Algorithmic Mathematics

a web-book by Leonard Soicher & Franco Vivaldi

This is the textbook for the course MAS202 Algorithmic Mathematics. This material is in a fluid state —it is rapidly evolving— and as such more suitable for on-line use than printing. If you find errors, please send an e-mail to: F.Vivaldi@qmul.ac.uk.

Last updated: January 18, 2017

Preface

This text contains sufficient material for a one-semester course in mathematical algorithms, for second year mathematics students. The course requires some exposure to the basic concepts of discrete mathematics, but no computing experience.

The aim of this course is twofold. Firstly, to introduce the basic algorithms for computing exactly with integers, polynomials and vector spaces. In doing so, the student is expected to learn how to think algorithmically and how to design and analyze algorithms.

Secondly, to provide a constructive approach to abstract mathematics, algebra in particular. When introducing the elements of ring and field theory, algorithms offer concrete tools, constructive proofs, and a crisp environment where the benefits of rigour and abstraction become tangible.

We shall write algorithms in a straightforward language, which incorporates freely standard mathematical notation. The specialized constructs are limited to the if-structure and the while-loop, which are universal.

Exercises are provided. They have a degree of difficulty comparable to that of examination questions. Some of the exercises consist of short essays; in this context, the notation [$\not\in$] indicates that mathematical symbols are not permitted in the essay. Starred sections contain optional material, which is not examinable.

The Algorithmic Mathematics's web page is:

`algorithmicmathematics.com`

Contents

Chapter 1

Basics

Informally, an *algorithm* is a finite sequence of unambiguous instructions to perform a specific task. In this course, algorithms are introduced to solve problems in discrete mathematics.

An algorithm has a *name,* begins with a precisely specified *input,* and terminates with a precisely specified *output.* Input and output are *finite sequences* of mathematical objects. An algorithm is said to be *correct* if given input as described in the input specifications: (i) the algorithm terminates in a finite time; (ii) on termination the algorithm returns output as described in the output specifications.

Example 1.1.

`Algorithm SumOfSquares`
`INPUT`: $a, b, \in \mathbb{Z}$
`OUTPUT`: c, where $c = a^2 + b^2$.
  $c := a^2 + b^2$;
  `return` c;
`end`;

The name of this algorithm is `SumOfSquares.` Its input and output are integer sequences of length 2 and 1, respectively.

In this course all algorithms are *functions,* whereby the output follows from the input through a *finite sequence* of *deterministic steps;* that is, the outcome of each step depends only on the outcome of the previous steps. In the example above, the *domain* of `SumOfSquares` is the the set of integer pairs, the *co-domain* is the set of non-negative integers, and the *value* of `SumOfSquares`$(-2, -3)$ is 13. This function is clearly *non-injective;* its value at (a, b) is the same as that at $(-a, -b)$, or (b, a), etc. It also happens to be *non-surjective* (see exercises).

(Algorithms do not necessarily represent functions. The instruction: 'Toss a coin; if the outcome is head, add 1 to x, otherwise, do nothing' is legitimate and unambiguous, but not *deterministic.* The output of an algorithm containing such instruction is not a function of the input alone. Algorithms of this kind are called *probabilistic.*)

It is expedient to regard the flow from input to output as being parametrized by *time.* This viewpoint guides the intuition, and even when estimating run time is not a concern, time considerations always lurk in the background (will the algorithm terminate? If so, how many operations will it perform?).

1.1 The language of algorithms

The general form of an algorithm is the following

`Algorithm` ⟨ *algorithm name* ⟩
`INPUT:` ⟨ *input specification* ⟩
`OUTPUT:` ⟨ *output specification* ⟩
⟨ *statement* ⟩;
⟨ *statement* ⟩;
⋮
⟨ *statement* ⟩;
`end`;

(A quantity in angle brackets defines the type of an object, rather than the object itself.)

The heart of the algorithm is its *statement sequence,* which is implemented by a language. The basic elements of any algorithmic language are surprisingly few, and use a very standard syntax. We introduce them in the next sections.

1.1.1 Expressions

In this course, expressions are the data processed by an algorithm. We do not require a precise definition of what we regard to be a valid expression, since we shall consider only expressions of very basic type.

We begin with *arithmetical* and *algebraic* expressions, which are formed by assembling in familiar ways numbers and arithmetical operators. Algebraic expressions differ from arithmetical ones in that they contain *indeterminates* or *variables.* All expressions considered here will be *finite*, e.g.,

$$1+\cfrac{1}{2+\cfrac{1}{3+\cfrac{1}{4+\cfrac{1}{5}}}} \qquad\qquad \frac{(x-y)(x+y)(x^2+y^2)(x^4+y^4)}{x^8-y^8}$$

By combining expressions, we can construct composite expressions representing sequences, sets, etc.

$$(x-1,\, x+1,\, x^2+x+1,\, x^2+1) \qquad\qquad \left\{0,\, 1,\, \frac{1}{2},\, \frac{1}{3},\, \frac{2}{3},\, \frac{1}{4},\, \frac{3}{4}\right\}.$$

We are not concerned with the practicalities of evaluating expressions, and assume that a suitable computational engine is available for this purpose. In this respect, an expression such as

‘ the smallest prime number greater than 1000^{1000} ’

is perfectly legitimate, since its value is unambiguously defined (because there are infinitely many primes). We also ignore the important and delicate problem of agreeing on how the value of an expression is to be represented. For instance, the value of the expression ‘the largest real solution of $x^4 + 1 = 10x^2$’ can be represented in several ways, e.g.,

$$\sqrt{2} + \sqrt{3} = \sqrt{5 + 2\sqrt{6}} = 3.14626436994197234 23\ldots$$

and there is no a priori reason for choosing any particular one.

1.1.2 Assignment statement

The assignment statement allows the management of data within an algorithm.

SYNTAX: ⟨ *variable* ⟩:=⟨ *expression* ⟩;

EXECUTION: evaluate the expression and assign its value to the variable.

Assignement statements should not be confused with equations. Thus, $x := x + 1$; is an assignment statement (read: ‘x *becomes* $x + 1$’) which has the effect of increasing the value of x by 1. By contrast $x = x+1$ is an equation (which happens to have no solution).

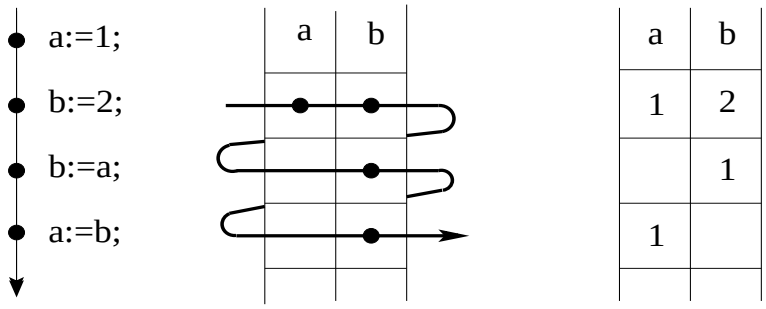

Figure 1.1: Tracing a sequence of statements; four assignement statements lead to four entries in the table. On exit, both a and b have value 1, the bottom entry in their respective columns.

We *trace* the above statement sequence by keeping track of the variables’ values with a table. Time flows from left to right within a row, and from top to bottom within the table; each entry corresponds to one evaluation of the corresponding expression, so there are as many entries as there are evaluations (Figure 1.1).

Example 1.2. We trace the following statement sequence

$i := 3;$
$i := (i^4 + 10)/13;$
$n := 4 - i;$
$S := (i, |n|);$
$n := 3n + i;$
$i := i + n;$
$S := (|i|, S);$

i	n	S
3		
7	-3	$(7, 3)$
	-2	
5		$(5, (7, 3))$

Persuade yourself that re-arranging the columns may change the number of rows.

Before a variable can be used in an expression it must have a value, which is either assigned on input or by an assignment statement. Assigning a value on input works as follows. Suppose we have

`Algorithm A`
`INPUT`: $\langle\, a_1, \ldots, a_k,$ *and their properties* $\rangle$
`OUTPUT`: $\langle$ *output specification* $\rangle$
$\langle$ *statement sequence* $\rangle$
`end`;

Then $a_1, \ldots, a_k$ are variables for algorithm `A`. When `A` is executed, it must be given an input sequence $v_1, \ldots, v_k$ of values. Then a_1 is assigned the value v_1, a_2 is assigned the value v_2, etc., ..., a_k is assigned the value v_k, before the statement sequence of `A` is executed (bearing in mind that the values assigned may be indeterminates). Thus, assigning values on input is analogous to evaluating `A`, as a function, at those values; this process could be replaced by a sequence of assignment statements at the beginning of the algorithm. This would be, of course, very inefficient, being equivalent to defining the function at a single point.

1.1.3 Return statement

The `return` statement produces an output sequence.

SYNTAX: `return` $\langle$ *expression* 1 $\rangle, \ldots, \langle$ *expression* k $\rangle$

EXECUTION: first *expression* 1, ..., *expression* k are evaluated, obtaining values $v_1, \ldots, v_k$, respectively. Then the sequence $(v_1, \ldots, v_k)$ is returned as the output of the algorithm, and the algorithm is terminated.

We treat an output sequence (v_1) of length 1 as a simple value v_1.

Example 1.3. The expressions

$i := 5;$
`return` 3^2, $i + 5$, $27;$

returns the sequence $(9, 10, 27)$ as output.

1.1.4 If-structure

A *boolean constant* or a *boolean value*, is an element of the set {TRUE, FALSE} (often abbreviated to $\{T, F\}$). A *boolean* (or *logical*) expression is an expression that evaluates to a boolean value. We postpone a detailed description of boolean expressions until section 1.2. For the moment we consider expressions whose evaluation involves testing a single (in)equality, called *relational expressions*. For instance, the boolean value of

$$10^3 < 2^{10}, \qquad 9^3 + 10^3 \neq 1^3 + 12^3$$

is TRUE and FALSE, respectively. Evaluating more complex expressions such as

$$2^{13,466,917} - 1 \quad \text{is prime}, \qquad 2^{(2^{13,466,917} - 1)} - 1 \quad \text{is prime} \tag{1.1}$$

can be reduced to evaluating *finitely many* relational expressions, although this may involve formidable difficulties. Thus several weeks of computer time were required to prove that the leftmost expression above is TRUE, giving the largest prime known to date. This followed nearly two and half years of computations on tens of thousands of computers, to test well over 100,000 unsuccessful candidates for the exponent. By contrast, the value of the rightmost boolean expression is not known, and may never be known.

The *if-structure* makes use of a boolean value to implement decision-making in an algorithmic language. It is defined as follows:

SYNTAX:

`if` ⟨ *boolean expression* ⟩ `then`
⟨ *statement-sequence 1* ⟩
`else`
⟨ *statement-sequence 2* ⟩
`fi`;

EXECUTION: if the value of the *boolean expression* is TRUE, the *statement-sequence 1* is executed, and not *statement-sequence 2*. If the boolean expression evaluates to FALSE, then *statement-sequence 2* is executed, and not *statement-sequence 1*.

The boolean expression that controls the if-structure is called the *if control expression*.

Example 1.4.

```
if i > 0 then
  t := t − i;
else
  t := t + i;
fi;
```

We remark that an if-structure is logically equivalent to a single statement. A variant of the above construct is given by

SYNTAX:

```
if ⟨ boolean expression ⟩ then
  ⟨ statement-sequence ⟩
fi;
```

The execution is the same as the execution of

```
if ⟨ boolean expression ⟩ then
  ⟨ statement-sequence ⟩
else
fi;
```

which is obtained from the general form by having an empty statement sequence, which does nothing.

Example 1.5.

```
if θ > 0 then
  θ := 5 θ (1 − θ);
fi;
```

If the above if-statement starts execution with $\theta \leq 0$, then the statement has no effect.

Example 1.6.

```
Algorithm MinimumInt
INPUT: a, b, ∈ ℤ
OUTPUT: c, where c is the minimum of a and b.
if a < b then
   return a;
else
   return b;
fi;
end;
```

The input and output are integer sequences of length 2 and 1, respectively. When regarded as a function, the domain of `MinimumInt` is the the set $\mathbb{Z}^2$ of integer pairs, the co-domain

is $\mathbb{Z}$, and the value of `MinimumInt`$(-2, 12)$ is -2.

1.1.5 While-loops

A loop is the structure that implements repetition. Loops come in various guises, the most basic of which is the while-loop.

SYNTAX:

`while` ⟨ *boolean-expression* ⟩ `do`
⟨ *statement sequence* ⟩
`od`;

EXECUTION:

(*i*) Evaluate the boolean expression.

(*ii*) If the boolean expression evaluates to TRUE, then execute the statement sequence, and repeat from step (*i*). If the boolean expression is FALSE, do not execute the statement sequence but terminate the `while`-loop and continue the execution of the algorithm after the "`od`".

The boolean expression that controls the loop is called the *loop control expresssion*.

Example 1.7. The loop

`while` $0 \neq 1$ `do`
`od`;

will run forever and do nothing. The loop

`while` $0 = 1$ `do`
$\ldots$
`od`;

will not run at all.

Example 1.8. We trace the following statements

$i := 2$;
$k := i$;
`while` $i^3 > 2^i$ `do`
$i := i + 3$;
$k := k - i$;
`od`;

Besides the variables i and k, we also trace the value of the boolean expression that

controls the loop

i	k	$i^3 > 2^i$
2	2	TRUE
5	-3	TRUE
8	-11	TRUE
11	-22	FALSE

This example illustrates a general property of loops: on exit, the value of the loop control expression is necessarily FALSE, lest the loop would not be exited. By contrast, the boolen expression controlling an `if`-statement, can have any value on exit (Figure 1.2).

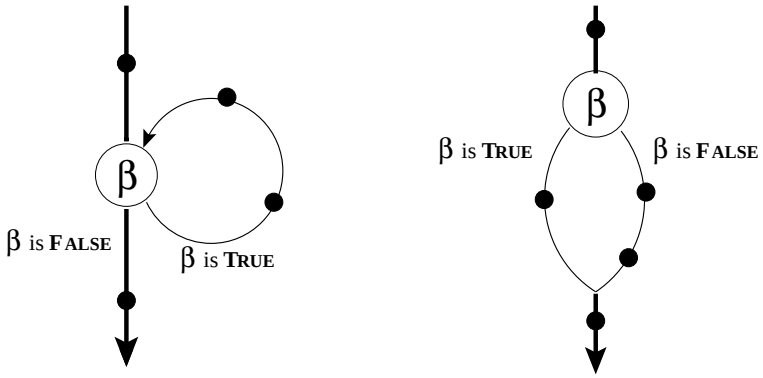

Figure 1.2: The basic structures of algorithms: loops and if-statements. The evaluation of the boolean expression β gives rise to a bifurcation point in the execution of the algorithm.

It may be difficult, even impossible, to decide how many times a given `while`-loop will be executed (or indeed, whether the loop will terminate at all, see section 2.6 for an example).

As an example, consider the following algorithm

`Algorithm NextPrime`
`INPUT`: n, a positive integer.
`OUTPUT`: p, where p is the least prime greater than n.
$p := n + 1$;
`while` p is not prime `do`
  $p := p + 1$;
`od`;
`return` p;
`end`;

Since the number of primes is infinite, we know the loop will terminate, but we do not know how many times it will be executed. Indeed, it is possible to show (see exercises) that arbitrarily large gaps between consecutive primes exist, hence arbitrarily large number of repetitions of the above loop.

A structure is *nested* if it appears inside another structure. Nesting is a means of

constructing complex algorithms from simple ingredients, so tracing a nested structure can be laborious.

Example 1.9.

```
while ⟨ boolean-expression ⟩ do
    while ⟨ boolean-expression ⟩ do
        ⟨ expression ⟩;
    od;
    ⟨ expression ⟩;
    if ⟨ boolean-expression ⟩ then
        ⟨ expression ⟩;
    else
        ⟨ expression ⟩;
    fi;
od;
```

In this example, the body of the outer loop consists of three expressions, two of which are structures (Figure 1.3).

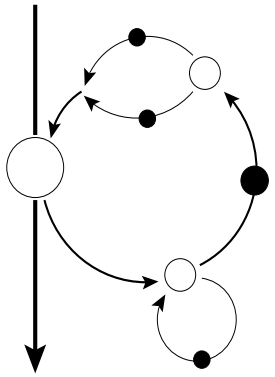

Figure 1.3: Nested structures: a loop containing a loop and an if-statement.

1.2 Boolean calculus

Boolean expressions may be constructed from boolean constants and relational expressions by means of *boolean operators*. This process is analogous to the construction of arithmetical expressions from arithmetical constants (i.e., numbers) and operators (+, −, etc.).

The basic boolean operators are NOT, and AND. The operator NOT is *unary,* that is, it takes just one boolean operand and produces a boolean result. The operator AND is *binary,* which means that it acts on two operands and produces a boolean result.

The following table, called a *truth table,* defines the value of NOT and AND on all possible choices of boolean operands

NOT P	P
F	T
T	F

P	P AND Q	Q
T	T	T
T	F	F
F	F	T
F	F	F

Other binary operators may be constructed from the above two. The most commonly used are OR, $\Longrightarrow$, and $\Longleftrightarrow$. We define them directly with truth tables, although they can also be defined in terms of NOT and AND (see remarks following proposition 1, below).

P	P OR Q	Q
T	T	T
T	T	F
F	T	T
F	F	F

P	$P \Longrightarrow Q$	Q
T	T	T
T	F	F
F	T	T
F	T	F

P	$P \Longleftrightarrow Q$	Q
T	T	T
T	F	F
F	F	T
F	T	F

(1.2)

We note that if $A =$ TRUE and $B =$ FALSE, then

$$(A \Longrightarrow B) \neq (B \Longrightarrow A)$$

that is, the operator $\Longrightarrow$ is *non-commutative.*

Example 1.10.

$P := 2 < 3$;
$Q := 2 \geq 3$;
$R := (P \text{ OR } Q) \Longrightarrow (P \text{ AND } Q)$;
$S := (P \text{ AND } Q) \Longrightarrow (P \text{ OR } Q)$;

P	Q	P OR Q	P AND Q	R	S
T	F	T	F	F	T

Proposition 1 *For all* $P, Q, \in \{$TRUE, FALSE$\}$*, the following holds*

(*i*) NOT $(P \text{ OR } Q) =$ (NOT P) AND (NOT Q)

(*ii*) NOT $(P \text{ AND } Q) =$ (NOT P) OR (NOT Q)

(*iii*) $P \Longrightarrow Q =$ (NOT P) OR Q

$(iv)\ P \iff Q = ((P \Longrightarrow Q)\,\text{AND}\,(Q \Longrightarrow P))$.

Proof: The proof consists of evaluating each side of these equalities for all possible P, Q. We prove (iv). The other proofs are left as an exercise. The left-hand side of (iv) was given in (1.2). Let $\mathcal{R}$ be the right-hand side. We compute $\mathcal{R}$ explicitly.

P	$P \Longrightarrow Q$	$\mathcal{R}$	$Q \Longrightarrow P$	Q
T	T	T	T	T
T	F	F	T	F
F	T	F	F	T
F	T	T	T	F

Hence the left hand side is equal to the right hand side for all $P, Q \in \{\text{TRUE}, \text{FALSE}\}$. ■

The statements (i) and (ii) are known as *De Morgan's laws.* Using Proposition 1, one can express the operators OR, $\Longrightarrow$, $\iff$ in terms of NOT and AND (see exercises).

1.3 Characteristic functions

Characteristic functions link boolean quantities to sets.

Def: A *characteristic function* is a function assuming boolean values.

