\alpha \in R$. The preceding expression looks so much like the polynomial p that it is common practice to identify the function and the polynomial (indeed the indeterminate x is frequently referred to as a *variable*). This is potentially dangerous since, e.g., the notions of equality between polynomials and between functions are completely different. Recall that two functions $f, g : R \to R$ are said to be equal if and only if $f(\alpha) = g(\alpha)$ for all $\alpha \in R$. Now let p be as above and

$$q = b_0 + b_1 x + \cdots + b_m x^m.$$

We have that

$$\begin{aligned} p = q &\Leftrightarrow a_i = b_i, \quad \text{for } i = 0, 1, \ldots \\ &\Leftrightarrow p = q = 0, \quad \text{or} \\ &\quad\ \deg(p) = \deg(q) \ \&\ a_i = b_i, \quad \text{for } 0 \le i \le \deg(p). \end{aligned}$$

On the other hand if we think of p and q as functions (i.e., identify them with $\hat{p}$ and $\hat{q}$) then we have

$$p = q \Leftrightarrow p(\alpha) = q(\alpha) \quad \text{for all } \alpha \in R.$$

Fortunately as long as R is an *infinite integral domain* (such as $\mathbb{Z}$ or any field) then it can be shown that the two definitions coincide and so there is no harm in confusing the polynomial p with the corresponding function $\hat{p}$. (In fact there is a deeper reason, the ring of polynomial functions is *isomorphic* to the ring of polynomials.)

Exercise 4.16 *Let p be a polynomial with coefficients from an integral domain R. A number α is said to be a root of p if $p(\alpha) = 0$. It can be shown that if $p \neq 0$ then it has at most* $\deg(p)$ *roots in R. Use this fact to show that $p = q$ if and only if $\hat{p} = \hat{q}$ for two polynomials p and q when R is infinite.*

When R is finite the situation is completely different. Let the elements of R be $r_1, \ldots, r_n$ and put

$$Z(x) = (x - r_1) \cdots (x - r_n).$$

Assume that $n > 1$ so that $1 \neq 0$ in R. Then clearly $Z(x)$ is not the zero polynomial (it has degree n and leading coefficient 1). However the associated polynomial function $\hat{Z}$ is the zero function.

Exercise 4.17 *This exercise shows that if R is not an integral domain then the distinction between polynomials and the corresponding functions is important even if R is infinite.*

Let R_1, R_2 be two commutative rings with identities 1_A and 1_B respectively. Show that the obvious definitions turn

$$R_1 \times R_2 = \{\, (r_1, r_2) \,|\, r_1 \in R_1, r_2 \in R_2 \,\}$$

into a ring with identity $(1_A, 1_B)$ and zero $(0_A, 0_B)$. Now let $a = (1_A, 0_B)$ and deduce that every element of the form $(0_A, r_2)$ is a root of the polynomial ax.

4.7.2 Digression: Formal Power Series

Another way of defining polynomials is as *finite* expressions of the form

$$\sum_{i=0}^{n} a_i x^i.$$

We can proceed with our definitions as before.

The definition via infinite tuples suggests a possible generalization: drop the condition that all but finitely many of the coefficients must be 0. All the definitions go through exactly as before and we thus obtain the ring $R[[x]]$ of *formal power series* in the indeterminate x with coefficients from R. Notice that here there is no question of worrying about convergence. This matter only arises when we wish to consider a formal power series $P(x)$ as a *function* $R \to R$ just as in the case of polynomial functions. Clearly every polynomial is a formal power series and indeed the ring of polynomials $R[x]$ is a *subring* of $R[[x]]$. Algebraically speaking we make quite a gain in moving from $R[x]$ to $R[[x]]$. For example in $R[x]$ an element p has a multiplicative inverse (i.e., there is a q such that $pq = 1$) if and only if p is a constant (i.e., an element of R) and has an inverse in R. The only invertible polynomials in $\mathbb{Z}[x]$ are 1 and -1. However in the ring of formal power series $R[[x]]$ it is easy to see that an element P has a multiplicative inverse if and only if its constant term (i.e. the coefficient of x^0) is non-zero and has an inverse in R—the other coefficients can be arbitrary. For example in $\mathbb{Z}[[x]]$ we have

$$(1 - x)(1 + x + x^2 + x^3 + \cdots + x^n + \cdots) = 1$$

so that

$$\frac{1}{1 - x} = 1 + x + x^2 + x^3 + \cdots + x^n + \cdots$$

(It is common practice to use $1/a$ to denote the inverse of an invertible element of a ring R.) These ideas are very useful in many areas; we give an illustration from enumerative combinatorics. Let $a_0, a_1, \ldots$ be a sequence of numbers. We define the *generating function* of this sequence by

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots$$

(although $A(x)$ is called a generating *function* this is just a nod to tradition and it is really a formal power series). To illustrate the usefulness of this notion let us consider our old friends the Fibonacci numbers $F_0, F_1, F_2, \ldots$. The generating function for these is

$$F(x) = F_0 + F_1 x + F_2 x^2 + \cdots$$

Now it is clear that

$$F(x)(1-x-x^2) = x$$

and this is true in $\mathbb{R}[[x]]$. Moreover $1-x-x^2$ has a non-zero constant term and so it has an inverse in $\mathbb{R}[[x]]$. Thus

$$\begin{aligned} F(x) &= \frac{x}{1-x-x^2} \\ &= \frac{-x}{(\phi+x)(\hat{\phi}+x)} \end{aligned}$$

where $\phi = \frac{1}{2}(1+\sqrt{5})$ and $\hat{\phi} = \frac{1}{2}(1-\sqrt{5})$ (of course $-\phi$ and $-\hat{\phi}$ are the two roots of $1-x-x^2$). Thus, from the partial fraction decomposition of the last expression, we have

$$F(x) = \frac{x}{\sqrt{5}}\left(\frac{1}{\phi+x} - \frac{1}{\hat{\phi}+x}\right).$$

It is clear that $\phi\hat{\phi} = -1$ and so

$$\begin{aligned} \frac{1}{\hat{\phi}+x} &= \frac{-\phi}{1-\phi x} \\ &= -\phi(1+\phi x + (\phi x)^2 + \cdots) \\ &= -\phi - \phi^2 x - \phi^3 x^2 - \cdots \end{aligned}$$

and similarly

$$\begin{aligned} \frac{1}{\phi+x} &= \frac{-\hat{\phi}}{1-\hat{\phi}x} \\ &= -\hat{\phi} - \hat{\phi}^2 x - \hat{\phi}^3 x^2 - \cdots. \end{aligned}$$

Thus

$$\begin{aligned} F(x) &= \frac{x}{\sqrt{5}}\left(\frac{1}{\phi+x} - \frac{1}{\hat{\phi}+x}\right) \\ &= \frac{1}{\sqrt{5}}(\phi-\hat{\phi})x + \frac{1}{\sqrt{5}}(\phi^2-\hat{\phi}^2)x^2 + \frac{1}{\sqrt{5}}(\phi^3-\hat{\phi}^3)x^3 + \cdots. \end{aligned}$$

Equating coefficients we obtain

$$\begin{aligned} F_n &= \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n) \\ &= \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right). \end{aligned}$$

4.7.3 Polynomials in Several Indeterminates

Since $R[x]$ is a commutative ring with identity we can repeat the whole polynomial ring construction with a new indeterminate y to obtain a new ring $R[x][y]$ of polynomials in y with coefficients from $R[x]$. Every element of $R[x][y]$ can be written in the form

$$\sum_{i,j=0}^{\infty} r_{ij}x^i y^j \tag{4}$$

where $r_{ij} \in R$ for all i, j. This helps to reveal the fact that $R[x][y]$ and $R[y][x]$ are essentially the *same* ring (i.e. they are *isomorphic*). Because of this we use the notation $R[x, y]$. (Note that that x, y commute, i.e., $xy = yx$.) When taking this latter view each product $x^i y^j$ in (4) is called a *power product* in x, y of degree $i + j$. Thus in this view the degree of (4) is undefined if all coefficients r_{ij} are 0 and otherwise it is the maximum degree of a power product which occurs in it with non-zero coefficient. The properties of degrees given above carry through to the new situation. We use a phrase such as 'degree in x' to indicate that we are viewing a polynomial of $R[x, y]$ as being in the indeterminate x with coefficients from $R[y]$, this degree is denoted by $\deg_x(\cdot)$. Similar comments apply to the leading coefficient, which is defined when we are focusing on one indeterminate as the 'main' one. For example

$$\begin{array}{rl}
\deg(1 + x + x^2y + x^3y + 2x^3y^2) = 5 & \qquad \text{lc not defined} \\
\deg_x(1 + x + x^2y + x^3y + 2x^3y^2) = 3 & \qquad \mathrm{lc}_x(1 + x + x^2y + x^3y + 2x^3y^2) = y + 2y^2 \\
\deg_y(1 + x + x^2y + x^3y + 2x^3y^2) = 2 & \qquad \mathrm{lc}_y(1 + x + x^2y + x^3y + 2x^3y^2) = 2x^3
\end{array}$$

The concept of leading coefficient does not make sense when we consider polynomials in several indeterminates at once because then there are usually several different power products of the same (highest) degree. Consider for example the polynomial

$$x^3 + x^2y + xy^2 + y^3 + xy + x + y + 1.$$

This has degree 3 and has four power products of this degree. In order to give a meaning to the concept of leading coefficient we must have some notion of one power product being more important than another, i.e., we must introduce a *total order* on power products. This is a very useful idea and we will discuss it later on when considering representations of polynomials as well as the construction of Gröbner bases. Of course if we specify a power product t then it always makes sense to ask for its coefficient, which could be 0, in a polynomial p. This is denoted by $\mathrm{coeff}(t, p)$.