Let X be a set and $A \subseteq X$. The function

$$\mathcal{C}_A : X \to \{\text{TRUE}, \text{FALSE}\} \qquad x \mapsto \begin{cases} \text{TRUE} & \text{if } x \in A \\ \text{FALSE} & \text{if } x \notin A \end{cases}$$

is called the *characteristic function of* A (in X). Conversely, let $f : X \to \{\text{TRUE}, \text{FALSE}\}$ be a characteristic function. Then $f = \mathcal{C}_A$, where $A = f^{-1}(\text{TRUE})$. So there is a bi-unique correspondence between the characteristic functions defined on X, and the subsets of X.

Theorem 2 *Let X be a set, and let $A, B \subseteq X$. The following holds*

$$\begin{array}{lrcl} (i) & \text{NOT}\,\mathcal{C}_A & = & \mathcal{C}_{X\setminus A} \\ (ii) & \mathcal{C}_A\ \text{AND}\ \mathcal{C}_B & = & \mathcal{C}_{A\cap B} \\ (iii) & \mathcal{C}_A\ \text{OR}\ \mathcal{C}_B & = & \mathcal{C}_{A\cup B} \\ (iv) & \mathcal{C}_A \Longrightarrow \mathcal{C}_B & = & \mathcal{C}_{X\setminus(A\setminus B)} \\ (v) & \mathcal{C}_A \Longleftrightarrow \mathcal{C}_B & = & \mathcal{C}_{(A\cap B)\cup(X\setminus(A\cup B))} \end{array}$$

Proof: To prove (i) we note that the function $x \mapsto \text{NOT}\,\mathcal{C}_A(x)$ evaluates to TRUE if $x \notin A$ and to FALSE otherwise. However, $x \notin A \iff x \in (X \setminus A)$, from the definition of difference between sets.

Next we prove (iv). Let $P := x \in A$ and $Q := x \in B$. Then, from (1.2), we have that $P \Longrightarrow Q$ is TRUE precisely when $(P, Q) \neq (\text{TRUE}, \text{FALSE})$. This means that x is such that the expression

$$\text{NOT}\,(x \in A\,\text{AND}\,(\,\text{NOT}\,x \in B)) = \text{NOT}\,(x \in A \setminus B) = x \in (X \setminus (A \setminus B))$$

is TRUE, from part (i). But the rigthmost expression is the definition of the characteristic function of the set $X \setminus (A \setminus B)$.

The proof of (ii), (iii), (v) is left as an exercise. ■

Example 1.11. The functions $x \mapsto (x \geq a)$ and $x \mapsto (x \leq b)$ are the characteristic functions of two rays. If $a \leq b$, then $x \mapsto ((x \geq a) \text{ AND } (x \leq b))$ is the characteristic function of the closed interval $[a, b]$.

Exercises

Exercise 1.1. Use a table to trace the values of a, b, $b > 0$, $a > b$, as the statement sequence below is executed.

```
a := 15;
b := 10;
while b > 0 do
   while a > b do
      a := a − 2;
   od;
   b := b − 3;
od;
```

(The table should have 4 columns and 12 rows.)

Exercise 1.2. Use a table to trace the execution of the following statement sequence:

```
x := 3;
y := 0;
while |x| ≥ |y| do
    if x  is even then
       x := x/2 − y;
    else
       x := (x + 1)/2 − y;
       y := y − x;
    fi;
od;
```

Exercise 1.3. Prove that the function `SumOfSquares`: $\mathbb{Z}^2 \to \mathbb{N}$ of Example 1.1 is not surjective.

Exercise 1.4. Evaluate each of the following boolean expressions

(a) $((2^6 > 70) \text{ OR } (2^8 > 250)) \text{ AND } (2^{10} > 1000)$

(b) $\dfrac{21}{34} < \dfrac{34}{55} \Longrightarrow \dfrac{34}{55} < \dfrac{55}{89}$

(*c*) $\sqrt{2} > \frac{7}{5}$ AND $\sqrt{2} < \frac{17}{12}$

(*d*) $\frac{1}{(\sqrt{3}-2)^2} < 14$

(*e*) (('93 is prime' $\Longrightarrow$ '97 is prime') OR '87 is prime') $\iff$ '91 is prime'

(*f*) '47 is the sum of two squares'.

Use only integer arithmetic. In part (*f*) aim for precision and conciseness.

Exercise 1.5. Show that

$$(P \Longrightarrow Q) = ((\,\text{NOT}\,Q) \Longrightarrow (\,\text{NOT}\,P))$$

for all $P, Q \in \{\text{TRUE}, \text{FALSE}\}$.

Exercise 1.6. Describe in one sentence. Get the essence, not the details. [$\not\in$]

$$X \text{ OR } Y \;=\; \text{NOT}\,((\,\text{NOT}\,X) \text{ AND } (\,\text{NOT}\,Y)) \qquad \forall\, X, Y \in \{\text{TRUE}, \text{FALSE}\}.$$

Exercise 1.7. Complete the proof of Proposition 1, hence express OR, $\Longrightarrow$, and $\iff$ in terms of NOT and AND.

Exercise 1.8. Complete the proof of Theorem 2.

Exercise 1.9.

(*a*) Write an algorithm to the following specifications:

`Algorithm Min3Int`

`INPUT`: $a, b, c \in \mathbb{Z}$.

`OUTPUT`: m, where m is the minimum of a, b, c.

(*b*) Explain in one sentence [$\not\in$] what goes wrong in your algorithm if the input specification is changed to `INPUT:` $a, b, c \in \mathbb{C}$, while leaving everything else unchanged.

Exercise 1.10. Explain concisely what this algorithm does, and how. [$\not\in$]

`INPUT`: $x, e \in \mathbb{Z}, x \neq 0, e \geq 0$.
`OUTPUT`: ??
$a := 1$;
`while` $e > 0$ `do`
  $a := a \cdot x$;
  $e := e - 1$;
`od`;
`return` a;
`end`;

Exercise 1.11. Consider the following algorithm

```
Algorithm A
INPUT: n, S, where n is a positive integer and S = (S_1, ..., S_n)
       is a sequence of n integers.
OUTPUT: ??
Z := 0;
i := 1;
while i ≤ n do
  j := i;
  while j ≤ n do
    Z := Z + S_i;
    j := j + 1;
  od;
  i := i + 1;
od;
return Z;
end;
```

(*a*) Write the output specifications.

(*b*) Rewrite the algorithm in such a way that it has only one loop.

Exercise 1.12. Write an algorithm to the following specifications

```
Algorithm IsSumOfSquares
INPUT: x ∈ ℤ
OUTPUT: TRUE, if x is the sum of two squares, FALSE otherwise.
```

The algorithm should use integer arithmetic only.

Chapter 2

Arithmetic

2.1 Divisibility of integers

Let $a, b, \in \mathbb{Z}$. We say that b *divides* a (and write $b|a$) if $a = bc$ for some $c \in \mathbb{Z}$. The statements that a is a *multiple* of b and b is a *divisor* of a have the same meaning.

Example 2.12. The expression $b|a$ is boolean, while the expressions b/a is arithmetical.

$$3|6 = \textsc{True}, \qquad 3/6 = 1/2, \qquad 6|3 = \textsc{False}, \qquad 0|5 = \textsc{False}.$$

The expression $5/0$ is undefined.

Let $a \in \mathbb{Z}$. Then

- $1|a$ since $a = 1 \cdot a$
- $a|a$ since $a = a \cdot 1$
- $a| - a$ since $-a = a \cdot -1$
- $a|0$ since $0 = a \cdot 0$.

Note that if $0|a$ then $a = 0 \cdot c$ for some $c \in \mathbb{Z}$, in which case $a = 0$.

Theorem 3 *Let $b \in \mathbb{Z}$, with $b \neq 0$. Then there exist unique integers q, r such that*

$$a = bq + r \qquad 0 \leq r < |b|.$$

The proof will be found in chapter 7.

We denote such q and r by $a \,\textsc{div}\, b$ and $a \,\textsc{mod}\, b$, respectively. If $a \geq 0$ and $b > 0$, we can calculate $a \,\textsc{div}\, b$ and $a \,\textsc{mod}\, b$ by *long division.* When $a \geq 0$ and $b > 0$, then q is the quotient of division of a by b, that is, the integer part of a/b (the largest integer not exceeding a/b). The non-negative integer r is the remainder of such integer division, that is, r/b is the fractional part of a/b. By the same token, $b \cdot q$ is the largest multiple of b not exceeding a, etc. When a or b are negative, the value of q and r is a bit less straightforward.

Example 2.13. Check these calculations carefully

$$\begin{aligned} 293 \text{ DIV } 8 &= 36 & 293 \text{ MOD } 8 &= 5 \\ -293 \text{ DIV } 8 &= -37 & -293 \text{ MOD } 8 &= 3 \\ 293 \text{ DIV } -8 &= -36 & 293 \text{ MOD } -8 &= 5 \\ -293 \text{ DIV } -8 &= 37 & -293 \text{ MOD } -8 &= 3 \end{aligned}$$

Let $a, b \in \mathbb{Z}$, with $b \neq 0$. If $a \,\text{MOD}\, b = 0$, then $a = bq + 0$, for some $q \in \mathbb{Z}$, $\Longrightarrow b|a$. Conversely, if $b|a$, then $a = bq$ for some $q \in \mathbb{Z}$, $\Longrightarrow a = bq + 0 \Longrightarrow a \,\text{MOD}\, b = 0$.

Thus, $b|a \iff a \,\text{MOD}\, b = 0$.

Def: Let $n \in \mathbb{Z}$. We say that n is *even* if $2|n$; otherwise n is *odd.*

Thus, n is even iff $n \,\text{MOD}\, 2 = 0$, and odd iff $n \,\text{MOD}\, 2 = 1$.

Example 2.14. We construct the characteristic function of the integers divisible by a given integer m. The case $m = 0$ has to be treated separately.

```
Algorithm Multiple
INPUT: (x, m) ∈ Z^2.
OUTPUT: TRUE if x is a multiple of m, FALSE otherwise.
if m = 0 then
   return x = 0
else
   return x MOD m = 0
fi;
end;
```

2.2 Prime numbers

Def: An integer n is said to be prime if (i) $|n| \geq 2$ and (ii) the only divisors of n are $1, -1, n, -n$.

Example 2.15. The integers $-1, 9, 0$ are not prime. The integers $2, -2, -17$ are prime.

We have that n is prime iff $-n$ is prime (see exercise). Since $|n| = n$ or $|n| = -n$, we conclude that n is prime iff $|n|$ is prime. We now consider the problem of testing whether a given non-negative integer is prime.

Lemma 4 *Let $b, n \in \mathbb{Z}$. Then $b|n$ if and only if $-b|n$.*

Proof: $(\Longrightarrow)$: $b|n \Longrightarrow \exists\, c \in \mathbb{Z}$ s.t. $n = bc$. Thus, $n = (-b)(-c) \Longrightarrow -b|n$.

($\Longleftarrow$): $-b|n \implies -(-b)|n$ (from ($\Longrightarrow$)) $\implies b|n$. ■

Now let $n \in \mathbb{Z}$, $n \geq 0$. Then n is prime iff $n \geq 2$ and the only positive divisors of n are 1 and n. Furthermore, if $n > 0$ and $b \in \mathbb{Z}$, $b > n$, then $b \not| n$. Thus, if $n \geq 2$, then n is prime iff $i \not| n$ for $i = 2, \ldots n-1$. This gives us a crude way of testing if n is prime.

Lemma 5 *Let i, a be positive integers, and suppose $i|a$, $1 < i < a$, and $i^2 > a$. Then $\exists\, j \in \mathbb{Z}$ such that $j|a$ and $1 < j^2 < a$.*

Proof: Since $i|a$, there is a $j \in \mathbb{Z}$ such that $a = ij$. We see that $j|a$. Next, $1 < i < a \implies j = a/i > 1$. Now $a^2 = (ij)^2 = i^2 j^2$ and $i^2 > a \implies j^2 = a^2/i^2 < a^2/a = a$. ■

Proposition 6 *Let $a \in \mathbb{Z}$, $a \geq 2$. Then a is prime if and only if no integer i such that $1 < i^2 \leq a$ divides a.*

Proof: ($\Longrightarrow$): Suppose a is prime. Then the only positive divisors of a are 1 and a. As $a \geq 2$, we have $a^2 > a$, so there is no integer i such that $i|a$ and $1 < i^2 \leq a$.

($\Longleftarrow$): Suppose $i \not| a$ for each i with $1 < i^2 \leq a$. Then by the previous lemma, there is no i dividing a with $1 < i < a$ and $i^2 > a$. We conclude that $i \not| a$ for all i with $1 < i < a$, and since $a \geq 2$, a must be prime. ■

Let $a \in \mathbb{Z}$, $a \geq 0$. If a is even, then a is prime iff $a = 2$ (0 is not prime, 2 is prime, if a is even and $a > 2$, then $2|a$ and $1 < 2 < a$). If a is odd, then no even integer $2k$ divides a, because otherwise $a = 2kl$ for some $l \in \mathbb{Z}$, and a would be even.

We have now justified the algorithm `IsPrime`, which tests primality. More precisely, `IsPrime` is the characteristic function of the set of primes in $\mathbb{Z}$.

```
Algorithm IsPrime
INPUT: n ∈ ℤ.
OUTPUT: TRUE if n is prime, FALSE if n is not prime.

a := |n|;   (* n is prime iff a is prime. We shall test if a is prime *)
if a < 2 then          (*   a is not prime   *)
  return FALSE;
fi;
if 2|a then
  return (a = 2);      (*   a is prime iff a = 2   *)
fi;
(*   at this point, a ≥ 3 and a is odd.   *)
i := 3;
while i^2 ≤ a do
   if i|a then
```

```
        return FALSE;  (*  a is not prime  *)
    fi
    i := i + 2;     (*  i is set to the next odd number  *)
od;
(*  at this point, odd a ≥ 3 has no odd prime divisor i > 1,
          such that i^2 ≤ a. It follows that a is prime  *)
return TRUE;
end;
```

Example 2.16. `IsPrime(107)`

a	$a<2$	$2\|a$	i	i^2	$i^2 \le a$	$i\|a$
107	F	F	3	9	T	F
			5	25	T	F
			7	49	T	F
			9	81	T	F
			11	121	F	

return TRUE, so 107 is prime.

Example 2.17. `IsPrime(-2468)`

a	$a<2$	$2\|a$	i	i^2	$i^2 \le a$	$i\|a$
2468	F	T				

return $(a = 2) =$ FALSE, so -2468 is not prime.

Example 2.18. `IsPrime(91)`

a	$a<2$	$2\|a$	i	i^2	$i^2 \le a$	$i\|a$
91	F	F	3	9	T	F
			5	25	T	F
			7	49	T	T

return FALSE, so 91 is not prime.

2.3 Factorization of integers

We begin with the *fundamental theorem of arithmetic*

Theorem 7 *Let $n \in \mathbb{Z}$, $n > 1$. Then n has a unique factorization of the form*

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_m^{a_m}$$

such that $p_1, \ldots, p_m$ are positive primes, $a_1, \ldots, a_m$ are positive integers, and $p_1 < p_2 < \cdots < p_m$.

Example 2.19.

$$719 = 719^1, \qquad 720 = 6! = 2^4 \cdot 3^2 \cdot 5^1, \qquad 721 = 7^1 \cdot 103^1.$$

The algorithmic problem is: given $n > 1$, find its unique factorization into primes. This is a very difficult problem, in general. Some cryptosystems are based on it.

We begin with some preliminaries. Let

$$A = (a_1, \ldots, a_k) \qquad B = (b_1, \ldots, b_l)$$

be finite sequences of length k and l, respectively. We define:

1. The *concatenation* $A \,\&\, B$ of A and B is a sequence of length $k + l$ given by

$$A \,\&\, B = (a_1, \ldots, a_k, b_1, \ldots, b_l). \tag{2.1}$$

The operator & is called the *concatenation operator*. If (a) is a one-element sequence, we write $A \,\&\, a$ for $A \,\&\, (a)$.

2. *Equality of sequences.* We say that $A = B$ if

$$k = l \quad \text{and} \quad a_1 = b_1,\ a_2 = b_2,\ \ldots\ , a_k = b_k. \tag{2.2}$$

Thus,

$$(0,0) \neq (0) \neq ((0)), \quad (3,4,4,-1) \neq (3,4,-1,-1), \quad (2,1) \neq (1,2), \quad (1,2,1) = (1,2,1).$$

Equality of *sets* has a quite different meaning

$$\{0,0\} = \{0\}, \qquad \{\{0\}\} \neq \{0\}, \qquad \{1,2\} = \{2,1\} = \{1,2,1\}.$$

3. *Length of a sequence.* We denote the cardinality of A by $\#A$. (We use this notation for sets as well as sequences.)

We now have all we need to develop the following

`Algorithm IntegerFactorization`
`INPUT`: n, an integer > 1.
`OUTPUT`: $(p_1, \ldots, p_k)$, such that $p_1, \ldots, p_k$ are positive primes,
$p_1 \leq p_2 \leq \ldots \leq p_k$, and $n = p_1 \cdot p_2 \cdots p_k$.

$P := (\,)$;
`while` $2|n$ `do`
  $n := n/2$;
  $P := P \,\&\, 2$;
`od`;
$i := 3$;

```
while i^2 ≤ n do
   while i|n do
        n := n/i;
        P := P & i;
  od;
   i := i + 2;
od;
if n > 1 then
      P := P & n;
fi;
return P;
end;
```

Example 2.20. `IntegerFactorization`(4018)

n	P	$2\|n$	i	$i^2 \le n$	$i\|n$	$n > 1$
4018	()	T				
2009	(2)	F	3	T	F	
			5	T	F	
			7	T	T	
287	(2,7)				T	
41	(2,7,7)				F	
			9	F		T
	(2,7,7,41)					

return $(2, 7, 7, 41)$.

Thus, $4018 = 2^1 \cdot 7^2 \cdot 41^1$ is the factorization of 4018 into primes.

2.4 Digits

Let n, b be integers, with $n \ge 0$ and $b > 1$. The *representation of n in base b* is given by

$$n = \sum_{k \ge 0} d_k \, b^k \qquad\qquad d_k \in \{0, 1, \ldots, b-1\}. \tag{2.3}$$

The coefficients d_k are the *digits* of n in base b. The sum (2.3) contains only finitely many nonzero terms, since each term is non negative.

Example 2.21. Digits of $n = 10^3$, for various bases b.

b	$(d_0, d_1, \dots)$
2	$(0,0,0,1,0,1,1,1,1,1)$
7	$(6,2,6,2)$
29	$(14,5,1)$
1001	(1000)

We develop an algorithm to construct the digits an integer in a given base. We define

$$n_l := \sum_{k\geq 0} d_{k+l}\, b^k \qquad l = 0, 1, \dots$$

giving

$$n_l = d_l + \sum_{k\geq 1} d_{k+l}\, b^k = d_l + b \sum_{k\geq 0} d_{k+l+1}\, b^k = d_l + b\, n_{l+1}. \tag{2.4}$$

Because, by construction, $0 \leq d_l < b$, we have that $d_l = n_l \,\mathrm{MOD}\, b$, and $n_{l+1} = n_l \,\mathrm{DIV}\, b$, and therefore n_{l+1} and d_l are *uniquely determined* by n_l and b (Theorem 3). We obtain the recursion relation

$$n_0 = n \qquad n_{l+1} = n_l \,\mathrm{DIV}\, b \qquad l \geq 0$$

which shows that the entire sequence (n_l) is uniquely determined by the initial condition $n_0 = n$, and by b. It then follows that the entire sequence of digits (d_l) is also uniquely determined by n and b.

It is plain that $n_{l+1} < n_l$, and since the n_l are non-negative integers, this sequence eventually reaches zero.

We have proved the uniqueness of the sequence of digits of n to base b, as well as the correctness of the following algorithm:

```
Algorithm Digits
INPUT: n, b, ∈ ℤ, n ≥ 0, b > 1.
OUTPUT: D, where D is the sequence of digits of n in base b,
            beginning from the least significant one.
if n = 0 then
    return 0;
fi;
D := ( );
while n > 0 do
    D := D & n MOD b;
    n := n DIV b;
od;
return D;
end;
```

Of note is the fact that no indices representing subscripts are needed.

From equation (2.3) we find

$$\begin{aligned} b\,n &= d_0 b + d_1 b^2 + d_2 b^3 \cdots \\ \frac{n - d_0}{b} &= d_1 + d_2 b + d_2 b^2 + \cdots \end{aligned}$$

So, multiplication and division by the base b correspond to *shifts* of digits

$$\begin{aligned} n &\longleftrightarrow (d_0, d_1, d_2, \ldots) \\ b\,n &\longleftrightarrow (0, d_0, d_1, \ldots) \\ \frac{n - d_0}{b} &\longleftrightarrow (d_1, d_2, d_3, \ldots). \end{aligned}$$

A much used base is $b = 2$, because it suits computers. The following algorithm performs the multiplication of two integers using only addition, as well as multiplication and division by 2 (see exercises).

```
Algorithm Mult
INPUT: m, n, with m, n ∈ ℤ, and n ≥ 0.
OUTPUT: l, such that l = mn.
l := 0;
while n > 0 do
  if (n MOD 2) = 0 then
    m := 2m;
    n := n/2;
  else
    l := l + m;
    n := n − 1;
  fi;
od;
return l;
end;
```

2.5 Nested algorithms

In the process of evaluating a function, we may have to evaluate another function, i.e., $\sin(x+\tan(x))$. Likewise, within `Algorithm1`, we may wish to execute `Algorithm2` (Figure 2.1). Within an expression in `Algorithm1`, the expression

`Algorithm2`(⟨ *expression 1* ⟩, ... ,⟨ *expression k* ⟩);

is executed as follows:

- The execution of `Algorithm1` is suspended.
- ⟨ *expression 1* ⟩, ... ,⟨ *expression k* ⟩ are evaluated to values $v_1, \ldots, v_k$, respectively.

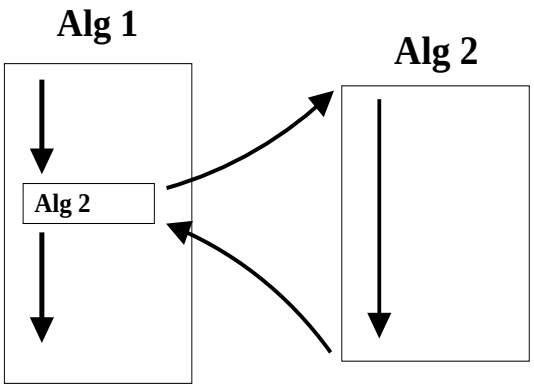

Figure 2.1: Nested algorithms: Algorithm 2 is executed within Algorithm 1.

- `Algorithm2` is executed with input sequence $(v_1, \ldots, v_k)$.
- In `Algorithm1,` the value of `Algorithm2`($\langle$ *expression 1* $\rangle, \ldots, \langle$ *expression k* $\rangle$) becomes the output sequence of this execution of `Algorithm2.`
- The execution of `Algorithm1` continues.

In the rest of this section, we consider a common construct that requires nested algorithms: counting the number of elements of a subset of $\mathbb{Z}$, that lie in a given interval.

2.5.1 Counting subsets of the integers

Let A be a subset of $\mathbb{Z}$, and let $[a, b]$ $a, b, \in \mathbb{Z}$, be a closed interval. We with to count the elements of A that belong to $[a, b]$. To do so, we assume that the characteristic function of A in $\mathbb{Z}$ is available

`Algorithm ChiA`
`INPUT`: $x \in \mathbb{Z}$
`OUTPUT`: TRUE, if $x \in A$, FALSE otherwise.

The structure of the counting algorithm is straightforward

`Algorithm CountA`
`INPUT`: $a, b \in \mathbb{Z}$, $a \leq b$.
`OUTPUT`; n, where $n = \#\{x \in A \,|\, a \leq x \leq b\}$.

$n := 0$; (* initialize counter *)
$x := a$; (* initialize position *)
`while` $x \leq b$ `do`
  `if ChiA`(x) `then`

```
        n := n + 1;          (* increase counter *)
    fi;
    x := x + 1;              (* increase position *)
od;
return n;
end;
```

Example 2.22. A prime p such that $p+2$ is also prime, is called called a *twin prime*. The sequence of twin primes is conjectured to be infinite

$$3, 5, 11, 17, 29, 41, \ldots$$

although no proof of this conjecture is know.

The characteristic function of the set of twin primes is easily constructed

```
Algorithm IsTwinPrime
INPUT: x ∈ ℕ
OUTPUT: TRUE, if x is a twin prime, FALSE otherwise.
   return  IsPrime(x) AND IsPrime(x + 2);
end;
```

To count twin primes in an interval, it suffices to replace `CharA` with `IsTwinPrime` in the algorithm `CountA`.

2.6 The halting problem*

We recall that an algorithm is said to be *correct,* if it terminates in a finite time for all valid input, giving the correct output. We now provide a simple but telling example where defining what constitutes *valid input,* is essentially impossible; for even though the problem is simply formulated, one cannot be certain that the algorithm will terminate given arbitrary input. This is the celebrated *halting problem* of the theory of algorithms.

Let us consider the so-called '$3x+1$' function

$$T : \mathbb{N} \mapsto \mathbb{N} \qquad T(x) = \begin{cases} x/2 & x \text{ even} \\ 3x+1 & x \text{ odd} \end{cases}$$

which is easily implemented.

```
Algorithm T
INPUT: x ∈ ℕ.
OUTPUT; y, where y is the image of x under the '3x + 1' function.
```

```
if (x MOD 2) = 0 then
   return x/2;
else
   return 3x + 1;
fi;
end;
```

Since domain and image of T coincide, we can *iterate* this function; we choose an arbitrary *initial condition* $x \in \mathbb{N}$, compute $T(x)$ to obtain a new point in $\mathbb{N}$, then apply T to this point, and so on. If we choose $x = 1$, we obtain

$$1 \mapsto 4 \mapsto 2 \mapsto 1 \mapsto 4 \mapsto \cdots$$

a *periodic integer sequence.* Let us call the cycle (4,2,1) a *limit cycle* for T. The '$3x + 1$' conjecture says that any initial condition will eventually lead to that cycle, i.e.,

$$9,\ 28,\ 14,\ 7,\ 22,\ 11,\ 34,\ 17,\ 52,\ 26,\ 13,\ 40,\ 20,\ 10,\ 5,\ 16,\ 8,\ \mathbf{4}$$

and it is easy to persuade oneself that one can access such cycle only through the point $x = 4$. Proving this conjecture seems beyond the reach of modern mathematics.

The craziness of this phenomenon becomes apparent when we construct an *directed graph,* called the *Collatz graph,* whose vertices are the natural integers, and where there is a edge joining x to y if $y = T(x)$. The '$3x+1$' conjecture can now be phrased by saying that the Collatz graph is connected, and has only one cycle. Alternatively, removing the vertices 1 and 2, and the related edges, one obtains a *tree,* rooted at 4.

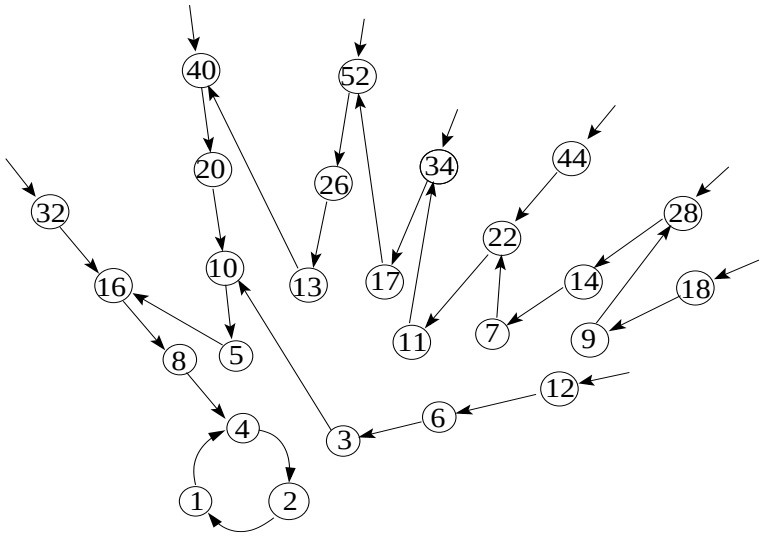

Figure 2.2: The Collatz graph, in a neighbourhood of the limit cycle $(4, 2, 1)$.

We thus introduce the *transit function* $\mathcal{T}$ as the time it takes to get to the limit cycle (4,2,1)

$$\mathcal{T}(8) = 1 \qquad\qquad \mathcal{T}(9) = 17.$$

The '$3x + 1$' conjecture states that $\mathcal{T}$ is well-defined, that is, that $\mathcal{T}(x) < \infty$, for all positive integers x.

```
Algorithm TransitTime
INPUT: x ∈ ℕ
OUTPUT: t, where t is the transit time from x to the (1,4,2)-cycle of T.
if x < 3 then
   return 0;
fi;
t := 0:
while x ≠ 4 do
   t := t + 1;
   x := T(x);
od;
return t;
end;
```

Plainly, for some value of the input, this algorithm may not halt. However, no evidence of such phenomenon has ever been found, so these integers, if they exist, are necessarily very large.

Exercises

Exercise 2.1. List all divisors of $n = 120$. List all primes p such that $120 < p < 140$.

Exercise 2.2.

(a) Let a, b, c, x, y be integers, such that a divides b and a divides c. Show that a divides $xb + yc$.

(b) Let n be an integer. Show that if $-n$ is not prime, then n is not prime. Use this to prove that n is prime if and only if $-n$ is prime.

(c) Show that, if n is odd, then $(n^2 \text{ MOD } 8) = 1$.

(d) Using the above, show that $x^2 + y^2 = z^2$ cannot be true in integers, when both x and y are odd. Give an example.

Exercise 2.3. Using the operators DIV and MOD, write an algorithm to the following specifications:

```
Algorithm Nint
INPUT: a, b ∈ ℤ, b ≠ 0.
OUTPUT: c, where c is an integer nearest to a/b.
```

(Note: *an* integer, not *the* integer, so you may have a choice. Begin with the case $a, b > 0$.)

Exercise 2.4. Apply the algorithm `IsPrime` to each of the integers 433, 437, to determine which of these integers is prime.

Exercise 2.5. Apply the algorithm `IntegerFactorization` to each of the integers 127, 216, 4018

to determine their factorization into primes.

Exercise 2.6. How many divisions will `IsPrime` have to perform to verify that

$$p = 2^{127} - 1 = 170141183460469231731687303715884105727$$

is prime? (Lucas showed that p is prime in 1876). On the fastest computer on earth ($\sim 10^{12}$ divisions per second), how many years will it take to complete the calculation?

The largest known prime is the prime on the left in (1.1). On the fastest computer on earth, and assuming that the lifetime of the universe is 20 billion years, how many lifetimes of universes will it take to test its primality with `IsPrime`?

Exercise 2.7. Consider the following algorithm

`INPUT` $a, b \in \mathbb{Z}$, $a \geq 0, b > 0$.
`OUTPUT` ??
`while` $a \geq 0$ `do`
  $a := a - b$;
`od`
`return` $a + b$;
`end`;

Write the output specifications of this algorithm, and explain how it works, keeping the use of mathematical symbols to a minimum. What happens if the constraints $a \geq 0$, $b > 0$ are removed from the input?

Exercise 2.8. Write an algorithm to the following specifications:

`Algorithm NumMul`
`INPUT`: $a, b, x \in \mathbb{Z}$, $x > 0$, $0 < a < b$.
`OUTPUT`: n, where n is the number of multiples of x
which are greater than a and smaller than b.

Try to make the computation efficient.

Exercise 2.9. Write an algorithm to the following specifications:

`Algorithm Test`
`INPUT`: $x, a, b \in \mathbb{Z}$, $a, b \neq 0$.
`OUTPUT`: TRUE if x is divisible by a or by b, but not by both,
and FALSE otherwise.

Exercise 2.10. An integer is *square-free* if it is not divisible by any square greater than 1.

(a) Find all square-free integers in the interval [40, 60].

(b) Consider the following algorithm

`Algorithm SquareFree`

`INPUT`: n, a positive integer.

`OUTPUT`: TRUE if n is square-free, FALSE otherwise.

Explain how to use `IntegerFactorization` to construct `SquareFree`. [$\not\in$, 50]

(c) Write the algorithm `SquareFree`, using `IntegerFactorization`.

Exercise 2.11. A *Sophie Germain (SG) prime* is an odd prime p such that $2p+1$ is also prime.

(a) There are 6 SG-primes smaller than 50: find them. Can you find all SG-primes between 50 and 100?

(b) Using `IsPrime`, write an algorithm to the following specifications

`Algorithm SGprime`

`INPUT`: $x \in \mathbb{N}$.

`OUTPUT`: TRUE if x is a SG-prime, and FALSE otherwise.

Try to make it efficient (think of the calculation in part (a)).

(c) Write an algorithm to the following specifications

`Algorithm NumberSGprimes`

`INPUT`: $a, b \in \mathbb{N}$, $a < b$.

`OUTPUT`: n, where n is the number of SG-primes in the closed interval $[a, b]$.

Exercise 2.12. Consider the following algorithm

`Algorithm Square`

`INPUT`: n, a positive integer.

`OUTPUT`: TRUE, if n is a square, FALSE otherwise.

(a) Explain how the algorithm `IntegerFactorization` can be used to construct `Square`. [$\not\in$, 50]

(b) Write the algorithm `Square`, using `IntegerFactorization`.

Exercise 2.13. Let n be a positive integer, and recall that

$$n! = n \cdot (n-1) \cdots 2 \cdot 1.$$

Show that there is no prime p such that $n! + 2 \leq p \leq n! + n$. (Thus, there are arbitrarily large gaps between consecutive primes.)

Exercise 2.14. Find the smallest integer x for which `NextPrime`$(x) - x = 8$. (see Section 1.1.5). The previous problem guarantees that such x does not exceed 10!, which is a very poor upper bound.

Exercise 2.15. Consider the algorithm `Mult` (Section 2.4).

(a) Trace it with the following input (m, n):

$$(7, 6), \qquad (12, 18), \qquad (-13, 4).$$

In each case, use a table to show how the values of m, n and l change as `Mult` is executed, and indicate what is returned as output.

(b) Prove its correctness.

Chapter 3

Relations and partitions

We introduce a basic concept of higher mathematics: an equivalence relation. Our main application will be modular arithmetic, in Chapter 4.

3.1 Relations

The cartesian product of two sets X and Y is defined as

$$X \times Y = \{(x, y) \mid x \in X,\, y \in Y\}.$$

If $X = Y$, we often write X^2 for $X \times X$.

Example 3.23. The cartesian product $\mathbb{R}^2$ is called the *cartesian plane.* The domain of the algorithm `SumOfSquares` of Example 11 is the set $\mathbb{Z}^2$.

Def: A *relation* from X to Y is a subset of $X \times Y$. A relation from X to X is called a *relation on* X.

If R is a relation from X to Y we write xRy to mean $(x, y) \in R$ (think of this as "x is related by R to y"). The expression xRy is therefore boolean. Note that the symbol R is used here to represent two different objects: a set (as in $R \subseteq X^2$) and a *relational operator* (as in xRy). The meaning of the notation will be clear from the context, and will not lead to ambiguity.

Example 3.24. Let

$$X = \{1, 2, 3\} \qquad R = \{(1, 2), (1, 3), (2, 2), (2, 3), (3, 1)\}.$$

Then R is a relation on X. We have that $2R\,3$ is TRUE but $3R\,2$ is FALSE.

The relation R on X can be represented as a *directed graph,* whose vertices are the elements of X, and where an arc joins x to y if xRy (Figure 3.1).

Def: Let R be a relation on X.

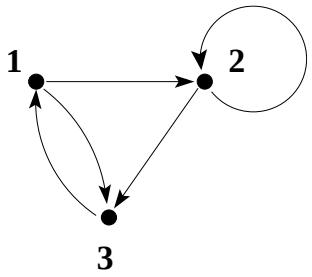

Figure 3.1: Directed graph of the relation of Example 3.24.

- R is *reflexive* if for all $x \in X$, xRx.
- R is *symmetric* if for all $x, y \in X$, $xRy \implies yRx$.
- R is *transitive* if for all $x, y, z \in X$, $(xRy \text{ AND } yRz) \implies xRz$.
- R is an *equivalence relation* if R is reflexive, symmetric and transitive.

Checking the above properties amounts to evaluating boolean expressions. Specifically, we define the following functions

$$\begin{array}{lll} \rho: & X \to \{\text{TRUE}, \text{FALSE}\} & x \mapsto xRx \\ \sigma: & X^2 \to \{\text{TRUE}, \text{FALSE}\} & (x,y) \mapsto xRy \implies yRx \\ \tau: & X^3 \to \{\text{TRUE}, \text{FALSE}\} & (x,y,z) \mapsto (xRy \text{ AND } yRz) \implies xRz \end{array}$$

Thus, for instance, R is symmetric precisely when the function σ assume the value TRUE everywhere on X^2.

We begin with the relations on X corresponding to the trivial subsets of X^2, namely the empty set and X^2. The relation $R = \{\}$ is the empty relation on X. For every $x, y, z \in X$, we have $x\{\}y = y\{\}z = x\{\}z = \text{FALSE}$, and therefore the boolean expression $(x\{\}y \text{ AND } y\{\}x) \implies x\{\}z$ evaluates to TRUE. We conclude that $\{\}$ is transitive. By a similar argument one shows that $\{\}$ is symmetric. (Is $\{\}$ reflexive?) Let now $R = X^2$. Because for all $x, y \in X$ the expression xX^2y evaluates to TRUE, so do the boolean expressions defining reflexivity, symmetry and transitivity. We conclude that X^2 is an equivalence relation on X. The corresponding graph is a complete graph.

Example 3.25. Consider the following relations R on $\mathbb{Z}$

	xRy	reflexive?	symmetric?	transitive?	equivalence?
(i)	$x = y$	yes	yes	yes	yes
(ii)	$x \leq y$	yes	no	yes	no
(iii)	$x \mid y$	yes	no	yes	no
(iv)	$x + y = 6$	no	yes	no	no
(v)	$2 \mid x - y$	yes	yes	yes	yes

(ii) is not symmetric: $3 \leq 4$ but $4 \not\leq 3$.

(iii) is transitive. Suppose $x|y$ and $y|z$. Then $y = xa$ for some $a \in \mathbb{Z}$ and $z = yb$ for some $b \in \mathbb{Z}$, so that $z = xab \Longrightarrow x|z$.

(iv) is not reflexive ($1+1 \neq 6$), and not transitive ($1+5 = 6$ and $5+1 = 6$, but $1+1 \neq 6$).

(v) is a special case of a more general construct, with is dealt with in the following

Theorem 8 *Let m be an integer, and define the relation $\equiv_m$ on $\mathbb{Z}$ by $x \equiv_m y$ if $m|x-y$. Then $\equiv_m$ is an equivalence relation.*

Proof:

(i) Let $x \in \mathbb{Z}$. Then

$$m|0 \implies m|x-x \implies x \equiv_m x$$

i.e., $\equiv_m$ is reflexive.

(ii) Let $x, y \in \mathbb{Z}$, and suppose $x \equiv_m y$. Then

$$m|x-y \implies m|-(x-y) \implies m|y-x \implies y \equiv_m z$$

so $z \equiv_m$ is symmetric.