Clearly we can generalize polynomial rings to finite sets $X = \{x_1, \ldots, x_n\}$ of indeterminates. The power products here are all expressions of the form

$$x_1^{i_1} \cdots x_n^{i_n}$$

where $i_1, \ldots, i_n \in \mathbb{N}$. The degree of such a power product is just $i_1 + \cdots + i_n$. Finally an expression of the form ct where $c \in R$ and t is a power product is called a *monomial*.

4.7.4 Differentiation

Let

$$p = a_0 + a_1x + \cdots + a_nx^n$$

be a polynomial in x with coefficients from some commutative ring R. Then we can define its *derivative* in a purely formal manner as

$$p' = a_1 + 2a_2x + \cdots + na_nx^{n-1}.$$

(We sometimes use the traditional notation dp/dx instead of p'.) Note that there is no reliance on limits here. It is easy to see that all the algebraic properties of derivatives (i.e., ones that do not depend on limits) still hold. Thus

$$\begin{aligned}
(p + q)' &= p' + q' \\
(pq)' &= p'q + pq'.
\end{aligned}$$

It follows from the definition that $p' = 0$ if $p \in R$. However the converse is not necessarily true: consider $x^2 + 1 \in \mathbb{Z}_2[x]$. Of course the converse is true when the coefficients come from the familiar number systems $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ or $\mathbb{C}$.

Let k be any field and $k(x)$ consist of all ratios p/q where $p, q \in k[x]$ and $q \neq 0$. If we define addition and multiplication in the obvious way then $k(x)$ is a field (see §4.7.8). We can extend the definition of differentiation to $k(x)$ by putting

$$(p/q)' = \frac{p'q - pq'}{q^2},$$

where $p, q \in k[x]$. We can also define partial derivatives (e.g., for elements of $k(x, y)$) in the obvious way.

Exercise 4.18 *Let f be an element of $\mathbb{Z}_p[x]$ where p is a prime number. Show that $f' = 0$ if and only if $f \in \mathbb{Z}_p[x^p]$ i.e., f has the form*

$$f = a_0 + a_1x^p + a_2x^{2p} + \cdots + a_nx^{pn}.$$

4.7.5 Factorization and Greatest Common Divisors

Let R be a unique factorization domain (UFD), which for the moment can be interpreted as $\mathbb{Z}$, or any field (normally $\mathbb{Q}$ or $\mathbb{Z}_p$ for p a prime). The first important consequence of this assumption is that

$$\deg(pq) = \deg(p) + \deg(q), \quad \text{for all } p, q \in R[x].$$

We know that in R we can factorize non-zero elements into primes, can we extend this to polynomials from $R[x]$? Given an arbitrary (non-zero) polynomial p we can try to write it as a product $p = hk$ where h, k have degree which is strictly smaller than that of p (this means that they cannot be constant polynomials). If this is possible then we can repeat the process with h and k and so on. This process must eventually stop because at each stage the degree of the relevant polynomials decreases and of course it is always greater than 0. Thus we have

$$p = p_1p_2 \cdots p_s.$$

where each p_i cannot be expressed as a product of two polynomials of smaller degree. We should like this factorization to be unique in the same sense that the factorization of numbers into primes is unique. Consider however the factorizations

$$\begin{aligned} 6x^2 + 30x + 36 &= (2x + 4)(3x + 9) \\ &= 2 \cdot 3(x + 2)(x + 3) \end{aligned}$$

where the polynomials are from $\mathbb{Z}[x]$. None of the factors is invertible in $\mathbb{Z}[x]$ but the number of factors in the two factorizations is different. The source of the problem is obvious: in the first factorization the polynomials $2x + 4$ and $3x + 9$ are *not* irreducible elements of the ID $\mathbb{Z}[x]$. In other words the process which we described did not carry out a complete factorization. (This kind of difficulty does not arise when R is a field—can you see why?) Now given

$$f = a_0 + a_1x + \cdots + a_nx^n$$

define the *content* of f, denoted by $\mathrm{cont}(f)$, to be the gcd of its coefficients. If the content of f is c then we may write

$$f = cg$$

and call g the *primitive part* of f, denoted by $\mathrm{pp}(f)$. (A polynomial is called *primitive* if its content is 1 or any invertible element.) In order to obtain a unique factorization for p we write each p_i as $c_i p_i'$ where c_i is the content of p_i. We may then collect all the contents together into one element c of R and put

$$p = cp_1'p_2' \cdots p_s'.$$

Suppose that we also have

$$p = dq_1q_2 \cdots q_t,$$

where each q_i is primitive and cannot be written as the product of two polynomials of lower degree. Then it is a remarkable fact that:

1. $d = \epsilon c$ for some invertible element ϵ of R,
2. $s = t$, and
3. there is a permutation π of $1, 2, 3, \dots, s$ such that $p_i' = \epsilon_i q_{\pi(i)}$ for $1 \le i \le n$ where each ϵ_i is an invertible element of R.