(iii) Let $x, y, z \in \mathbb{Z}$, and suppose that $x \equiv_m y$ and $y \equiv_m x$. Then

$$\begin{aligned} &\implies m|x-y \quad \text{and} \quad m|y-z \\ &\implies m|(x-y)+(y-z) \\ &\implies m|x-z \\ &\implies x \equiv_m z, \end{aligned}$$

i.e., $\equiv_m$ is transitive. Because $\equiv_m$ is reflexive, symmetric and transitive, $\equiv_m$ is an equivalence. ■

3.2 Partitions

Def: A *partition* $\mathcal{P}$ of a set X is a set of non-empty subsets of X, such that each element of X is in exactly one element of $\mathcal{P}$ (Figure 3.2).

The elements of a partition are called *parts*. If $\mathcal{P}$ is a partition of X and $x \in X$, we denote by $P(x)$ the unique part in $\mathcal{P}$ containing x. Note that, for all $x, y \in X$, either $P(x) = P(y)$ or $P(x) \cap P(y) = \{\}$; furthermore $\mathcal{P} = \{P(x) \,|\, x \in X\}$.

Example 3.26. Let $X = \{1, 2, 3\}$, $\mathcal{P} = \{\{1, 3\}, \{2\}\}$. Then $\mathcal{P}$ is a partition of X with parts $\{1, 3\}$ and $\{2\}$:

$$P(1) = \{1, 3\} = P(3) \qquad P(2) = \{2\}.$$

Def: Let $\mathcal{P}$ be a partition on X. We define the relation $R_{\mathcal{P}}$ on X by $x \, R_{\mathcal{P}} y$ if $P(x) = P(y)$.

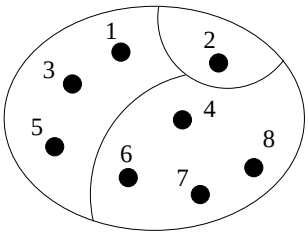

Figure 3.2: Partition of the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$ into 3 parts. We have $P(1) = P(5) = \{1, 3, 5\}$.

Example 3.27. Let $X = \{1, 2, 3\},\ \mathcal{P} = \{\{1, 3\}, \{2\}\}$. Then

$$R_{\mathcal{P}} = \{(1, 1),\ (1, 3),\ (3, 1),\ (3, 3),\ (2, 2)\}.$$

(Draw $R_{\mathcal{P}}$ as a directed graph.) One verifies that $R_{\mathcal{P}}$ is an equivalence relation. This is always the case, as shown by the following

Theorem 9 *If $\mathcal{P}$ is a partition of X, then $R_{\mathcal{P}}$ is an equivalence relation on X.*

Proof:

(*i*) $R_{\mathcal{P}}$ is reflexive because, for all $x \in X$, $P(x) = P(x)$.

(*ii*) $R_{\mathcal{P}}$ is symmetric because, for all $x, y \in X$, $P(x) = P(y) \implies P(y) = P(x)$.

(*iii*) $R_{\mathcal{P}}$ is transitive because, for all $x, y, z \in X$, $P(x) = P(y)$ and $P(y) = P(z) \implies P(x) = P(z)$. ■

Example 3.28. Let

$$X = \{a, b, c, d, e, f, g, h, i, j\} \qquad \mathcal{P} = \{\{a, b, c, d\},\ \{e, f, g\},\ \{h, i\},\ \{j\}\}.$$

The equivalence $R_{\mathcal{P}}$ is displayed in Figure 3.3 as the union of *complete graphs*.

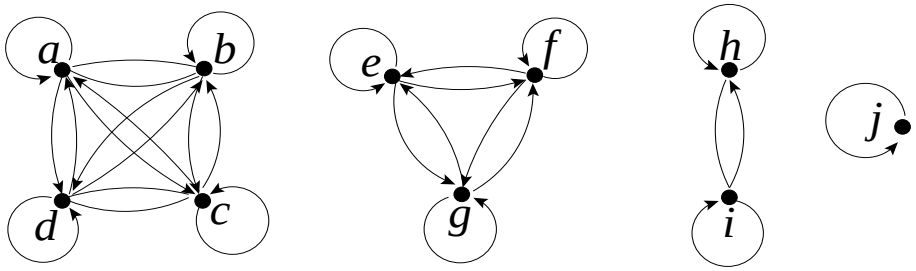

Figure 3.3: Directed graph of the equivalence generated by the partition of Example 3.28.

Example 3.29. Suppose E is the equivalence relation on $X = \{1, 2, 3, 4, 5\}$ given by

$$E = \{(1,1), (1,3), (2,2), (2,4), (3,1), (3,3), (4,2), (4,4), (5,5)\}.$$

To construct a partition from E, we proceed as follows. We choose an arbitrary point in X, $x = 1$, say, and we let $P(1) := \{y \in X \,|\, 1Ey\}$. We find $P(1) = \{1, 3\}$. Of the remaining points in $X \setminus P(1)$, we choose $x = 2$ and repeat the construction, to obtain $P(2) := \{y \in X \,|\, 2Ey\} = \{2, 4\}$. There is only one point left: $P(5) = \{y \in X \,|\, 5Ey\} = \{5\}$. Now, $P(1)$, $P(2)$ and $P(5)$ are non empty, disjoint (because E is an equivalence relation, think about it), and their union is X. Thus we have obtained a partition of X: $\mathcal{P} = \{P(1), P(2), P(3)\}$. One verifies that $E = R_{\mathcal{P}}$, from the definition of $R_{\mathcal{P}}$.

The above construction works in general. Suppose we know an equivalence relation E on X, and we want to determine a partition $\mathcal{P}$ of X such that $E = R_{\mathcal{P}}$. We construct $\mathcal{P}$ as follows. Let $x \in X$. Then $P(x)$ —the part containing x— is equal to

$$\{y \in X \,|\, P(x) = P(y)\} = \{y \in X \,|\, x\, E\, y\}.$$

Knowing each part of $\mathcal{P}$, we then know $\mathcal{P}$, since $\mathcal{P} = \{P(x) \,|\, x \in X\}$. In the last formula, the requirement $x \in X$ may be highly redundant; to construct $\mathcal{P}$ algorithmically, one would instead identify recursively a representative x from each part of $\mathcal{P}$, as we did in the previous example.

Do all equivalence relations originate from partitions? In other words, if we start with an arbitrary equivalence relation E on X, is it true that $E = R_{\mathcal{P}}$ for some partition $\mathcal{P}$ of X? The answer to this question is affirmative:

Theorem 10 *Let E be an equivalence relation on a set X, and for each $x \in X$, define $E(x)$ to be the set $\{y \in X \,|\, x\, E\, y\}$. Let $\mathcal{P} = \{E(x) \,|\, x \in X\}$. Then (i) $\mathcal{P}$ is a partition of X, and (ii) $E = R_{\mathcal{P}}$.*

Proof: (*i*) Each element $E(x)$ of $\mathcal{P}$ is a subset of X. Furthermore, $x \in E(x)$ (because E is reflexive), so $E(x) \neq \{\}$, and each element $x \in X$ is in some element of $\mathcal{P}$. We now only need to show that no element of X can be in more than one element of $\mathcal{P}$. To do his we suppose $x \in X$, $x \in E(y)$ and $x \in E(z)$, and then show we must have $E(y) = E(z)$.

We first show that $E(y) \subseteq E(z)$. Let $t \in E(y)$. Then we have

$$\begin{array}{lcll} yEx \text{ AND } zEx \text{ AND } yEt & \Longrightarrow & zEx \text{ AND } xEy \text{ AND } yEt & (E \textit{ symmetric}) \\ & \Longrightarrow & zEy \text{ AND } yEt & (E \textit{ transitive}) \\ & \Longrightarrow & zEt & (E \textit{ transitive}) \\ & \Longrightarrow & t \in E(z). & \end{array}$$

Thus, $t \in E(y) \Longrightarrow t \in E(z)$, so

$$E(y) \subseteq E(z). \tag{3.1}$$

Interchanging the role of y and z in the above argument, we obtain $E(z) \subseteq E(y)$, which, together with (3.1), implies that $E(y) = E(z)$. This concludes the proof that $\mathcal{P}$ is a partition of X.

(*ii*) We have shown that $\mathcal{P} = \{E(x) \,|\, x \in X\}$ is a partition of X. Since $x \in E(x)$, we have that $E(x) = P(x)$, the unique part of $\mathcal{P}$ containing x.

Let $x, y \in X$. Then

$$\begin{array}{lcll} xEy & \iff & y \in E(x) & (\textit{by def. of } E(x)) \\ & \iff & y \in E(x) \quad \text{AND} \quad y \in E(y) & (\textit{since } y \in E(y)) \\ & \iff & y \in E(x) = E(y) & (\textit{since } \mathcal{P} \textit{ is a partition}) \\ & \iff & P(x) = P(y) & \\ & \iff & xR_{\mathcal{P}}y. & \end{array}$$

Thus, $E = R_{\mathcal{P}}$. ■

Def: Let E be an equivalence relation on X, and $x \in X$. Then $E(x) = \{y \in X \,|\, xEy\}$ is called the *equivalence class (of E) containing x*, and the partition $\{E(x) \,|\, x \in X\}$ is denoted by X/E.

Note that we have proved that $E = R_{X/E}$.

Let θ be the map from the set of equivalence relations on X to the set of partitions of X, defined by

$$\theta(E) = X/E$$

for each equivalence relation E on X.

Theorem 11 (*i*) *The map θ is a bijection (one-to-one and onto);* (*ii*) *If $\mathcal{P}$ is a partition of X, then $\theta^{-1}(\mathcal{P}) = R_{\mathcal{P}}$.*

Proof: (*i*) We first prove that θ is one-to-one. Suppose E_1 and E_2 are equivalence relations on X, and $\theta(E_1) = \theta(E_2)$. This means

$$X/E_1 = X/E_2 \quad \Longrightarrow \quad E_1 = R_{X/E_1} = R_{X/E_2} = E_2$$

(using previous theorem, part (*ii*)).

We now prove that θ is onto. Let $\mathcal{P}$ be a partition of X. We have that $R_{\mathcal{P}} = R_{X/R_{\mathcal{P}}}$ (by the previous theorem, part (*ii*)), hence $\mathcal{P} = X/R_{\mathcal{P}}$ (since $\mathcal{P}$ is completely determined by $R_{\mathcal{P}}$), and therefore $\mathcal{P} = \theta(R_{\mathcal{P}})$.

(*ii*) θ is a bijection, so θ is invertible. The proof that θ is onto shows that $\theta(R_{\mathcal{P}}) = \mathcal{P}$ for each partition $\mathcal{P}$ of X. Therefore $\theta^{-1}(\mathcal{P}) = R_{\mathcal{P}}$. ■

Schematically, the content of the last theorem is the following

$$\begin{array}{ccc} \text{equivalences on } X & & \text{partitions of } X \\ \hline E & \xrightarrow{\theta} & X/E \\ R_{\mathcal{P}} & \xleftarrow{\theta^{-1}} & \mathcal{P} \end{array}$$

Exercises

Exercise 3.1. Let $X = \{a, b, c\}$.

(a) Determine all partitions of X.

(b) Determine all equivalence relations on X (as a subset of $X \times X$).

Exercise 3.2. Let $X = \{1, 2, 3, 4\}$.

(a) Determine all the partitions $\mathcal{P}$ of X such that $\mathcal{P}$ has exactly two parts.

(b) For each such partition $\mathcal{P}$, write down the corresponding equivalence relation $R_{\mathcal{P}}$.

Exercise 3.3. Let $X = \{1, 2, 3\}$. Determine relations R_1, R_2, R_3 on X, such that

(a) R_1 is symmetric and transitive, but not reflexive.

(b) R_2 is reflexive and transitive, but not symmetric.

(c) R_3 is reflexive and symmetric, but not transitive.

In each case, try to make relations have as few elements as possible.

Exercise 3.4. Let $X = \{1, 2, 3, 4, 5, 6, 7\}$, and let

$$R = \{(1, 7),\ (1, 4),\ (3, 1),\ (4, 3),\ (6, 2)\}$$

be a relation on X. Suppose E is an equivalence relation on X such that $R \subseteq E$ and E has as few elements as possible.

(a) Determine the partition X/E corresponding to this equivalence.

(b) How many elements does E have?

Exercise 3.5. Determine the possible cardinalities of an equivalence relation on a set of 5 elements.

Exercise 3.6. Let R be an equivalence relation on a finite set X. Prove that $\#R$ has the same parity as $\#X$.

Exercise 3.7. Let $\mathcal{P}$ be a partition of a finite set X, and let $P(x)$ be the part of $\mathcal{P}$ containing x.

(a) Explain why the formula $\mathcal{P} = \{P(x) : x \in X\}$ does not translate into an efficient algorithm for constructing $\mathcal{P}$ from the knowledge of X and $\mathcal{P}$.

(b) Write an algorithm to the following specifications

`Algorithm Partition`

`INPUT`: X, P

`OUTPUT`: $\mathcal{P}$

You may use set operators (union, etc.), and represent the elements of a set using subscripts $A = \{A_1, A_2, \ldots\}$. Be careful that $\mathcal{P}$ is a *set of sets.*

Exercise 3.8. Let f be the characteristic function of a relation $R \subseteq X^2$, where $X = \{1, 2, \ldots, n\}$, $n \geq 1$. Write an algorithm to the following spefifications

`Algorithm IsSymm`

`INPUT`: n, P

`OUTPUT`: TRUE, if R is symmetric, FALSE otherwise.

Chapter 4

Modular arithmetic

The sum of two odd integers is even, their product is odd. Modular arithmetic affords a vast generalization of statements of this type, by defining arithmetical operations between infinite families of integers, of which the even and the odd integers are a special example.

Let $m \in \mathbb{Z}$. Recall that the relation $\equiv_m$ on $\mathbb{Z}$, defined by $i \equiv_m j$ if $m|i-j$ is an equivalence relation on $\mathbb{Z}$ (Theorem 8). Such relation is called a *congruence,* and the corresponding equivalence classes are called *congruence classes* or *residue classes.*

Let $i, j \in \mathbb{Z}$. Then

$$\begin{aligned} i \equiv_m j &\iff j \equiv_m i \\ &\iff m|j-i \\ &\iff j - i = mk \quad \text{for some} \quad k \in \mathbb{Z} \\ &\iff j = i + mk \quad \text{for some} \quad k \in \mathbb{Z}. \end{aligned}$$

Thus, the equivalence class containing i is

$$\{j \in \mathbb{Z} \,|\, i \equiv_m j\} = \{i + m\,k \,|\, k \in \mathbb{Z}\}.$$

We denote the equivalence class containing i by $[i]_m$.

Example 4.30. The following is readily verified from the definition

$$[3]_5 = \{3 + 5\,k \,|\, k \in \mathbb{Z}\} = \{\ldots, -12, -7, -2, 3, 8, 13, \ldots\} = [-12]_5.$$

Remarks:

(*i*) $[i]_0 = \{i + 0\,k \,|\, k \in \mathbb{Z}\} = \{i\}$

(*ii*) $[i]_1 = \{i + 1\,k \,|\, k \in \mathbb{Z}\} = \mathbb{Z}$

(*iii*) $[i]_{-m} = \{i + (-m)k \,|\, k \in \mathbb{Z}\} = \{i + m\,k \,|\, k \in \mathbb{Z}\} = [i]_m$.

Equality (i) says that the relations $=$ and $\equiv_0$ are the same on $\mathbb{Z}$, so the case $m = 0$ is trivial. Equality (iii) says that considering negative moduli is superfluous. So in the rest of this chapter we assume $m > 0$. Then Theorem 3 tells us that there are unique integers q, r such that $i = qm + r$ and $0 \leq r < m$. (Recall that such q and r are denoted by $i \,\mathrm{DIV}\, m$ and $i \,\mathrm{MOD}\, m$, respectively.) In particular, we see that $i \equiv_m (i \,\mathrm{MOD}\, m)$ so $[i]_m = [i \,\mathrm{MOD}\, m]_m$.

The partition $\mathbb{Z}/\equiv_m$, corresponding to the equivalence relation $\equiv_m$ is usually denoted by $\mathbb{Z}/(m)$, and called the set of *integers modulo* m. Thus

$$\begin{aligned} \mathbb{Z}/(m) &= \mathbb{Z}/\equiv_m = \{[i]_m \,|\, i \in \mathbb{Z}\} \\ &= \{[i \,\mathrm{MOD}\, m]_m \,|\, i \in \mathbb{Z}\} \\ &= \{[0]_m, [1]_m, \ldots, [m-1]_m\}. \end{aligned} \tag{4.1}$$

Now suppose that $0 \leq i, j, < m$, and $[i]_m = [j]_m$. Then $i = k\,m + j$ for some $k \in \mathbb{Z}$, hence $i \,\mathrm{MOD}\, m = j$ (since $0 \leq j < m$). But $i \,\mathrm{MOD}\, m = i$, so we must have $i = j$. This tells us that $[0]_m, [1]_m, \ldots, [m-1]_m$ are distinct, so

$$\mathbb{Z}/(m) = ([0]_m, [1]_m, \ldots, [m-1]_m)$$

has size m. The integers $0, 1, \ldots, m-1$ are a common choice for representatives of equivalence classes, and are called the *least non-negative residues* modulo m.

The notation $x \equiv_m y$ is shorthand for Gauss' notation

$$x \equiv y \,(\mathrm{mod}\ m).$$

The symbol MOD for remainder of division, as well as the term *modular arithmetic* derive from it.

4.1 Addition and multiplication in $\mathbb{Z}/(m)$

Theorem 12 *Let $m, a, b, c, d \in \mathbb{Z}$, such that $a \equiv_m c$ and $b \equiv_m d$. Then*

$$(i) \quad a + b \equiv_m c + d \qquad\qquad (ii) \quad ab \equiv_m cd.$$

So congruences can be added and multiplied together: in this respect they behave like equations.

Proof: $a \equiv_m c$ means $c = a + km$ for some $k \in Z$, and $b \equiv_m d$ means $d = b + lm$ for some $l \in Z$.

(i) We have

$$\begin{aligned} c + d &= a + km + b + lm \\ &= (a + b) + (k + l)m \equiv_m a + b. \end{aligned}$$

(ii) We have

$$\begin{aligned} cd &= (a+km)(b+lm) \\ &= ab + alm + kmb + kmlm \\ &= ab + (al + kb + klm)m \equiv_m ab \end{aligned} \qquad \blacksquare$$

Now, define addition and multiplication in $\mathbb{Z}/(m)$ by

$$[a]_m + [b]_m = [a+b]_m \qquad\qquad [a]_m[b]_m = [ab]_m.$$

The above theorem implies that these operations are well-defined, in the sense that the result does not depend on our choice of representatives for equivalence classes.

Example 4.31. Check the following equalities carefully.

$$[3]_5 + [2]_5 = [5]_5 = [0]_5 \qquad [3]_5[2]_5 = [6]_5 = [1]_5$$
$$[3]_6 + [2]_6 = [5]_6 \qquad [3]_6[2]_6 = [6]_6 = [0]_6$$

Theorem 13 *For all $a, b, c \in \mathbb{Z}/(m)$ the following holds*

(i) $a + b \in \mathbb{Z}/(m)$, $ab \in \mathbb{Z}/(m)$

(ii) $(a + b) + c = a + (b + c)$, $(ab)c = c(bc)$

(iii) $a + b = b + a$, $ab = ba$

(iv) $a + [0]_m = a$, $a[1]_m = a$

(v) *there is an element* $-a \in \mathbb{Z}/(m)$ *such that* $a + (-a) = [0]_m$

(vi) $a(b + c) = ab + ac$

Proof: (i) follows from definitions of addition and multiplication. The properties (ii), (iii) and (vi) are inherited from $\mathbb{Z}$. For example, we prove (vi) in detail. Suppose $a = [i]_m$, $b = [j]_m$, $c = [k]_m$. Then

$$\begin{aligned} a(b+c) &= [i]_m([j]_m + [k]_m) \\ &= [i]_m[j+k]_m \\ &= [i(j+k)]_m \\ &= [ij + ik]_m \\ &= [ij]_m + [ik]_m \\ &= [i]_m[j]_m + [i]_m[k]_m \\ &= ab + ac. \end{aligned}$$

The proofs of (ii) and (iii) are similar (try them!).