In other words $R[x]$ is a UFD. (Why do we not bother factorizing c and d?)

What we have is that if R is a UFD then so is $R[x]$. Thus if R is a UFD then so is $R[x, y]$ because we can regard this as $R[x][y]$ and let $R[x]$ play the rôle of R in the preceding discussion. It is now clear that the same can be said of $R[x_1, \dots, x_n]$.

Naturally there is a huge difference between knowing that polynomials can be factorized and actually producing a factorization. This is one important application area of Computer Algebra.

It is worth noting that the phrase *irreducible polynomial* is normally used in the following sense: p is said to be irreducible if we cannot write it as the product of two other polynomials of smaller degree. Thus $2x+4$ is an irreducible polynomial of $\mathbb{Z}[x]$ even though it is not an irreducible element of this ring! As another example $x^2 + 1$ is irreducible as a polynomial in $\mathbb{R}[x]$. To see this suppose otherwise, then we must have

$$x^2 + 1 = (a_1x + a_0)(b_1x + b_0)$$

for some real numbers a_0, a_1, b_0, b_1. Note that $a_1 \ne 0$ otherwise the degree of the right hand side is two low. But now this means that $-a_0/a_1$ is a root of x^2+1 and this is impossible since the square of any real number is non-negative. This example brings out a very important fact: the concept of irreducibility depends on the ring R of coefficients. If we regard $x^2 + 1$ as a polynomial in $\mathbb{C}[x]$ then of course we can factorize it as $(x + i)(x - i)$ where $i = \sqrt{-1}$.

Exercise 4.19 *Prove that $x^2 - 2$ is irreducible in $\mathbb{Z}[x]$ but of course in $\mathbb{R}[x]$ it factorises.*

We now turn our attention to the gcd and assume that $f \ne 0$ or $g \ne 0$ so any gcd is non-zero. Recall from §4.2 that h is a gcd of two polynomials f, g if h divides both of the polynomials and any other common divisor of f, g also divides h. Furthermore any two gcd's are related by an invertible element, which in the case of polynomial rings must be an invertible constant from the ring of coefficients R (prove this). It is easy to see that no polynomial of degree strictly larger than that of h can be a common divisor of f, g. If R is a field then all nonzero constants are invertible and this means that we can define gcd's of f, g to be all those common divisors of f, g of largest possible degree. If R is not a field then this is not quite enough to capture the gcd (we might be missing some constant non-invertible factor). It is standard practice to abuse notation and use $\gcd(f, g)$ to stand for a greatest common divisor of f, g and even talk of *the* gcd. This does no harm because we are not normally bothered about multiples by invertible elements. In many cases it is possible to remove the ambiguity by insisting on some extra condition that must be satisfied by a gcd. For example if the coefficients are rationals then we can insist that the leading coefficient should be 1 (i.e., the gcd is *monic*).

Just as in the case of the integers there is a very close connection between the gcd and factorization. It is left up to you to work this out if you are interested.

We will also make use of the *least common multiple* of two polynomials f, g. This is a polynomial of least possible degree which is divisible by both f and g. It is unique up to constant factors and is denoted by $\operatorname{lcm}(f, g)$. In fact we have $\operatorname{lcm}(f, g) = fg/\gcd(f, g)$.

4.7.6 Euclid's Algorithm for Univariate Polynomials

Let f, g be univariate polynomials with coefficients from a field. Suppose we wish to find $\gcd(f, g)$ where $g \neq 0$ and we have

$$f = qg + r \tag{5}$$

where $r = 0$ or $\deg(r) < \deg(g)$. We call q the *quotient* of f divided by g and r the *remainder*. Now if $h \mid f$ and $h \mid g$ then certainly $h \mid r$. On the other hand if $h \mid g$ and $h \mid r$ then $h \mid f$. It follows from this that $\gcd(f, g)$ is the same as $\gcd(g, r)$. For the moment we assume that we can always find an expression of the form (5). This means that we can find polynomial gcd's by mimicking Euclid's algorithm for integer gcd's. To be specific we put $r_0 = f$, $r_1 = g$ and

$$\begin{aligned} r_0 &= q_1 r_1 + r_2 \\ r_1 &= q_2 r_2 + r_3 \\ r_2 &= q_3 r_3 + r_4 \\ &\vdots \\ r_{s-2} &= q_{s-1} r_{s-1} + r_s \\ r_{s-1} &= q_s r_s + r_{s+1} \end{aligned}$$

where $r_{s+1} = 0$ and $\deg(r_i) < \deg(r_{i-1})$ for $1 \leq i \leq s$. Note that we must eventually have $r_i = 0$ for some i since $\deg(r_0) > \deg(r_1) > \ldots > \deg(r_i) > \ldots \geq 0$.

Let us now see how to find q, r so that (5) holds. If $f = 0$ then we just take $q = 0$, $r = 0$. Otherwise we could try $q = 0$, $r = f$. This is fine unless $\deg(f) \geq \deg(g)$; however if this is so then we can improve matters by (repeatedly) replacing q with $q + (a/b)x^{m-n}$ and r with $r - (a/b)x^{m-n}g$ where $a = \operatorname{lc}(r)$, $b = \operatorname{lc}(g)$, $m = \deg(r)$ and $n = \deg(g)$. Each time this is done the degree of r drops and so we must eventually have $r = 0$ or $\deg(r) < \deg(g)$ as required. The correctness of the method follows from the loop invariant $qg + r = f$ (check this).

Note that we can calculate the quotient and remainder polynomials by hand in a manner similar to the long division of integers. For example we can divide $x^3 - x^2 + x - 1$ by $x^2 - 2x + 1$ as follows:

$$\begin{array}{rrrrr|rrrrrrr}
 & & & & & x & + & 1 & & & & \\ \cline{6-12}
x^2 & - & 2x & + & 1 & x^3 & - & x^2 & + & x & - & 1 \\
 & & & & & x^3 & - & 2x^2 & + & x & & \\ \cline{6-10}
 & & & & & & & x^2 & - & 1 & & \\
 & & & & & & & x^2 & - & 2x & + & 1 \\ \cline{8-12}
 & & & & & & & & & 2x & - & 2
\end{array}$$

The quotient is $x + 1$ and the remainder is $2x - 2$. Here we were lucky in that no fractions were needed, in general the calculations involve fractions even if the inputs have integer coefficients.

We now focus on polynomials with rational coefficients. Note that the Euclidean Algorithm relies quite heavily on rational arithmetic and this can be quite costly due to the many gcd computations (on coefficients) carried out. We could avoid this by clearing denominators from the inputs. We may also use the fact that if f, g are non-zero polynomials with integer coefficients and $\deg(f) \geq \deg(g)$ then we can find polynomials q, r with *integer* coefficients such that

$$\operatorname{lc}(g)^{\deg(f)-\deg(g)+1} f = qg + r$$

where $r = 0$ or $\deg(r) < \deg(g)$. This leads to an obvious modification of Euclid's algorithm. Unfortunately this process can lead to very large integers even when the input consists of small ones. For example consider

$$\begin{aligned} f &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, \\ g &= 3x^6 + 5x^4 - 4x^2 - 9x + 21. \end{aligned}$$

The sequence of remainders obtained by applying the modified algorithm is

$$-15x^4 + 3x^2 - 9,$$

$$15795x^2 + 30375x - 59535,$$

$$1254542875143750x - 1654608338437500,$$

$$12593338795500743100931141992187500.$$

(Collectors of obscure misprints will be interested to learn that the final 4 in the last remainder is given as a 5 in [21], p. 69.) Obviously we could try to keep the growth of coefficients down by removing their common factors. Thus

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

could be replaced by

$$(a_n/d)x^n + (a_{n-1}/d)x^{n-1} + \cdots + (a_0/d)$$

where

$$d = \gcd(a_n, a_{n-1}, \dots, a_0).$$

But the whole point of avoiding rational arithmetic was because of its need to compute many gcd's!

The excessive growth of the coefficients is clearly caused by the factor $\mathrm{lc}(g)^{\deg(f)-\deg(g)+1}$ which enables us to stay with integer arithmetic. In fact this is larger than necessary and it is possible to construct algorithms, known as *sub-resultant polynomial remainder sequences*, which avoid this large growth. We do not go into details here since we will look at a different method later on which is the one used in practice.

Finally the discussion in §4.6.1 on the Extended Euclidean Algorithm for the integers carries over directly to the version for polynomials. We can therefore find polynomials u, v such that

$$uf + vg = d$$

where $d = \gcd(f, g)$. Moreover we can ensure that $u = 0$ or $\deg(u) < \deg(g)$ and $v = 0$ or $\deg(v) < \deg(f)$.

Exercise 4.20 *Consider equations of the form*

$$fY + gZ = h$$

where f, g and h are polynomials in x and Y, Z are unknowns. Find a a necessary and sufficient condition for such an equation to have solutions in X, Y that are themselves polynomials. Describe a method for finding such solutions when they exist.