(*iv*) Suppose $a = [i]_m$. Then

$$\begin{aligned} a + [0]_m &= [i]_m + [0]_m = [i+0]_m = [i]_m = a \\ a[1]_m &= [i]_m[1]_m = [i\,1]_m = [i]_m = a. \end{aligned}$$

(*v*) Suppose $a = [i]_m$, and define $-a = [-i]_m$. (This is well-defined, verify it.) Then

$$a + (-a) = [i]_m + [-i]_m = [i + (-i)]_m = [0]_m.$$

This completes the proof. ■

4.2 Invertible elements in $\mathbb{Z}/(m)$

Def: Let $a \in \mathbb{Z}/(m)$. We say that a is *invertible* if there exists an element $b \in \mathbb{Z}/(m)$ such that $ab = [1]_m$.

Proposition 14 *Suppose $a, b, c \in \mathbb{Z}/(m)$, a is invertible, and $ab = ac = [1]_m$. Then $b = c$.*

Proof: We find

$$\begin{aligned} ab = ac &\implies bab = bac \\ &\implies abb = abc \\ &\implies [1]_m b = [1]_m c \\ &\implies b = c. \qquad \blacksquare \end{aligned}$$

Suppose a is an invertible element of $\mathbb{Z}/(m)$. The above proposition says there is a *unique* $b \in \mathbb{Z}/(m)$ such that $ab = [1]_m$. We call this unique b the *(multiplicative) inverse of a,* which is denoted by a^{-1}. (Similarly, the additive inverse of an element of $\mathbb{Z}/(m)$ is unique.)

Let $a, b \in \mathbb{Z}/(m)$. The notation $a - b$ means $a + (-b)$, and if b is invertible a/b means ab^{-1}.

Example 4.32. In $\mathbb{Z}/(4)$ we have

a	$[0]_4$	$[1]_4$	$[2]_4$	$[3]_4$
$-a$	$[0]_4$	$[3]_4$	$[2]_4$	$[1]_4$
a^{-1}	$-$	$[1]_4$	$-$	$[3]_4$

Also

$$\frac{[2]_4 - [1]_4}{[3]_4} = ([2]_4 - [1]_4)\,[3]_4^{-1} = [1]_4[3]_4 = [3]_4.$$

When we are working in $\mathbb{Z}/(m)$ and there is no risk of confusion, we will use i to denote $[i]_m$. Thus in $\mathbb{Z}/(5)$

$$\frac{3}{2} + 4 \equiv 3 \cdot 3 + 4 \equiv 9 + 4 \equiv 13 \equiv 3 \,(\text{mod } 5).$$

The number of invertible elements of $\mathbb{Z}/(m)$ is denoted by $\phi(m)$. Thus $\phi(4) = 2$. The function ϕ is called *Euler's ϕ-function.*

4.3 Commutative rings with identity

Theorem 13 shows that $\mathbb{Z}/(m)$ is an example of a *commutative ring with identity,* defined as follows.

Def: Let R be a set on which two binary operations, addition $+$ and multiplication $\cdot$ are defined. Then R is said to be a *commutative ring with identity* if R contains elements 0 and 1, and the following rules hold for all $a, b, c \in R$

(*i*) $a + b \in R,\ \ a \cdot b \in R.$ [*closure laws for addition and multiplication*]

(*ii*) $(a + b) + c = a + (b + c)$,
$(a \cdot b) \cdot c = c \cdot (b \cdot c)$ [*associative laws for addition and multiplication*]

(*iii*) $a + b = b + a$, $a \cdot b = b \cdot a$ [*commutative laws for addition and multiplication*]

(*iv*) $a + 0 = a$, $a \cdot 1 = a$ [*additive and multiplicative identity elements*]

(*v*) There exists $-a \in R$ such that $a + (-a) = 0$. [*existence of additive inverse*]

(*vi*) $a \cdot (b + c) = a \cdot b + a \cdot c.$ [*distributive law for multiplication over addition*]

Note that in $\mathbb{Z}/(m)$, the additive and multiplicative identities 0 and 1 are $[0]_m$ and $[1]_m$, respectively. Familiar examples of commutative rings with identity include $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$, as well as polynomials (chapter 5).

The additive and multiplicative identities can be shown to be unique. Furthermore for all $a, b, \in R$, we have

(*i*) $a \cdot 0 = 0$

(*ii*) $(-a) \cdot b = -(a \cdot b)$.

See Section 7.1 for proofs.

Example 4.33. The set $[0]_2$ is a commutative ring without identity, while $[1]_2$ is not a ring at all, since it is not closed under addition.

Exercises

Exercise 4.1. (All quantities here are integers.) Let $a \equiv_m b$; prove that

(a) if d is a divisor of m, then $a \equiv_d b$

(b) if $n > 0$, then $a^n \equiv_m b^n$.

Exercise 4.2. Determine all solutions to the equation $x^2 = x$ in each of $\mathbb{Z}/(6)$, $\mathbb{Z}/(7)$, and $\mathbb{Z}/(8)$.

Exercise 4.3. Let m be an integer, and $s, t \in \mathbb{Z}/(m)$.

(a) Determine all invertible elements in each of $\mathbb{Z}/(6)$, $\mathbb{Z}/(7)$, $\mathbb{Z}/(8)$.

(b) Prove that if s and t are invertible, then so is s^{-1} and st.

(c) Suppose that t is invertible. Prove that $st = [0]_m$ if and only if $s = [0]_m$.

Exercise 4.4. Evaluate the following expressions in $\mathbb{Z}/(7)$. In each case show your calculations, and give an answer of the form $[k]_7$, where $0 \leq k < 7$. (In the computation, you may use congruence notation.)

(a) $[7004]_7[7003]_7 + [1000]_7$ (b) $[-11]_7[-2]_7^{-1} + [-8]_7^2$

(c) $\displaystyle\sum_{k=1}^{6} \frac{[1]_7}{[k]_7}$ (d) $[3]_7^{3000001}$

Exercise 4.5. Let n be a non-negative integer, whose decimal notation is $d_k d_{d-1} \ldots d_0$.

(a) Show that if $m = 3$ or $m = 9$, then

$$[n]_m = [d_0 + d_1 + \cdots + d_k]_m.$$

(b) Show that if $m = 3$ or $m = 9$, then m divides n if and only if m divides $d_0 + d_1 + \cdots + d_k$.

(c) Using (b), find a 10-digit integer which is divisible by 3, but not by 9.

Exercise 4.6. Let $x \in \mathbb{Z}/(m)$. We say that x is a square if there exists $y \in \mathbb{Z}/(m)$ such that $x = y^2$.

(a) Find all squares in $\mathbb{Z}/(13)$.

(b) Write an algorithm to the following specifications

`Algorithm MSquare`

`INPUT`: $a, m \in \mathbb{Z}$, $m > 1$.

`OUTPUT`: TRUE is $[a]_m$ is a square in $\mathbb{Z}/(m)$, and FALSE otherwise.

Exercise 4.7. The *additive order* of $[x]_m \in \mathbb{Z}/(m)$ is the smallest positive integer t such that $[tx]_m = [0]_m$. Write an algorithm to the following specifications

`Algorithm AddOrder`

`INPUT`: $x, m \in \mathbb{Z}$, $m > 0$.

`OUTPUT`: t, where t is the additive order of $[x]$ in $\mathbb{Z}/(m)$.

Prove that your algorithm terminates, i.e., that the additive order exists for every element of $\mathbb{Z}/(m)$.

Exercise 4.8. The *multiplicative order* of $[x] \in \mathbb{Z}/(m)$ is the smallest positive integer t such that $[x]^t = [1]$. Such integer does not necessarily exist.

(a) Compute the multiplicative order of $[11]$ in $\mathbb{Z}/(13)$.

(b) Describe the structure of the sequence $t \mapsto x^t$ in the case in which the multiplicative order of $x \in \mathbb{Z}/(m)$ is not defined. [$\not\ell$]

Chapter 5

Polynomials

We are familiar with polynomials with real coefficients. We now consider polynomials with other types of coefficients. Let $R = \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ or $\mathbb{Z}/(m)$ (or indeed, any commutative ring with identity).

Def: A *polynomial* a with coefficients in R (also called a polynomial *over* R), is an expression of the form

$$a = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1} + a_nx^n = \sum_{k=0}^{n} a_kx^k$$

where $a_0, a_1, \ldots, a_n \in R$. The quantity x is called the *indeterminate;* each summand a_kx^k is a *monomial.*

One normally writes a_1x and a_0 for a_1x^1 and a_0x^0, respectively. Furthermore, terms of the form $0x^i$ are usually left out, and one writes x^i for $1x^i$. Finally, to represent coefficients of a polynomial over $\mathbb{Z}/(m)$, we use the shorthand notation i for $[i]_m$.

Example 5.34. Let $R = \mathbb{Z}/(6)$, the polynomial

$$[-5]_6\, x^3 + [12]_6\, x^2 + [1]_6\, x^1 + [-3]_6\, x^0$$

is written as $-5\, x^3 + x - 3$ or $x^3 + x + 3$, etc.

Let $a_0 + a_1x + \cdots + a_nx^n$ be a polynomial with coefficients in R. We call a the *zero polynomial,* and write $a = 0$, if $a_0 = a_1 = \cdots = a_n = 0$. (Recall that $0 = [0]_m$ if $R = \mathbb{Z}/(m)$.)

Def: The *degree* $\deg(a)$ of a polynomial $a \neq 0$ is the greatest integer k such that $a_k \neq 0$. If a is the zero polynomial, we let $\deg(a) = -1$.

Example 5.35. $R = \mathbb{Z}$.

$$\deg(0x^4 + 3x^2 + 2x - 3) = 2 \qquad \deg(-1) = 0 \qquad \deg(0) = -1.$$

Def: The *leading coefficient* $\mathrm{ldcf}(a)$ of a non-zero polynomial a is $a_{\deg(a)}$. If $a = 0$, we define $\mathrm{ldcf}(0) = 0$.

Example 5.36. Let $R = Z/(5)$, and $a = -10x^4 + 8x^2 + 3$. Then $a = 3x^2 + 3$ and so $\mathrm{ldcf}(a) = [3]_5$ ($\deg(a) = 2$).

Let $a = a_0 + a_1x + \cdots + a_nx^n$, $b = b_0 + b_1x + \cdots + b_lx^l$ be polynomials with coefficients in R. We consider a and b to be equal, and write $a = b$, if $\deg(a) = \deg(b)$ and $a_i = b_i$ for $i = 0, 1, \ldots, \deg(a)$. Thus $a = b$ precisely when equality holds for the corresponding sequences of coefficients

$$(a_0, a_1, \ldots) = (b_0, b_1, \ldots)$$

in the sense of equation (2.2), Section 2.3.

We denote by $R[x]$ the set of all polynomials with coefficients in R. Let $a, b \in R[x]$. Then we can add and multiply the polynomials a and b in the usual way, taking care to add and multiply the coefficients correctly in R. Then $x + b \in R[x]$ and $ab \in R[x]$.

In the theory of polynomials the indeterminate does not play an active role: its purpose is to organize the coefficients is such a way that the arithmetical operations can be defined naturally. Polynomials over R could be defined as finite sequences of elements of R, without any reference to an indeterminate.

Example 5.37. Let $R = \mathbb{Z}/(6)$, and let $a, b \in R[x]$ be given by

$$a = 2x^2 + x + 1 \qquad\qquad b = 3x^3 + 5x^2 + 2x + 3.$$

Then

$$\begin{aligned} a + b &= (0 + 3)x^3 + (2 + 5)x^2 + (1 + 2)x + (1 + 3) \\ &= 3x^3 + x^2 + 3x + 4. \end{aligned}$$

$$\begin{aligned} ab &= 2x^2b + xb + b \\ &= 6x^5 + 10x^4 + 4x^3 + 6x^2 + 3x^4 + 5x^3 \\ &\qquad +2x^2 + 3x + 3x^3 + 5x^2 + 2x + 3 \\ &= x^4 + x^2 + 5x + 3. \end{aligned}$$

Note that in this case $\deg(ab) = 4 \neq \deg(a) + \deg(b) = 5$.

It can be shown that if R is a commutative ring with identity, then so is $R[x]$ (with $1 = 1x^0$ and $0 = 0x^0$).

Def: Let $a \in R$, a commutative ring with identity. We say that a is *invertible* if there exists an element $b \in R$ such that $ab = 1$.

In $\mathbb{Z}$, the only invertible elements are 1 and -1. In $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ all elements except 0 are invertible. We shall show that if p is a prime, then all elements of $\mathbb{Z}/(p)$, except 0 are invertible, which will give the arithmetic modulo a prime a special status.

Theorem 15 *Let R be a commutative ring with identity, let $a, b \in R[x]$, with $b \neq 0$, and let* ldcf(b) *be an invertible element of R. Then there are unique polynomials $q, r, \in R[x]$ such that*

$$a = bq + r \qquad \deg(0) \leq \deg(r) < \deg(b). \tag{5.1}$$

The proof is given in chapter 7. Note the structural similarity with Theorem 3, chapter 2.

We denote this unique q and r by $a \,\text{DIV}\, b$ and $a \,\text{MOD}\, b$, respectively. The quantities $a \,\text{DIV}\, b$ and $a \,\text{MOD}\, b$ can be calculated by 'long division of polynomials'.

Example 5.38. Let $R = \mathbb{Z}/(3)$, and let $a = x^4 + 1$, $b = 2x^2 + x + 2 \in R[x]$. We compute $a \,\text{DIV}\, b$ and $a \,\text{MOD}\, b$ by long division

$$\begin{array}{rl|l} & 2x^2 \;+2\,x & = \quad a \,\text{DIV}\, b \\ 2x^2 + x + 2 & \overline{\;\; x^4 \qquad\qquad +1} & \\ & -(x^4 \;+2x^3 \;+x^2) & \\ \hline & \qquad x^3 \;+2x^2 \qquad +1 & \\ & \quad -(x^3 \;+2x^2 \;+x) & \\ \hline & \qquad\qquad 2x \;+1 & = \quad a \,\text{MOD}\, b \end{array}$$

To develop an algorithm for quotient and remainder of polynomial division, we require the notion of a *loop invariant.*

5.1 Loop invariants

Proving statements about algorithms that contain loops, requires a variant of the method of induction, which is based on the notion of a *loop invariant.*

Def: Let `W` be a while-loop with loop control expression β. A *loop invariant* $\mathcal{L}$ for `W` is a boolean expression which evaluates to TRUE when β is evaluated.

Whether $\mathcal{L}$ is evaluated before or after β is immaterial, since the evaluation of a boolean expression does not alter the value of any variable. Thus a loop invariant is TRUE when the loop first starts execution, and is TRUE after each complete execution of the statement-sequence of that loop. Therefore, proving that $\mathcal{L}$ is a loop invariant is mathematical induction in disguise.

The base case consists of showing that $\mathcal{L}$ is TRUE when `W` is first entered (before its statement-sequence is executed even once). The inductive hypothesis amounts to assuming that $\mathcal{L}$ is TRUE at the beginning of the execution of `W`'s statement-sequence. The inductive step amounts to proving that then $\mathcal{L}$ is also TRUE at the end of the execution of statement-sequence (it does not matter if $\mathcal{L}$ is FALSE at some other point in the statement-sequence).

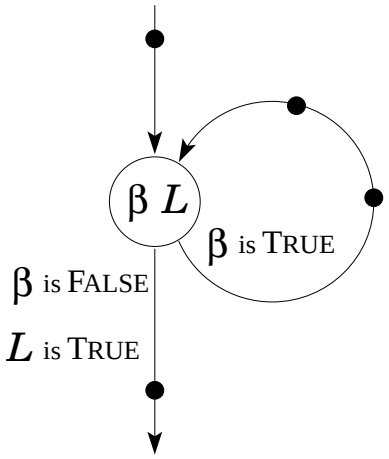

Figure 5.1: A loop with loop invariant $\mathcal{L}$ and loop control expression β. The filled-in circles denote expressions, which may change the value of $\mathcal{L}$ and β. However, inside the loop, the value of $\mathcal{L}$ is eventually restored to TRUE.

Suppose we are given a while-loop `W` of the form

`while` β `do`
    $\langle$ *statement-sequence* $\rangle$
`od`;

where $\mathcal{L}$ is a proven loop invariant for `W`. Then, if `W` terminates execution normally (after the `od`;), we know that on this terminations of `W` we must have that $\mathcal{L}$ is TRUE and β is FALSE (see Figure 5.1). This knowledge can help us prove that an algorithm works. Clearly, any boolean expression that always evaluates to TRUE (such as TRUE, or $2 < 3$) is a trivial loop invariant for any loop. The difficulty lies in identifying 'useful' loop invariants.

As an illustration of loop invariance, we develop an algorithm for quotient and remainder of polynomial division. Let $R = \mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$ or $\mathbb{Z}/(m)$ (or indeed, any commutative ring with identity).

`Algorithm PolynomialQuoRem`
`INPUT`: $a, b \in R[x]$, $b \neq 0$, and $\mathrm{ldcf}(b)$ invertible.
`OUTPUT`: $q, r \in R[x]$, such that $q = a \,\mathrm{DIV}\, b$ and $r = a \,\mathrm{MOD}\, b$.
$q := 0$;
$r := a$;
$\gamma := \mathrm{ldcf}(b)^{-1}$;
`while` $\deg(r) \geq \deg(b)$ `do`
    $t := \gamma \cdot \mathrm{ldcf}(r) \cdot x^{\deg(r)-\deg(b)}$;
    $q := q + t$; (* the quotient is updated *)
    $r := r - tb$; (* the degree of r is lowered *)
`od`;

```
return (q, r);          (*  now, a = bq + r and deg(r) < deg(b)  *);
end;
```

In this algorithm q stores the current value of the quotient, and r that of the remainder. The statement $t := \gamma \cdot \mathrm{ldcf}(r) \cdot x^{\deg(r)-\deg(b)}$; achieves the purpose of matching degree and leading coefficient of r and tb: $\deg(tb) = \deg(r) \geq 0$, and $\mathrm{ldcf}(tb) = \mathrm{ldcf}(r)$. Because the loop control expression is $\deg(r) \geq \deg(b)$, and $a = bq + r$ is a loop invariant (see below), on termination, equation 5.1 holds.

Example 5.39. Let $R = \mathbb{Z}/(3)$, and let $f = x^4 + 1$, $g = 2x^2 + x + 2 \in R[x]$. We trace

`PolynomialQuoRem`(f, g)

a	b	q	r	γ	$\deg(r) \geq \deg(b)$	t
x^4+1	$2x^2+x+2$	0	x^4+1	2	T	$2x^2$
		$2x^2$	x^3+2x^2+1		T	$2x$
		$2x^2+2x$	$2x+1$		F	

`return` $(2x^2 + 2x, 2x + 1)$

Thus $f \,\mathrm{DIV}\, g = 2x^2 + 2x$ and $f \,\mathrm{MOD}\, g = 2x + 1$, as we found in Example 5.38.

Proposition 16 *The algorithm* `PolynomialQuoRem` *is correct.*

We first prove by induction that $a = bq + r$ is a loop invariant for the while-loop of `PolynomialQuoRem`.

(*Induction basis.*) When this while-loop first starts execution, we have $q = 0$, and $r = a$, so

$$bq + r = 0 + a = a.$$

We next show that if $a = bq + r$ holds at the beginning of the while-loop's statement-sequence, then $a = bq + r$ holds at the end of that statement-sequence.