Exercise 4.21 *A naïve attempt at extending Euclid's algorithm to multivariate polynomials does not succeed; why?*

4.7.7 A Remainder Theorem

Let $p(x) \in k[x]$ be a polynomial in the indeterminate x with coefficients from the field k. Choose any $\alpha \in k$. Then we have

$$p(x) = (x - \alpha)q(x) + r(x)$$

for some $q(x), r(x) \in k[x]$ where $r(x) = 0$ or $\deg(r) < \deg(x - \alpha) = 1$ so that $r(x) \in k$, i.e., $r(x)$ is a *constant*. Indeed if we substitute α for x we see that $p(\alpha) = r$ (we do not write $r(\alpha)$ since r does not depend on x). Note that α is a root of $p(x)$ if and only if $r = 0$, i.e., if and only if $x - \alpha \mid p(x)$. This fact can be used to show that $p(x)$ has at most $\deg(p)$ roots (easy exercise in induction). We can summarize the discussion by saying that for all $\alpha \in k$

$$p(x) - p(\alpha) = (x - \alpha)q(x),$$

for some $q(x) \in k[x]$. We can prove this directly by showing that

$$x^i = ((x - \alpha) + \alpha)^i = (x - \alpha)q_i(x) + \alpha^n,$$

for some $q_i(x) \in k[x]$. This is fairly obvious and is easy to prove by induction on i. The full result follows from the fact that $p(x)$ is a linear combination of $1, x, x^2, \dots$. The reason for looking at this proof is that it readily generalizes to multivariate polynomials. Suppose that $f(x_1, x_2, \dots, x_n) \in$

$k[x_1, x_2, \ldots, x_n]$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in k$. We say that $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a root of $f(x_1, x_2, \ldots, x_n)$ if and only if $f(\alpha_1, \alpha_2, \ldots, \alpha_n) = 0$. Suppose that

$$f = (x_1 - \alpha_1)g_1 + (x_2 - \alpha_2)g_2 + \cdots + (x_n - \alpha_n)g_n,$$

for some $g_1, g_2, \ldots, g_n \in k[x_1, x_2, \ldots, x_n]$. Then it is clear that $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ is a root of f. In fact the converse is true for f is a linear combination of power products $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$. Now, putting $y_i = x_i - \alpha_i$ as a notational convenience, we have

$$\begin{aligned} x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n} &= (y_1 + \alpha_1)^{i_1} (y_2 + \alpha_2)^{i_2} \cdots (y_n + \alpha_n)^{i_n} \\ &= (y_1 h_1 + \alpha_1^{i_1})(y_2 h_2 + \alpha_2^{i-2}) \cdots (y_n h_n + \alpha_n^{i_n}) \end{aligned}$$

where $h_i \in k[x_i]$. Now it is clear that the final expression, when multiplied out can be collected so as to have the form

$$y_1 q_1 + y_2 q_2 + \cdots + y_n q_n + \alpha_1^{i_1} \alpha_2^{i_2} \cdots \alpha_n^{i_n}, \tag{$\dagger$}$$

where $q_1, q_2, \ldots, q_n \in k[x_1, x_2, \ldots, x_n]$. Note that there may be more than one way of collecting terms so there may be more than one choice for the q_i. As an example consider $x_1^2 x_2$ with $\alpha_1 = 1$, $\alpha_2 = 2$ so that $y_1 = x_1 - 1$ and $y_2 = x_2 - 2$. Then

$$\begin{aligned} x_1^2 x_2 &= (y_1 + 1)^2 (y_2 + 2) \\ &= y_1^2 y_2 + 2y_1^2 + 2y_1 y_2 + 4y_1 + y_2 + 2 \end{aligned}$$
.

Now the last term can written as

$$y_1(y_1 y_2 + 2y_1 + 2y_2 + 4) + y_2 + 2$$

or as

$$y_1(2y_1 + 2y_2 + 4) + y_2(y_1^2 + 1) + 2$$

as well as many other ways, all of which have the form ($\dagger$). Now using the fact that f is a linear combination of power products of the x_i we see that

$$f = y_1 g_1 + y_2 g_2 + \cdots + y_n g_n + f(\alpha_1, \alpha_2, \ldots, \alpha_n)$$

for some $g_1, g_2, \ldots, g_n \in k[x_1, x_2, \ldots, x_n]$. Finally we have

$$f(x_1, x_2, \ldots, x_n) - f(\alpha_1, \alpha_2, \ldots, \alpha_n) = (x_1 - \alpha_1)g_1 + (x_2 - \alpha_2)g_2 + \cdots + (x_n - \alpha_n)g_n.$$

We will use this fact later on when we look at the solution of simultaneous polynomial equations.

Before leaving this section it is worth noting that it provides an excellent illustration of a process that happens quite frequently in Mathematical subjects. We have a result for a special case (here the remainder theorem for univariate polynomials). This result can be derived quite easily from Euclid's Algorithm. However the view implied by this proof does not generalize (Euclid's Algorithm does not hold for multivariate polynomials). However a little more work shows that we can find another proof that gives us not only another view of the result but also generalizes. Indeed this process of generalization will reach a very impressive level when we look at Gröbner bases since these can be seen as a generalization of Euclid's Algorithm to multivariate polynomials.

4.7.8 Rational Expressions

Let k be any field. Given the polynomial ring $k[x_1, \ldots, x_n]$ we can form the set

$$k(x_1, \ldots, x_n) = \{p/q \mid p, q \in k[x_1, \ldots, x_n] \ \& \ q \neq 0\}.$$

Two elements p/q and p'/q' of $k(x_1, \ldots, x_n)$ are said to be *equal* if and only if $pq' - p'q = 0$ as a polynomial in $k[x_1, \ldots, x_n]$. We define addition and multiplication by

$$\begin{aligned}(p/q) + (p'/q') &= (pq' + p'q)/qq',\\ (p/q)(p'/q') &= pp'/qq'.\end{aligned}$$

(Can you prove that these definitions are unambiguous?) Given these operations $k(x_1, \ldots, x_n)$ becomes a field and is called the *field of rational expressions* (or *rational functions* or *quolynomials*) in $x_1, \ldots, x_n$ over k.

Just as in the case of polynomials it is common practise to regard an expression such as $(x^2 - 1)/(x - 1)$ as a function. But this can be rather confusing because here the expression $x + 1$ does not denote the same function since the latter is defined at $x = 1$ while the former is not. (Of course the two functions are both defined at all other points and have equal values.) Note however that as elements of $k(x)$ the two expressions are equal since $(x^2 - 1) = (x + 1)(x - 1)$ as polynomials. Computer Algebra systems work with such expressions in the algebraic rather than the functional sense. Despite this it is common mathematical practice to call such objects *rational functions* rather than rational *expressions* as we have done. See also [21], pp. 72–74.

4.7.9 Representation of Polynomials and Rational Expressions

First of all we deal with polynomials. We are considering the elements of $R[x_1, \ldots, x_n]$ where R is a ring (for whose elements we have a representation). Representations can be classified as recursive or distributive either of which may be dense or sparse. This gives us four possibilities, the choice depends on the application at hand. The most common representation is the recursive sparse one although for Gröbner bases a distributed sparse one is most useful.

Recursive Representation

This is just an expression of the isomorphism

$$R[x_1, \ldots, x_n] \cong R[x_1, \ldots, x_{n-1}][x_n]$$

and we regard x_n as the main indeterminate. For example

$$f = 3xy^2 + 2y^2 - 4x^2y + y - 1$$

may be represented as

$$(3x + 2)y^2 + (-4x^2 + 1)y + (-1)y^0$$

where y is the main indeterminate. In general we use $\sum c_i x_n^i$ where the c_i are polynomials which are themselves represented in a similar format. The Axiom type is `UP(x,UP(y,INT))` (here and elsewhere `INT` can be replaced by any integral domain). Note that Axiom prints back the polynomial in expanded form but it is held as indicated as a use of `coefficients f` shows.

Distributive Representation

We consider the power products in the given indeterminates e.g., $x_1^2x_3x_5^7$. We pick a total order on the power products such that 1 (which stands for $x_1^0\cdots x_n^0$) is least and for each power product there are only finitely many less than it. We may now write a polynomial $p(x_0,\ldots,x_n)$ as

$$p(x_0,\ldots,x_n)=\sum_{t\le\bar{t}} c_t t$$

where $c_t\in R$ for each t. As an example of a suitable order we could first sort according to degree and within each degree use the lexicographic order: we order the indeterminates, e.g.,

$$x_1 >_L x_2 >_L \cdots >_L x_n$$

and then

$$x_1^{i_1}\cdots x_n^{i_n} >_L x_1^{j_1}\cdots x_n^{j_n}$$

if and only if there is a k such that $i_l = j_l$ for $1\le l<k$ and $i_k>j_k$. This is called the *graded lexicographic order*. By contrast if we order power products purely lexicographically then we do not have a suitable order because there can be infinitely many power products less than a given one (see the example on p. 69). With f as above, the Axiom type is `DMP([x,y],INT)` (an alternative is `HDMP([x,y],INT)` which imposes a different order on power products.

Dense Representations

In a dense representation we record all the coefficients up to the highest degree of the main indeterminate or the highest power product. Thus for a recursive representation we might have

$$\sum_{i=0}^{m} c_i x^i \longleftrightarrow (c_0,\ldots,c_m)$$

and for a distributed representation

$$\sum_{t\le\bar{t}} c_t t \longleftrightarrow (c_1, c_{t_1},\ldots,c_{\bar{t}}),$$

where $(\ldots)$ denotes a list or array. The problem is that this representation can lead to a great deal of wasted space, e.g., consider $x^{1000}+1$ or x^4y^7+x+1.

Exercise 4.22 *How many power products of degree d are there in the indeterminates x, y? Generalize to power products in n indeterminates.*

Sparse Representations

For these we drop all zero coefficients. This means that with each non-zero coefficient we must record the corresponding degree or power product. For example

$$x^{1000}+1 \longleftrightarrow ((1,1000),(1,0)),$$
$$x^4y^7+2x+1 \longleftrightarrow ((1,(4,7)),(2,(1,0)),(1,(0,0))).$$

In the second example a power product $x_1^{e_1}\cdots x_n^{e_n}$ is represented by $(e_1,\ldots,e_n)$. In general Axiom uses a sparse representation.