(*Induction step.*) Let q' and r' be the respective values of q and r at the beginning of the statement-sequence, and assume that $a = bq' + r'$. Now the statement-sequence does not change the values of a and b, but does change the values of q and r, by assigning the value $q' + t$ to q, and $r' - tb$ to r. Thus at the end of the statement-sequence, we have

$$\begin{aligned} bq + r &= b(q' + t) + r' - tb \\ &= bq' + r' + bt - tb \\ &= a + 0 \\ &= a, \end{aligned}$$

as required. This completes the proof of the loop invariant of the while-loop of the algorithm `PolynomialQuoRem`.

Now each time the statement-sequence of the while-loop of `PolynomialQuoRem` is executed, the degree of r is decreased, and b is unchanged. Thus, after finitely many executions of this statement-sequence, we will have $\deg(r) < \deg(b)$ (because $\deg(b) \geq 0$), and the while-loop will terminate.

Upon this termination we know that the loop invariant $a = bq + r$ is TRUE, and $\deg(r) \geq \deg(b)$ is FALSE. Thus $a = bq + r$ and $\deg(r) < \deg(b)$, which means that $q = a \operatorname{DIV} b$, and $r = a \operatorname{MOD} b$. Thus `PolynomialQuoRem` works, terminating after a finite time and returning the correct output. ■

5.2 Recursive algorithms

We have seen (Section 2.5) that one algorithms can call another algorithm. An algorithm is said to be *recursive* when it calls itself. More precisely, a recursive algorithm calls a different and independent execution of the same algorithm.

Example 5.40. A recursive algorithm to compute $n!$.

```
Algorithm Factorial

INPUT: n, a positive integer.
OUTPUT: n!.
if n = 1 then
  return 1;
else
  return n · Factorial(n − 1);
fi;
end;
```

Example 5.41. We execute $\texttt{Factorial}(4) = \texttt{Factorial}_1(4)$.

$\texttt{Factorial}_1(4)$

Alg	n	$n = 1$	$n \cdot \texttt{Factorial}_k(n-1)$	`return`
$\texttt{Factorial}_1$	4	F	$4 \cdot \texttt{Factorial}_2(3)$	24
$\texttt{Factorial}_2$	3	F	$3 \cdot \texttt{Factorial}_3(2)$	6
$\texttt{Factorial}_3$	2	F	$2 \cdot \texttt{Factorial}_4(1)$	2
$\texttt{Factorial}_4$	1	T		1

`return` 24

Induction is the most common device used to prove the correctness of a recursive algorithm.

Proposition 17 *The algorithm* `Factorial` *is correct.*

Proof: By induction of the input n.

(*Induction basis.*) If $n = 1$, then `Factorial` terminates and returns the correct result 1.

(*Induction step.*) Let k be an arbitrary, but fixed positive integer, and assume that algorithm `Factorial` works correctly with input $n = k$. Then if $n = k + 1$, the algorithm returns

$$\begin{aligned}(k+1)\cdot \texttt{Factorial}(k) &= (k+1)\cdot k! \\ &= (k+1)\cdot k\,(k-1)\cdots 2\cdot 1 \\ &= (k+1)!\end{aligned}$$

which is correct. ■

5.3 Greatest common divisors

In this section we introduce Euclid's algorithm, one of the best known recursive algorithms of discrete mathematics.

Def: A *field* F is a commutative ring with identity, which contains at least 2 elements, and such that if $0 \neq \alpha \in F$, then α is invertible.

The sets $\mathbb{Q}, \mathbb{R}, \mathbb{C}$ are fields. The commutative ring with identity $\mathbb{Z}/(2) = \{[0]_2, [1]_2\}$ is a field, because its only nonzero element $[1]_2$ is invertible, as easily verified. We will prove later that $\mathbb{Z}/(p)$, where p is a *prime* is a field.

Let F be a field. If $a, b \in F[x]$, with $b \neq 0$, then $a \,\text{DIV}\, b$ and $a \,\text{MOD}\, b$ are always defined (since ldcf(b) is invertible), and we have

$$a = (a \,\text{DIV}\, b)b + a \,\text{MOD}\, b \qquad \deg(0) \leq \deg(a \,\text{MOD}\, b) < \deg(b).$$

Recall that a similar situation holds in $\mathbb{Z}$. If $a, b, \in \mathbb{Z}$, $b \neq 0$, then

$$a = (a \,\text{DIV}\, b)b + a \,\text{MOD}\, b \qquad |0| \leq |a \,\text{MOD}\, b| < |b| \tag{5.2}$$

and for this reason we shall deal with division with integers and with polynomials over a field in a unified manner.

Thus let $E = \mathbb{Z}$, or $E = F[x]$, and $a, b \in E$. We say that b *divides* a, and write $b|a$, if $a = bc$ for some $c \in E$. We call $d \in E$ a *common divisor* of a and b if $d|a$ and $d|b$. We call g a *greatest common divisor* (or gcd) of a and b if

(i) g is a common divisor of a and b

(ii) if d is any common divisor of a and b, then $d|g$.

Example 5.42. Let $E = \mathbb{Z}$, $a = -18$, $b = 24$. The common divisors of a and b are

$$1, -1, 2, -2, 3, -3, 6, -6.$$

The greatest common divisors of a and b are 6 and -6.

Theorem 18 *Let $E = \mathbb{Z}$ or $E = F[x]$, where F is a field, and let $a, b \in E$. The following holds:*

(*i*) *if $b = 0$, then a is a gcd of a and b;*

(*ii*) *if $b \neq 0$, then g is a gcd of a and b iff g is a gcd of b and $a \bmod b$.*

Proof: Part (*i*) follows immediately from the fact that a divides 0 and a. To prove (*ii*), we first show that

$$g|a \text{ AND } g|b \quad \Longleftrightarrow \quad g|a \text{ AND } g|(a \bmod b).$$

Indeed

($\Longrightarrow$) suppose $g|a$ and $g|b$. Then $g|b$ and $g|a - b(a \operatorname{DIV} b) = a \bmod b$.

($\Longleftarrow$) Suppose $g|b$ and $g|a \bmod b$. Then $g|b(a \operatorname{DIV} b) + a \bmod b = a$, and $g|b$.

Thus, if $a, b \in E$ and $b \neq 0$, the pairs a, b and $b, a \bmod b$ have exactly the same set of common divisors. This implies they also have the same set of greatest common divisors. ■

Let F be a field, and let $E = \mathbb{Z}$ or $E = F[x]$. For the gcd algorithms below, we assume that if $E = F[x]$, then we have algorithms for exact computations in F (which is certainly the case when $F = \mathbb{Q}$ or $\mathbb{Z}/(p)$, p prime). We begin with the celebrated *Euclid's algorithm*

```
Algorithm GCD
INPUT: a, b ∈ E.
OUTPUT: g ∈ E, such that g is a gcd of a and b.
if b = 0 then
   return a;
else
   return  GCD(b, a MOD b);
fi;
end;
```

Next, we consider the *extended Euclid's algorithm,* which, in addition to computing a greatest common divisor, also expresses it as a linear combination of the input data.

```
Algorithm ExtendedGCD
INPUT: a, b ∈ E.
OUTPUT: g, s, t ∈ E, such that g is a gcd of a and b, and g = sa + tb.
if b = 0 then
   return (a, 1, 0);
else
   (g, s, t) := ExtendedGCD(b, a MOD b);
   return (g, t, s − t (a DIV b));
fi;
end;
```

Of note is the assignment to a sequence: $(g, s, t) :=$ `ExtendedGCD`$(b, a \,\mathrm{MOD}\, b)$, whereby each variable on the left hand side is assigned the value of the corresponding output of `ExtendedGCD`.

Example 5.43. Let $F = \mathbb{Z}/(5)$, $E = F[x]$. We determine $\mathtt{GCD}(x^2 + 2x + 1, x^2 + 4)$.

$$\mathtt{GCD}_1(x^2 + 2x + 1, x^2 + 4)$$

Alg	a	b	$b = 0$	$a \,\mathrm{MOD}\, b$	$\mathtt{GCD}_k(b, a \,\mathrm{MOD}\, b)$	`return`
$\mathtt{GCD}_1$	$x^2 + 2x + 1$	$x^2 + 4$	F	$2x + 2$	$\mathtt{GCD}_2(x^2 + 4, 2x + 2)$	$2x + 2$
$\mathtt{GCD}_2$	$x^2 + 4$	$2x + 2$	F	0	$\mathtt{GCD}_3(2x + 2, 0)$	$2x + 2$
$\mathtt{GCD}_3$	$2x + 2$	0	T			$2x + 2$

$$\mathtt{return}\, 2x + 2$$

Def: Let $E = \mathbb{Z}$ or $F[x]$, where F is a field. For $a \in E$, define $\delta(a) = |a|$ if a is an integer, and $\delta(a) = \deg(a)$ if a is a polynomial.

Suppose a and b are in E. Then we always have that $\delta(b)$ is an integer, with $\delta(b) \geq \delta(0)$. In fact, $\delta(b) = \delta(0)$ if and only if $b = 0$. Furthermore, if b is nonzero, we have that $\delta(0) \leq \delta(a \,\mathrm{MOD}\, b) < \delta(b)$.

Proposition 19 *The algorithm* `ExtendedGCD` *is correct.*

Proof:

We do this by *strong induction* on $n = \delta(b)$, proving that for all $n \geq \delta(0)$, the algorithm `ExtendedGCD` works correctly for all input $a, b \in E$, with $\delta(b) = n$. (Note that this covers all possible input a, b to `ExtendedGCD`.)

(*Induction basis.*) If $n = \delta(b) = \delta(0)$, then $b = 0$ so a is a gcd of a and b, and $a = 1a + 0b$. Thus when $n = \delta(0)$ the algorithm `ExtendedGCD` terminates with the correct output $(a, 1, 0)$.

(*Induction step.*) Now assume that $n = \delta(b) > \delta(0)$, and that `ExtendedGCD` works correctly with input a', b' in E, whenever $\delta(b') < n$.

Since $\delta(b) > \delta(0)$, b must be nonzero, and `ExtendedGCD` first sets

$$(g, s, t) := \texttt{ExtendedGCD}(b, a \bmod b).$$

Since $\delta(a \bmod b) < \delta(b)$, from our inductive hypothesis, we have that g is a gcd of b and $a \bmod b$, and that $g = sb + t(a \bmod b)$.

It follows that g is a gcd of a and b (by Theorem 18, page 54), and that

$$\begin{aligned} g &= sb + t(a \bmod b) \\ &= sb + t(a - b(a \operatorname{div} b)) \\ &= ta + (s - t(a \operatorname{div} b))b. \end{aligned}$$

Thus `ExtendedGCD` terminates and returns the correct result $(g, t, s - t(a \operatorname{div} b))$, and the proof is complete. ■

The proof that algorithm `GCD` is correct is a simplified version of the above proof, and it is left as an exercise.

Example 5.44. We compute a greatest common divisor g of the polynomials $a = x^6 + 2x^5 + x$ and $b = x^4 + 3x^3 + 2x^2 + 4x + 4$ in $\mathbb{Z}/(5)$, and then express g as a linear combination of a and b. The computation of $\texttt{ExtendedGCD}(x^6 + 2x^5 + x,\ x^4 + 3x^3 + 2x^2 + 4x + 4)$, given in Table I, returns the sequence

$$(g, s, t) := (x + 3,\ 3x^2 + 3x + 4,\ 2x^4 + x^2 + x + 2).$$

We verify

$$\begin{aligned} sa + tb &= (3x^2 + 3x + 4)(x^6 + 2x^5 + x) \\ &\qquad + (2x^4 + x^2 + x + 2)(x^4 + 3x^3 + 2x^2 + 4x + 4) \\ &= 5x^8 + 15x^7 + 15x^6 + 20x^5 \\ &\qquad + 15x^4 + 15x^3 + 15x^2 + 16x + 8 \\ &= x + 3. \end{aligned}$$

Let F be a field, let $E = \mathbb{Z}$ or $F[x]$, and let $a, b \in E$. Recall that a gcd of a and b is defined to be a common divisor of g of a and b, such that g is divisible by each common divisor of a and b. It follows that, if $E = \mathbb{Z}$, then g is a gcd of a and b if and only if $-g$ is a gcd of a and b.

Now suppose that $E = F[x]$, and that g and g' are both gcds of a and b. Since g and g' are both common divisors of a and b, it follows from the definition of gcd that $g|g'$ and

TABLE I Tracing the calls to `ExtendedGCD` of Example 5.44, over $\mathbb{Z}/(5)$.

$\texttt{ExtendedGCD}_1(x^6 + 2x^5 + x,\ x^4 + 3x^3 + 2x^2 + 4x + 4)$

Alg	a	b	$b = 0$	$a \bmod b$	$\texttt{EGCD}_k$	g	s	t	$a \operatorname{DIV} b$	`return`
$\texttt{EGCD}_1$	$x^6 + 2x^5 + x$	$x^4 + 3x^3 + 2x^2 + 4x + 4$	F	$3x^2 + x + 1$	$\texttt{EGCD}_2$	$x + 3$	1	$3x^2 + 3x + 4$	$x^2 + 4x + 1$	$(x + 3,\ 3x^2 + 3x + 4,\ 2x^4 + x^2 + x + 2)$
$\texttt{EGCD}_2$	$x^4 + 3x^3 + 2x^2 + 4x + 4$	$3x^2 + x + 1$	F	$x + 3$	$\texttt{EGCD}_3$	$x + 3$	0	1	$2x^2 + 2x + 1$	$(x + 3,\ 1,\ 3x^2 + 3x + 4)$
$\texttt{EGCD}_3$	$3x^2 + x + 1$	$x + 3$	F	0	$\texttt{EGCD}_4$	$x + 3$	1	0	$3x + 2$	$(x + 3,\ 0,\ 1)$
$\texttt{EGCD}_4$	$x + 3$	0	T							$(x + 3, 1, 0)$

`return` $(x + 3,\ 3x^2 + 3x + 4,\ 2x^4 + x^2 + x + 2)$

and $g'|g$. We must therefore have (see exercises) that $\deg(g) = \deg(g')$ and that $g' = fg$ for some degree zero polynomial f in $F[x]$. Furthermore, for all degree zero polynomials f in $F[x]$, fg really is a gcd of a and b (see exercises).

Conclusion: if $E = \mathbb{Z}$ then $\{-g, g\}$ is the set of gcds of a and b. If $E = F[x]$, then $\{fg \mid f \in F[x], \deg(f) = 0\}$ is a set of gcds of a and b.

5.4 Modular inverse

Theorem 20 *Let m and i be integers. Then $[i]_m$ is an invertible element of $\mathbb{Z}/(m)$ if and only if 1 is a gcd of m and i.*

Proof: ($\Longrightarrow$) Suppose $[i]_m$ is an invertible element of $\mathbb{Z}/(m)$, and let $[j]_m = [i]_m^{-1}$. Then

$$[1]_m = [i]_m[j]_m = [ij]_m$$

which implies that $1 = ij + km$ for some integer k. We also see from this that any common divisor of m and i also divides 1, which is also a common divisor of m and i. From the definition of gcd, we have that 1 is a gcd of m and i.

($\Longleftarrow$) Suppose 1 is a gcd of m and i. Then $1 = sm + ti$, for some integers s and t, from the extended Euclid's algorithm (if the algorithm returns -1 as a gcd, then t and s are the negatives of the corresponding values in the output sequence). Therefore

$$\begin{aligned} [1]_m &= [sm + ti]_m \\ &= [s]_m[m]_m + [t]_m[i]_m \\ &= [s]_m[0]_m + [t]_m[i]_m \\ &= [t]_m[i]_m. \end{aligned}$$

Thus $[i]_m$ is invertible, and $[t]_m = [i]_m^{-1}$. ■

The above theorem and proof show us how to determine if an element of $\mathbb{Z}/(m)$ is invertible, and if so, how to find its inverse. We implement this method in the algorithm `Inverse`

```
Algorithm Inverse
INPUT: i, m ∈ ℤ, with m > 1.
OUTPUT: t, such that [t]_m = [i]_m^{-1}, if [i]_m is invertible;
        the empty sequence otherwise.
(g, s, t) := ExtendedGCD(m, i);
 if g = 1 then
   return t;     (*  since 1 = sm + ti  *)
fi;
if g = −1 then
   return −t;    (*  since 1 = −sm − ti  *)
```

```
fi;
return ();     (*  at this point we know that 1 is not a gcd of m, i  *)
end
```

Example 5.45. Trace `Inverse`$(19, 21)$.

i	m	i	`ExtendedGCD`(m, i)	g	s	t	$g = 1$	$g = -1$	`return`
19	21	19	`ExtendedGCD`$_1(21, 19)$	1	-9	10	T		10

We trace all calls to `ExtendedGCD` in a single table, writing `EGCD` for `ExtendedGCD`

Alg	a	b	$b = 0$	a MOD b	`EGCD`(m, i)	g	s	t	a DIV b	`return`
`EGCD`$_1$	21	19	F	2	`EGCD`$_2(19, 2)$	1	1	-9	1	$(1, -9, 10)$
`EGCD`$_2$	19	2	F	1	`EGCD`$_3(2, 1)$	1	0	1	9	$(1, 1, -9)$
`EGCD`$_3$	2	1	F	0	`EGCD`$_4(1, 0)$	1	1	0	1	$(1, 0, 1)$
`EGCD`$_4$	1	0	T							$(1, 1, 0)$

`return`10

Thus $[19]_{21}$ is invertible, and $[19]_{21}^{-1} = [10]_{21}$.

Theorem 21 *Let $m > 1$ be an integer. Then each nonzero element of $\mathbb{Z}/(m)$ is invertible if and only if m is prime.*

Proof: ($\Longrightarrow$) (By contradiction.) Suppose m is not prime. Then $m = ij$ for some integers i, j, with $1 < i < m$. Note that $[i]_m$ is a nonzero element of $\mathbb{Z}/(m)$. Since $i, -i$ are the gcds of m and i, we have that 1 is not a gcd of m and i, and applying Theorem 20, we see that $[i]_m$ is not invertible.

($\Longleftarrow$) Suppose m is prime. Then the only divisors of m are $1, -1, m, -m$. Thus when $1 \leq i < m$, we have that 1 is a gcd of m and i. Applying Theorem 20, we have that $[i]_m$ is invertible for each $i = 1, 2, \ldots, m - 1$.

■

Let m be an integer. Applying Theorem 21, we deduce the following important

Corollary 22 *The set $\mathbb{Z}/(m)$ is a field if and only if m is prime.*

These are prominent examples of *finite fields.* It can be shown that every field whose cardinality is a prime number p, is equal to $\mathbb{Z}/(p)$ (more precisely, it has the same arithmetic).

5.5 Polynomial evaluation

Let $R = \mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}/(m)$ (or indeed, let R be any commutative ring with identity), and let $f = f_0 + f_1 x + \cdots + f_n x^n \in R[x]$. Then f defines a function as follows

$$f : R \to R \qquad \alpha \mapsto f(\alpha) = f_0 + f_1\alpha + \cdots + f_n\alpha^n.$$

We say that $f(\alpha)$ is the polynomial f *evaluated at* α. Note that for $f, g \in R[x]$, $\alpha \in R$, we have

$$\begin{aligned} (f+g)(\alpha) &= f(\alpha) + g(\alpha) \\ (f\cdot g)(\alpha) &= f(\alpha)\cdot g(\alpha) \\ (-f)(\alpha) &= -f(\alpha) \end{aligned}$$

which follow from the way addition, subtraction and multiplication are defined for polynomials. The operators $+, -, \cdot$ appearing on the left-hand side of the above equations have a very different meaning from those on the right-hand side. Indeed, the former refer to arithmetic in $R[x]$, the latter to arithmetic in R.

Example 5.46. Let $f = x^3 + x + 1$, $g = 2x + 1 \in \mathbb{Z}/(3)[x]$. Then

$$f(0) = 0^3 + 0 + 1 = 1 \qquad f(1) = 1^3 + 1 + 1 = 0 \qquad f(2) = 2^3 + 2 + 1 = 2.$$
$$g(0) = 2\cdot 0 + 1 = 1 \qquad g(1) = 2\cdot 1 + 1 = 0 \qquad g(2) = 2\cdot 2 + 1 = 2.$$

Thus, $f \neq g$ (as polynomials), but $f(x) = g(x)$ as functions from $\mathbb{Z}/(3)$ to $\mathbb{Z}/(3)$.

If $f(\alpha) = 0$, then we call α a *zero* of f.

Proposition 23 *Let $f \in R[x]$, $\alpha \in R$. Then α is a zero of f if and only if $(x-\alpha) \mid f$.*

Proof: Since $\mathrm{ldcf}(x-\alpha) = 1$ is invertible (Theorem 20, Section 5.4), we have that $f = (x-\alpha)q + r$, for $q, r \in R[x]$, with $\deg(r) < \deg(x-\alpha) = 1$. So we have two possibilities: either $\deg(r) = -1$ and $r = 0$, or $\deg(r) = 0$ and $r = \beta\, x^0$, for some $0 \neq \beta \in R$.

($\Longrightarrow$) Suppose that α is a zero of f, but that $r = \beta x^0 \neq 0$. Then

$$0 = f(\alpha) = (\alpha - \alpha)q(\alpha) + r(\alpha) = 0 + \beta = \beta \neq 0,$$

a contradiction.

($\Longleftarrow$) Suppose $(x-\alpha) \mid f$. Then $f = (x-\alpha)g$ for some $g \in R[x]$, and therefore $f(\alpha) = (\alpha - \alpha)g(\alpha) = 0$. ■

Now, given $f = f_0 + f_1 x + \cdots + f_n x^n \in R[x]$, and $\alpha \in R$, how can we determine $f(\alpha)$ efficiently? The trick is to write

$$\begin{aligned} f &= f_0 + x(f_1 + f_2 x + \cdots + f_n x^{n-1}) \\ &= f_0 + x(f_1 + x(f_2 + f_3 x + \cdots + f_n x^{n-2})) \\ &\vdots \\ &= f_0 + x(f_1 + x(f_2 + x(f_3 + \cdots + x(f_{n-1} + x(f_n)) \cdots))). \end{aligned}$$

This leads to *Horner's algorithm* to evaluate f at α

```
Algorithm Evaluate
INPUT: f = f_0 + f_1 x + ··· + f_n x^n ∈ R[x], α ∈ R.
OUTPUT: f(α).
if f = 0 then
    return 0;
fi;
result := ldcf(f);
i := deg(f) − 1;
while i ≥ 0 do
    result := f_i + α · result;
    i := i − 1;
od;
return result;
end;
```

Example 5.47. Let $f \in \mathbb{Z}/(5)[x]$ be given by

$$\begin{aligned} f &= 4 + x + 2x^3 + 3x^4 \\ &= 4 + x\,(1 + x\,(0 + x\,(2 + x\,(3)))). \end{aligned}$$

We compute $f(3)$. One sees that using the nested expression for f, this will involve four multiplication and four additions.

$$\texttt{Evaluate}(f, 3)$$

result	i
3	3
$3 \cdot 3 + 2 = 1$	2
$3 \cdot 1 + 0 = 3$	1
$3 \cdot 3 + 1 = 0$	0
$3 \cdot 0 + 4 = 4$	-1

Thus, $f(3) = 4$.

Proposition 24 *The algorithm* `Evaluate` *is correct.*

Proof: If $f = 0$, then `Evaluate` terminates with the correct output 0.

Now suppose $\deg(f) = n \geq 0$. The statement-sequence of the while-loop executes for each $i = n-1, n-2, \ldots, 0$, exactly n times (performing n additions and n multiplications in R), then *result* is returned, and the algorithm terminates.

To prove correctness, we prove the loop invariant

$$result = \sum_{j=i+1}^{n} f_j \cdot \alpha^{j-(i+1)}.$$

On entry to the loop, $i = n - 1$, and

$$result = \mathrm{ldcf}(f) = f_n = \sum_{j=(n-1)+1}^{n} f_n \alpha^{j-n}$$

and the loop invariant holds.

Now suppose we are at the beginning of the statement-sequence of the loop, $i = k \geq 0$, and the loop invariant holds. Thus

$$result = \sum_{j=k+1}^{n} f_j \cdot \alpha^{j-(k+1)}.$$

We now execute the loop's statement-sequence, and then show that the loop invariant still holds. We have

$$\begin{aligned} result &= f_i + \alpha \cdot result \\ &= f_k + \alpha \sum_{j=k+1}^{n} f_j \alpha^{j-(k+1)} \\ &= f_k \alpha^0 + \sum_{j=k+1}^{n} f_j \alpha^{j-k} \\ &= \sum_{j=k}^{n} f_j \alpha^{j-k}. \end{aligned}$$

Then $i := i - 1$, and the loop statement-sequence is finished. But now $k = i + 1$, so that

$$result = \sum_{j=k}^{n} f_j \cdot \alpha^{j-k} = \sum_{j=i+1}^{n} f_j \cdot \alpha^{j-(i+1)}$$

and the loop invariant still holds.

When the loop terminates, we have $i = -1$ and the loop invariant holding. Thus, on termination of the loop $result = \sum_{j=0}^{n} f_j \alpha^j$, and the algorithm returns the correct result. ■

5.6 Polynomial interpolation*

Let $F = \mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, or $\mathbb{Z}/(p)$, p prime (or indeed, let F be any field). Suppose we are given $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \in F$. Does there exist an $f \in F[x]$ such that $f(\alpha_1) = \beta_1, \ldots, f(\alpha_n) = \beta_n$? If so, can we calculate one of degree smaller than n?

The answer to both questions is affirmative, using the algorithm `Interpolate` below. It turns out that such f is unique, as we shall prove later.

`Algorithm Interpolate`
`INPUT`: $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \in F$, with $n \geq 1$ and $\alpha_1, \ldots, \alpha_n$ distinct.
`OUTPUT`: $f \in F[x]$ such that $f(\alpha_1) = \beta_1, \ldots, f(\alpha_n) = \beta_n$ and $\deg(f) < n$.
$b := 1x^0$;
$i := 1$;
$f := \beta_1 x^0$;
`while` $i < n$ `do`
  $b := b(x - \alpha_i)$;
  $i := i + 1$;
  $f := f + \dfrac{\beta_i - f(\alpha_i)}{b(\alpha_i)}\, b$;
`od`;
`return` f;
`end`;

Example 5.48. Let $F = \mathbb{Z}/(3)$.

$$\texttt{Interpolate}((0,1,2),(1,0,1))$$

.

b	i	f
$1x^0$	1	$1x^0$
$1x^0$	2	$1 + \frac{(0-1)}{1}x = 2x + 1$
$x(x-1) = x^2 + 2x$	3	$(2x+1) + \frac{(1+1)}{2}b =$ $2x + 1 + 1(x^2 + 2x) = x^2 + x + 1$

`return` $x^2 + x + 1$

Check. Letting $f = x^2 + x + 1 \in \mathbb{Z}/(3)[x]$, we find $f(0) = 1$, $f(1) = 1 + 1 = 1 = 0$, $f(2) = 2^2 + 2 + 1 = 1$, as desired.

Proposition 25 *The algorithm* `Interpolate` *is correct.*

Proof: Termination follows from the fact that the while-loop executes its statement-sequence for each $i = 1, 2, \ldots, n-1$, and then terminates.

We prove the loop invariant $\mathcal{L}$, defined as

$$b = \prod_{l=1}^{i}(x - \alpha_l) \quad \text{AND} \quad f(\alpha_l) = \beta_l,\ l = 1, \ldots, i \quad \text{AND} \quad \deg(f) < i.$$

The loop is first entered with

$$b := 1x^0 \qquad i := 1 \qquad f := \beta_1 x^0.$$

which implies

$$b = 1x^0 = \prod_{l=1}^{0}(x - \alpha_l), \qquad f(\alpha_1) = \beta_1, \qquad \deg(f) < 1,$$

that is, $\mathcal{L}$ is TRUE on entry.

Now suppose $\mathcal{L}$ is TRUE on the loop entry, for some $1 \leq i = k < n$. Then

$$b = \prod_{l=1}^{k-1}(x - \alpha_l), \qquad f(\alpha_l) = \beta_l,\ l = 1, \ldots, k, \qquad \deg(f) < k.$$

And we have the assignments

$$\begin{aligned} b &:= b\,(x - \alpha_k) = (x - \alpha_1) \cdots (x - \alpha_{k-1})(x - \alpha_k) \\ i &:= i + 1 = k + 1. \end{aligned}$$

Now consider

$$f^* = f + \frac{\beta_{k+1} - f(\alpha_{k+1})}{b(\alpha_{k+1})} \cdot b$$

If $j \leq k$, then

$$\begin{aligned} f^*(\alpha_j) &= f(\alpha_j) + \frac{\beta_{k+1} - f(\alpha_{k+1})}{b(\alpha_{k+1})} \cdot b(\alpha_j) \\ &= f(\alpha_j) + \frac{\beta_{k+1} - f(\alpha_{k+1})}{b(\alpha_{k+1})} \cdot 0 = \beta_j \\ f^*(\alpha_{k+1}) &= f^*(\alpha_{k+1}) + \frac{\beta_{k+1} - f(\alpha_{k+1})}{b(\alpha_{k+1})} \cdot b(\alpha_{k+1}) \\ &= \beta_{k+1}. \end{aligned}$$

Also, $\deg(f^*) \leq \max(\deg(f), \deg(b)) = k < i = k + 1$. Now the new $f := f^*$, and we have shown that the loop invariant holds after the execution of the loop's statement-sequence.

On termination of the while-loop, we have $i = n$, and $\mathcal{L}$ is TRUE, so the algorithm returns the correct output on termination. ■

Theorem 26 *Let F be a field, and let $\alpha_1, \ldots, \alpha_n$, $\beta_1, \ldots, \beta_n \in F$, with $\alpha_1, \ldots, \alpha_n$ distinct. Then there exists a unique polynomial $f \in F[x]$, such that $f(\alpha_1) = \beta_1, \ldots, f(\alpha_n) = \beta_n$, and $\deg(f) < n$.*

Proof: The existence of f is given by Proposition 25. As to uniqueness, suppose $f, g \in F[x]$, with

$$f(\alpha_1) = g(\alpha_1) = \beta_1, \cdots, f(\alpha_n) = g(\alpha_n) = \beta_n \qquad \deg(f) < n, \quad \deg(g) < n.$$

Consider the polynomial $h = f - g$. We have

$$h(\alpha_i) = f(\alpha_i) - g(\alpha_i) \qquad i = 1, \ldots, n.$$

Thus, if $h \neq 0$, $h = a\,(x-\alpha_1)\cdots(x-\alpha_n)$, for some $0 \neq a \in F[x]$. Thus, $\deg(h) \geq n$. But this contradicts the fact that

$$h = f - g \qquad \deg(f) < n \qquad \deg(g) < n.$$

Thus, $h - 0$ and $f = g$. ■

Exercises

Exercise 5.1. Let $R = \mathbb{Z}/(4)$, and let $a = 2x^3 + 2x^2 + 3$, $b = 3x^3 + 2x^2 + x + 3$ be polynomials in $R[x]$. Determine

$$\deg(a^2), \quad \mathrm{ldcf}(b)^{-1}, \quad ab, \quad a + a, \quad a + b, \quad a \,\mathrm{MOD}\, b.$$

Exercise 5.2. Let $R = \mathbb{Z}/(7)$, and let $a = 4x^3 + 2x + 1$, $b = 3x^2 + x + 4$ be polynomials in $R[x]$. Use the algorithm `PolynomialQuoRem` to determine

$$a \,\mathrm{DIV}\, b, \quad a \,\mathrm{MOD}\, b, \quad b \,\mathrm{DIV}\, a, \quad b \,\mathrm{MOD}\, a.$$

Exercise 5.3. In the statements below, let m and n assume integer values, and let `W` denote the while-loop.

$a := m$;
$b := n$;
$c := 0$;
`while` $b \neq 0$ `do`
  $c := c + a$;
  $b := b - 1$;
`od`;

(a) For which integer values of n will `W` terminate? Explain your answer.

(b) Prove that $ab + c = mn$ is a loop invariant for `W`.

(c) Suppose that `W` terminates. Show that on termination we have that $c = mn$.

Exercise 5.4. Consider the statements below, where the variables m and n are assumed to have integer values, with $n \geq 0$, and let `W` denote the while-loop in these statements.

$a := m$;
$b := n$;
$c := 0$;
`while` $b > 0$ `do`
  `if` $2 \mid b$ `then`
    $a := 2a$;

```
        b := b/2;
    else
        c := c + a;
        b := b − 1;
    fi;
od;
```

(*a*) Prove that $ab + c = mn$ is a loop invariant for `W`. *b* odd separately.]

(*b*) Why must `W` terminate? Why must $b = 0$ on this termination?

(*c*) Show that on the termination of `W`, we have that $c = mn$.

(*d*) Is this approach to computing mn faster or slower than the approach in the previous problem? Justify your answer.

Exercise 5.5. Prove that the algorithm `GCD` is correct.

Exercise 5.6. Apply the algorithm `GCD`, to determine the greatest common divisor of 11468 and 5551.

Exercise 5.7. Let $F = \mathbb{Z}/(3)$, and let $c = x^3+2x^2+x+2$ and $d = 2x^2+1$ be polynomials in $F[x]$. Apply the algorithm `ExtendedGCD` to determine $g, s, t \in F[x]$, such that g is a gcd of c and d, and $g = sc + td$.

Exercise 5.8. Write down all the invertible elements of $\mathbb{Z}/(30)$.

Exercise 5.9. Apply the algorithm `Inverse` to determine whether $[27]_{38}$ is invertible, and if so, to find its inverse.

Exercise 5.10. Let f and g be non zero polynomials in $F[x]$. Prove that $\deg(fg) = \deg(f) + \deg(g)$.

Exercise 5.11. Suppose that g is a gcd of polynomials a and b in $F[x]$, F a field. Prove that if f is a degree zero polynomial in $F[x]$, then fg is also a gcd of a and b.

Exercise 5.12. Let $K = \mathbb{Z}/(5)$, $b \in K[x]$ and suppose that $2x^2 + x + 3$ is a greatest common divisor of a and b. Write down all the greatest common divisors of a and b.

Exercise 5.13. The Euler's ϕ-function is defined on the positive integers, as follows: $\phi(1) = 1$; for $m > 1$, $\phi(m)$ is the number of invertible elements in $\mathbb{Z}/(m)$. Write the algorithm to the following specifications:

`Algorithm Phi`

`INPUT`: m, a positive integer.

`OUTPUT`: $\phi(m)$.

Exercise 5.14. Consider the recursive algorithm

`Algorithm A`

`INPUT`: $x \in \mathbb{Z}, x > 0$

```
OUTPUT: ??
if x = 1 then
   return 1;
else
   return A(x − 1)/x;
fi;
end;
```

(*a*) Compute A(5).

(*b*) Explain in one sentence what this algorithm does. [$\not\in$]

Exercise 5.15. Consider the recursive algorithm

```
Algorithm B
INPUT: x ∈ ℤ, x > 0
OUTPUT: ??
if x = 1 then
   return 1;
else
   return x^2 + B(x − 1);
fi;
end;
```

(*a*) Trace B(5).

(*b*) Explain in one sentence what this algorithm does. [$\not\in$]

Exercise 5.16. Consider the recursive algorithm

```
Algorithm C
INPUT: x, y ∈ ℤ
OUTPUT: ??
if x = y then
   return 0;
else
  if x < y then
     return 1 + C(x, y − 1);
  else
     return  C(y, x);
  fi;
fi;
end;
```

(*a*) Trace C(3, −2)).

[*Hint:* there should be 7 calls to the algorithm.]

(*b*) Explain in one sentence what this algorithm does. [$\not\epsilon$]

Exercise 5.17. Find two integers a, b, with $0 \leq b < a$ such that the recursive computation of $\texttt{GCD}(a, b)$ involves 10 calls to the function `GCD`. Explain what you are doing.

Chapter 6

Algorithms for vectors

Let F be a field (for example, $\mathbb{Q}, \mathbb{R}, \mathbb{C}$, or $\mathbb{Z}/(p)$, where p is a prime). In this course, a *vector* v of *dimension* n over F is a sequence $v = (a_1, a_2, \ldots, a_n)$, of length n, such that $a_1, a_2, \ldots, a_n \in F$.

We consider two vectors $v = (a_1, a_2, \ldots, a_n)$ and $w = (b_1, b_2, \ldots, b_m)$ over F to be *equal* if they are equal as sequences (cf. equation (2.2)).

We denote by F^n the set of all vectors of dimension n over F. Thus

$$F^n = \{(a_1, a_2, \ldots, a_n) \,|\, a_1, a_2, \ldots, a_n \in F\}.$$

Example 6.49. A *relation* on a field F is a collection of two-dimensional vectors over F, that is, a subset of F^2 (chapter 3).

Example 6.50. If $F = \mathbb{Z}/(p)$, with p a prime, then F is a field (Corollary 22, page 59) and F^n has exactly p^n elements (see exercises). In the following example, $p = 3$ and $n = 2$

$$(\mathbb{Z}/(3))^2 = \{\,(0,0),\,(0,1),\,(0,2),\,(1,0),\,(1,1),\,(1,2),\,(2,0),\,(2,1),\,(2,2)\,\}.$$

We call $(0, 0, \ldots, 0) \in F^n$, the *zero vector,* and if there is no danger of confusion, denote it by 0 or $\underline{0}$.

Let $v = (a_1, \ldots, a_n) \in F^n$, $v \neq \underline{0}$. Let l be the smallest integer in the range $1 \leq l \leq n$ for which $a_l \neq 0$. Then we call l the *leading index* of v, and denote it by ldindx(v), and we call a_l the *leading term* of v, and denote it by ldterm(v). In other words, the leading index of a vector identifies the location of the leading term. Thus, the vector $(0, 0, 0, 2, 0, 1) \in \mathbb{Z}/(3)^6$ has leading index 4 and leading term $[2]_3$.

If $v = \underline{0}$, then we define the leading index of v to be $n + 1$, but leave ldterm(v) undefined.

Let $v = (a_1, \ldots, a_n)$ and $w = (b_1, \ldots, b_n)$ be vectors in F^n. We define the *sum* of vectors v and w by $v + w = (a_1 + b_1, \ldots, a_n + b_n)$ and the *scalar multiple* of v

by $c \in F$ $cv = (ca_1, \ldots, ca_n)$. We also define $-v = (-a_1, \ldots, -a_n)$ and $v - w = v + (-w)$. Let $w_1, \ldots, w_k \in F^n$. We say that w is a *linear combination* of $w_1, \ldots, w_k$ if $w = c_1w_1 + \cdots + c_kw_k$ for some $c_1, \ldots, c_k \in F$.

The *linear span* of $w_1, \ldots, w_k$, denoted by $\langle w_1, \ldots, w_k \rangle$, is defined to be the set of all linear combinations of $w_1, \ldots, w_k$. In symbols

$$\langle w_1, \ldots, w_k \rangle = \{c_1w_1 + \cdots + c_kw_k \,|\, c_1, \ldots, c_k \in F\}.$$

In the parlance of linear algebra, $W = \langle w_1, \ldots, w_k \rangle$ is a *subspace* of the vector space F^n, often called the subspace *generated by,* or *spanned by,* $w_1, \ldots, w_k$. We shall use the shorthand notation $\langle W \rangle$ for $\langle w_1, \ldots, w_k \rangle$.