Rational Expressions

The obvious representation for these is as a pair of polynomials consisting of the numerator and denominator. If the numerator is in normal form then we also have a normal form for rational expressions (since $f/g = 0$ iff $f = 0$, of course we must have $g \neq 0$ for f/g to be a rational expression at all). It is tempting, by analogy with the rational numbers, to remove $\gcd(f, g)$ from the numerator and denominator of f/g. However this can be very unwise: for example $(1 - x^n)/(1 - x)$ is very compact and requires little storage even for large n. Removing the gcd gives us $1 + x + \cdots + x^{n-1}$ which is expensive in storage space and clutters up any output in a nasty way! The situation becomes even worse when multivariate polynomials are involved. We also have to bear in mind that polynomial gcd's are fairly costly to compute (compared to integer ones). However the default behaviour of Axiom is to take out the gcd, by contrast Maple does not.

Exercise 4.23 *Consider rational expressions in one indeterminate and rational coefficients. Even if we remove the gcd from the numerator and denominator we still do not have a canonical form. Give a method that leads to a canonical form which uses only integer coefficients. (Here we assume that integers are held in a canonical form, which is always the case.)*

Intermediate Expression Swell

We know from §4.7.6 that computing the gcd of

$$\begin{aligned} A &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, \\ B &= 3x^6 + 5x^4 - 4x^2 - 9x + 21, \end{aligned}$$

using Euclid's algorithm over the integers leads to rather large integers, even though the input and output are small (we are justified in claiming that the output is small because any non-zero number, e.g., 1, is a valid answer, remember that we are working with coefficients over $\mathbb{Q}$). This phenomenon of *intermediate expression swell* is such a major problem in computer algebra that it is worth seeing another example of it.

Some systems (e.g., Maple) use a sum of products representation for polynomials (and other expressions). In this representation we take a sum over products of polynomials rather than just a sum over monomials. This representation is compact but it does mean that we cannot tell just by looking at it whether or not a given representation yields the zero polynomial. In order to achieve this it appears that we must expand terms—at least nobody knows of any way round this. Unfortunately this process can lead to an exponential increase in the number of terms produced. The following is an example of this. The *Vandermonde determinant* of n indeterminates $x_1, x_2, \ldots, x_n$ is defined by:

$$V(x_1, x_2, \ldots, x_n) = \begin{vmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \ldots & x_n^{n-1} \end{vmatrix}.$$

It can be shown that

$$V(x_1, x_2, \ldots, x_n) = \prod_{1 \le i < j \le n} (x_j - x_i).$$

(Thus $V(x_1, x_2, \dots, x_n) = 0$ if and only if $x_i = x_j$ for some $i \neq j$.) Now put

$$Z(x_1, x_2, \dots, x_{n+1}) = \begin{vmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{n+1} \\ x_1^2 & x_2^2 & \dots & x_{n+1}^2 \\ \vdots & \vdots & & \vdots \\ x_1^{n-1} & x_2^{n-1} & \dots & x_{n+1}^{n-1} \end{vmatrix}.$$

This is 0 since the first two rows are equal. However expanding along the first row we have

$$\begin{aligned} Z(x_1, x_2, \dots, x_{n+1}) &= \sum_{i=1}^{n+1} (-1)^{i+1} V(x_1, \dots, \hat{x}_i, \dots, x_{n+1}) \\ &= \sum_{i=1}^{n+1} (-1)^{i+1} \prod_{\substack{1 \le j < k \le n+1 \\ j,k \neq i}} (x_k - x_j), \end{aligned}$$

where $x_1, \dots, \hat{x}_i, \dots, x_{n+1}$ denotes the sequence $x_1, x_2, \dots, x_{n+1}$ with x_i deleted. Now the last expression is a perfectly good sum of products representation but any attempt at expanding it leads to $n!$ terms for each summand and cancellation takes place only between summands!

We therefore have a difficult choice: systems that expand automatically will face a huge penalty at some point. Systems that do not expand automatically leave open the possibility of undetected divisions by 0 and the user must look out for such difficulties. The next exercise shows a probabilistic approach to deciding equality to 0 without expanding.

Exercise 4.24 *Let $p(x_1, x_2, \dots, x_n)$ be a non-zero polynomial with coefficients from $\mathbb{Q}$ and put*

$$\mathrm{V}(p) = \{\, (a_1, a_2, \dots, a_n) \in \mathbb{Q}^n \mid p(a_1, a_2, \dots, a_n) = 0 \,\}.$$

It is a fact that if we pick the a_i randomly (from some range) then the probability that we obtain a member of $\mathrm{V}(p)$ (i.e., a root of p) is 0. (Think of algebraic curves in 2 dimensions and surfaces in 3 dimensions.) Can you use this to develop a fast randomized test for a polynomial expression being zero? To be more precise if the test tells us that the polynomial is non-zero then the answer must be correct. If on the other hand the test tells us that the polynomial is zero then the answer must be correct with high probability. (See J. T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, J. ACM, **27***, 4, (1980) 701–717.)*