Example 6.51. Let $F = \mathbb{Z}/(2)$, and let $w_1 = (1, 0, 1)$, $w_2 = (1, 1, 0) \in F^3$. Then

$$\begin{aligned} \langle w_1, w_2 \rangle &= \{0w_1 + 0w_2,\ 0w_1 + 1w_2,\ 1w_1 + 0w_2,\ 1w_1 + 1w_2\} \\ &= \{(0, 0, 0),\ (1, 1, 0),\ (1, 0, 1),\ (0, 1, 1)\}. \end{aligned}$$

6.1 Echelon form

The general problem is the following. Suppose we have a field F, $w_1, \ldots, w_k \in F^n$, and we want to determine whether $v \in \langle w_1, \ldots, w_k \rangle$. Furthermore, if $v \in \langle w_1, \ldots, w_k \rangle$, we want to determine $a_1, \ldots, a_k \in F$ such that $v = a_1w_1 + \cdots + a_kw_k$, that is, we want to write v as a linear combination of $w_1, \ldots, w_k$.

These problems can be solved by the algorithm `Sift` if the sequence $(w_1, \ldots, w_k)$ is in so-called *echelon form.*

Def: Let F be a field and $w_1, \ldots, w_k \in F^n$. We say that the sequence $(w_1, \ldots, w_k)$ is in *echelon form* if the following holds

(*i*) $w_i \neq \underline{0}$ for $i = 1, \ldots, k$,

(*ii*) $\mathrm{ldterm}(w_i) = 1$, for $i = 1, \ldots, k$,

(*iii*) $\mathrm{ldindx}(w_1) < \mathrm{ldindx}(w_2) < \cdots < \mathrm{ldindx}(w_k)$.

For formal reasons, we shall regard the empty sequence ( ) to be in echelon form.

Example 6.52. Let $F = \mathbb{Z}/(3)$, $n = 3$

vectors	echelon form?
((1,1,2),(0,0,1))	yes
((2,1,1),(0,0,1))	no
((0,0,1),(1,1,2))	no
((1,1,2),(0,0,0),(0,0,1))	no
((0,0,1))	yes
( )	yes

Let F be a field and n a non-negative integer. The algorithm `Sift` below computes in F^n, and we assume that we have algorithms for exact computation in F, which is certainly the case when $F = \mathbb{Q}$ or $\mathbb{Z}/(p)$ (p prime).

`Algorithm Sift`
`INPUT`: v, W, such that $v \in F^n$, and $W = (w_1, \ldots, w_k)$
is a sequence in echelon form of k vectors in F^n.
`OUTPUT`: $u, (a_1, \ldots, a_k)$, such that $u \in F^n$, $a_1, \ldots, a_k \in F$,
$v = u + a_1 w_1 + \cdots + a_k w_k$,
and $\mathrm{ldindx}(u) \notin \{\mathrm{ldindx}(w_1), \ldots, \mathrm{ldindx}(w_k)\}$.

$k := \#W$;
$L := \{\mathrm{ldindx}(w_1), \ldots, \mathrm{ldindx}(w_k)\}$;
$a_j := 0$, for $j = 1, \ldots, k$;
$u := v$;
`while` $\mathrm{ldindx}(u) \in L$ `do` (* **Loop invariant**: $v = u + a_1 w_1 + \cdots + a_k w_k$ *)
$i :=$ the integer i such that $\mathrm{ldindx}(w_i) = \mathrm{ldindx}(u)$;
$a_i := \mathrm{ldterm}(u)$;
$u := u - a_i w_i$; (* this step increases $\mathrm{ldindx}(u)$, since $\mathrm{ldterm}(w_i) = 1$ *)
`od`;
`return` $(u, (a_1, \ldots, a_k))$;
`end`;

The assignment '$i :=$the integer i such that $\mathrm{ldindx}(w_i) = \mathrm{ldindx}(u)$' is to be implemented as '$i :=$`MatchLeadingIndex(u,W)`', where `MatchLeadingIndex` is a suitable algorithm (see exercises).

Proposition 27 *The algorithm* `Sift` *is correct.*

Proof: Let `S` be the statement sequence of the while-loop of `Sift`. Then each time `S` is executed, $\mathrm{ldindx}(u)$ is increased, being made greater than $\mathrm{ldindx}(w_i) = \mathrm{ldindx}(u)$, for a different i each time `S` is executed. This shows that `S` is executed at most k times, after which `Sift` returns a result and terminates. This also shows that the statement $a_i = \mathrm{ldterm}(u)$ is executed at most once for each i. Indeed, before $a_i = \mathrm{ldterm}(u)$ is executed, we have $a_i = a_{i+1} = \cdots = a_k = 0$. We shall use this fact later.

We now show, by induction, that $v = u + a_1 w_1 + \cdots + a_k w_k$ is a loop invariant for the while-loop of `Sift`.

(*Induction basis.*) When the while-loop of `Sift` first starts execution, we have $u = v$, and $a_j = 0$, for $j = 1, \ldots, k$. Thus, at this point, we have $u + a_1 w_1 + \cdots + a_k w_k = v + 0 = v$, as required.

(*Induction step.*) Suppose we are at the beginning of `S`. Let u' be the value of u at this

point, and suppose $v = u' + a_1w_1 + \cdots + a_kw_k$. When `S` is executed, first i is set to be the integer i such that $\text{ldindx}(w_i) = \text{ldindx}(u)$. We know from the discussion above that at this point we must have

$$a_i = a_{i+1} = \cdots = a_k = 0$$

so $v = u' + a_1w_1 + \cdots + a_{i-1}w_{i-1}$. Then $a_i := \text{ldterm}(u)$ is executed, assigning u the value $u' - a_iw_i$. Thus, at the end of `S`:

$$\begin{aligned} v &= u' + a_1w_1 + \cdots + a_{i-1}w_{i-1} \\ &= u' + a_1w_1 + \cdots + a_{i-1}w_{i-1} + a_iw_i - a_iw_i \\ &= u + a_1w_1 + \cdots + a_iw_i \\ &= u + a_1w_1 + \cdots + a_kw_k \quad (\text{since } a_{i+1} = \cdots = a_k = 0). \end{aligned}$$

This completes the induction. Thus, $v = u + a_1w_1 + \cdots + a_kw_k$ is indeed, a loop invariant, so that on termination of the while-loop, we know that $v = u + a_1w_1 + \cdots + a_kw_k$, and $\text{ldindx}(u)$ is not in $\{\text{ldindx}(w_1), \ldots, \text{ldindx}(w_k)\}$. Therefore `Sift` returns the correct output, as specified. ■

Theorem 28 *Let F be a field, $v \in F^n$, and $(w_1, \ldots, w_k)$ a sequence in echelon form of vectors in F^n. Suppose*

$$\texttt{Sift}(v, (w_1, \ldots, w_k)) = (u, (a_1, \ldots, a_k)).$$

Then the following holds

(i) $u \in F^n$, $a_1, \ldots, a_k \in F$, $v = u + a_1w_1 + \cdots a_kw_k$ *and* $\text{ldindx}(u) \notin \{\text{ldindx}(w_1), \ldots, \text{ldindx}(w_k)\}$.

(ii) $\langle v, w_1, \ldots, w_k\rangle = \langle u, w_1, \ldots, w_k\rangle$.

(iii) $v \in \langle w_1, \ldots, w_k\rangle \iff u = \underline{0}$.

(iv) $v \in \langle w_1, \ldots, w_k\rangle \iff v = a_1v_1 + \cdots a_kv_k$.

Proof

(i) See proof of correctness of `Sift`.

(ii) From part *(i)* $v = u + a_1w_1 + \cdots + a_kw_k$. Thus, $v \in \langle u, w_1, \ldots, w_k\rangle$, and it follows that $\langle v, w_1, \ldots, w_k\rangle \subseteq \langle u, w_1, \ldots, w_k\rangle$. On the other hand, $u = v - a_1w_1 - \cdots - a_kw_k$. Thus, $u \in \langle v, w_1, \ldots, w_k\rangle$, so $\langle u, w_1, \ldots, w_k\rangle \subseteq \langle v, w_1, \ldots, w_k\rangle$. Therefore $\langle u, w_1, \ldots, w_k\rangle = \langle v, w_1, \ldots, w_k\rangle$.

(iii) ($\Longrightarrow$) Suppose $v \in \langle w_1, \ldots, w_k\rangle$. Then, by part *(ii)*

$$\langle w_1, \ldots, w_k\rangle = \langle v, w_1, \ldots, w_k\rangle = \langle u, w_1, \ldots, w_k\rangle$$

and therefore $u \in \langle w_1, \ldots, w_k\rangle$, hence $u = b_1w_1 + \cdots + b_kw_k$, for some $b_1, \ldots, b_k \in F$. Suppose $u \neq \underline{0}$. Then not all of $b_1, \ldots, b_k$ are zero. Let l be the least element of $\{1, \ldots, k\}$ such that $b_l \neq 0$, so $u = b_lw_l + b_{l+1}w_{l+1} + \cdots b_kw_k$, $b_l \neq 0$.

Because $(w_1, \ldots, w_k)$ is in echelon form, we must have $\text{ldindx}(u) = \text{ldindx}(w_l)$.

But this contradicts $\text{ldindx}(u) \notin \{\text{ldindx}(w_1), \ldots, \text{ldindx}(w_k)\}$. Thus, the assumption that $u \neq \underline{0}$ leads to a contradiction, so we must have $u = \underline{0}$.

($\Longleftarrow$) If $u = \underline{0}$, then

$$\begin{aligned} v &= u + a_1 w_1 + \cdots + a_k w_k \\ &= \underline{0} + a_1 w_1 + \cdots + a_k w_k \in \langle w_1, \ldots, w_k \rangle. \end{aligned}$$

(*vi*) ($\Longrightarrow$) Suppose $v \in \langle w_1, \ldots, w_k \rangle$. Then $u = \underline{0}$, by (*iii*), and therefore

$$v = u + a_1 w_1 + \cdots + a_k w_k = a_1 w_1 + \cdots + a_k w_k.$$

($\Longleftarrow$) If $v = a_1 w_1 + \cdots + a_k w_k$, then v is a linear combination of $w_1, \ldots, w_k$, so $v \in \langle w_1, \ldots, w_k \rangle$. ■

Example 6.53. $F = \mathbb{Z}/(3)$.

$$\underset{v}{\texttt{Sift}((2,1,2), ((1,2,2), (0,0,1)))}$$
$$\qquad\qquad\quad w_1 \quad w_2$$

k	L	a_1	a_2	u	$\text{ldindx}(u) \in L$	i
2	$\{1,3\}$	0	0	$(2,1,2)$	T	1
		2		$(0,0,1)$	T	2
			1	$(0,0,0)$	F	

$$\texttt{return } ((0,0,0), (2,1))$$

Thus, $(2,1,2) \in \langle (1,2,2), (0,0,1) \rangle$ and $(2,1,2) = 2 \cdot (1,2,2) + 1 \cdot (0,0,1)$.

We conclude this section by presenting a variant of `Sift`

`Algorithm Sift2`
`INPUT`: v, W, such that $v \in F^n$, and $W = (w_1, \ldots, w_k)$
is a sequence in echelon form of k vectors in F^n.
`OUTPUT`: $u, (a_1, \ldots, a_k)$, such that $u \in F^n$, $a_1, \ldots, a_k \in F$,
$v = u + a_1 w_1 + \cdots + a_k w_k$,
and $u_i = 0$ for $i \in \{\text{ldindx}(w_1), \ldots, \text{ldindx}(w_k)\}$.

$k := \#W$;
$u := v$;
$i := 1$;
`while` $i \leq k$ `do`
  $j := \text{ldindx}(w_i)$;
  $a_i := u_j$;
  $u := u - a_i w_i$;

$i := i + 1$;
od;
return $(u, (a_1, \ldots, a_k))$;
end;

This algorithm makes use of all echelon vectors to maximize the number of zero elements of u. It can be shown that the condition on u is stronger than the corresponding condition in `Sift`, and that Theorem 28 holds for `Sift2` as well (see exercises). Note that `Sift2` does not require a matching function for the leading index.

6.2 Constructing an echelon basis

Let F be a field. Suppose we are given $w_1, w_2, \ldots, w_k \in F^n$, and we wish to study $\mathcal{W}$, where

$$\mathcal{W} = \langle w_1, w_2, \ldots, w_k \rangle.$$

Suppose we could determine $u_1, u_2, \ldots, u_j \in F^n$ such that (i) $(u_1, u_2, \ldots, u_j)$ is in echelon form, and (ii) $\langle u_1, u_2, \ldots, u_j \rangle = \mathcal{W}$.

Then, given $v \in F^n$, we could determine whether or not $v \in \mathcal{W}$, by using the algorithm `Sift` as follows. Suppose $(u, (a_1, \ldots, a_j)) =$ `Sift`$(v, (u_1, \ldots, u_j))$. Then, from Theorem 28

$$u = \underline{0} \iff v \in \langle u_1, \ldots, u_j \rangle = \mathcal{W}.$$

Some remarks are in order here:

(i) $(u_1, \ldots, u_j)$ in echelon form $\implies$ $(u_1, \ldots, u_j)$ is a linearly independent sequence of vectors (prove it!).

(ii) $(u_1, \ldots, u_j)$ a linearly independent sequence of vectors and $\langle u_1, \ldots, u_j \rangle = \mathcal{W} \implies$ $(u_1, \ldots, u_j)$ is an ordered basis for $\mathcal{W}$.

(iii) $(u_1, \ldots, u_j)$ is an ordered basis for $\mathcal{W} \implies \dim(\mathcal{W}) = j$.

Let F be a field. For the algorithm `Echelonize` below, we assume that we have algorithms for exact computation in F, which is certainly the case when $F = \mathbb{Q}$ or $\mathbb{Z}/(p)$, p prime.

`Algorithm Echelonize`
`INPUT`: $W = (w_1, \ldots, w_k)$, a sequence of k vectors in F^n.
`OUTPUT`: U, a sequence in echelon form of vectors in F^n,
such that $\langle U \rangle = \langle w_1, \ldots, w_k \rangle$.

$k := \#W$; (* compute the number of elements of W *)
$U := (\,)$; (* initialize U to the empty sequence *)
$i := 1$;

```
while i ≤ k do
   (u, A) := Sift(w_i, U);
   if u ≠ 0 then
      u := ldterm(u)^{-1} · u;
      'insert u into the correct place in U,
         so that U remains in echelon form';
   fi;
   i := i + 1;
od;
return U;
end;
```

Before proving the correctness of `Echelonize`, we make a few remarks. The input of `Echelonize` consists of a single object (the sequence W), rather than k objects (the elements of W). In particular, the number of elements of W must be computed explicitly as the cardinality of the input datum.

The statement 'insert u into the correct place in U, so that U remains in echelon form' is an *assignment statement.* To analyze its structure, we begin to rewrite it as

$U :=$ 'sort $U \,\&\, u$ by leading index, in ascending order';

where $U \,\&\, u$ is the sequence obtained by appending the vector u at the end of the sequence U, using the concatenation operator & (equation (2.1) page 19). The new sequence will not be in echelon form, in general.

This is a very simple instance of sorting, called *insertion sorting,* familiar, for instance, when sorting a hand of cards. We consider it in a slightly more general setting. Let A be a set, and let β be a boolean function on $A \times A$, with the properties that, for all $a, b, c \in A$

- $\beta(a, a)$ is TRUE
- $\beta(a, b)$ OR $\beta(b, a)$ is TRUE
- if $\beta(a, b)$ AND $\beta(b, c)$ is TRUE, so is $\beta(a, c)$.

We call such β, an *ordering relation* on A. We say that the sequence $S = (s_1, \ldots, s_k) \in A^k$ *sorted by* β, if the boolean expressions $\beta(s_i, s_{i+1})$ are TRUE for $i = 1, \ldots, k-1$.

```
Algorithm InsertSort
INPUT: S, a, β, where S ∈ A^k, a ∈ A,
       β is an ordering relation on A, and S is sorted by β.
OUTPUT: S', where S' is a permutation of S & a, which is sorted by β.
k := #S;
if β(a, S) then
   return a & S;
 fi;
```

```
i := 2;
while i ≤ k do
  if β(a, S) then
    return ((s_1, ..., s_{i-1}, a, s_i, ..., s_k));
  fi;
  i := i + 1;
od;
return S &a ;
end;
```

(A *permutation* of the elements of a sequence is a rearrangement of them.) Thus, $A = \mathbb{Z}$, and $\beta : (x, y) \mapsto x \leq y$, corresponds to sorting integers in ascending order. Returning to `Echelonize`, we have $A = F^n$, and the ordering relation

$$\beta : F^n \times F^n \rightarrow \{\text{TRUE, FALSE}\} \qquad (u, v) \mapsto \text{ldindx}(u) \leq \text{ldindx}(v)$$

satisfies the above conditions (because the integers do). With this β, we rewrite our assignment as

$U :=$ `InsertSort`(U, u, β);

which now features a more familiar syntax.

Proposition 29 *The algorithm* `Echelonize` *is correct.*

Proof: We first show that the boolean expression

$$\mathcal{L} := \text{'}U \text{ is in echelon form'} \quad \text{AND} \quad \langle U \rangle = \langle w_1, \ldots, w_{i-1} \rangle$$

is a loop invariant for the while-loop `W` of the algorithm `Echelonize`.

(*Induction basis.*) When `W` first starts executing, $i = 1$, and $U = (\,)$, which is in echelon form. Also $\langle U \rangle = \langle\,\rangle = \langle w_1, \ldots, w_0 \rangle$, as required.

(*Induction step.*) Suppose now that we are executing at the beginning of the statement-sequence of `W`, $i \leq k$, and statement $\mathcal{L}$ is true, and $U = (u_1, \ldots, u_j)$, for some $j \geq 0$ (If $j = 0$, then U is empty). Then U is in echelon form, and $\langle U \rangle = \langle u_1, \ldots, u_j \rangle = \langle w_1, \ldots, w_{i-1} \rangle$.

If $u = \underline{0}$, then $= \langle u_1, \ldots, u_j \rangle = \langle u_1, \ldots, u_j, u \rangle = \langle w_1, \ldots, w_i \rangle$.

If $u \neq \underline{0}$, then we first set $u := \text{ldterm}(u)^{-1} \cdot u$ and we still have $\langle u_1, \ldots, u_j, u \rangle = \langle w_1, \ldots, w_i \rangle$ (see exercises). Now, $\text{ldterm}(u) = 1$, and (due to the output specification of `Sift`) $\text{ldindx}(u)$ is not in the set $\{\text{ldindx}(u_1), \ldots, \text{ldindx}(u_j)\}$. Thus, we can (and do), insert u into the correct place in U, so that U remains in echelon form, but now contains $j + 1$ elements.

In either case, we now have U in echelon form, and $\langle U \rangle = \langle w_1, \ldots, w_i \rangle$. Thus, after setting $i := i + 1$, $\mathcal{L}$ holds, and we are finished executing the statement-sequence of `W`. This completes the proof that $\mathcal{L}$ is a loop invariant for `W`.

The rest of the proof that `Echelonize` works is straightforward. The statement-sequence of `W` is executed exactly k times, so the algorithm terminates. On termination of the execution of `W`, we have $i = k+1$ and $\mathcal{L}$ holds. Therefore, at that point, U is in echelon forms and $\langle U \rangle = \langle w_1, \ldots, w_{(k+1)-1} \rangle = \langle w_1, \ldots, w_k \rangle$ so the correct output is recorded. ■

6.3 An example

Let $F = \mathbb{Z}/(7)$ and

$$w_1 = (5,3,1,3), \qquad w_2 = (3,6,2,6), \qquad w_3 = (6,5,4,1), \qquad w_4 = (2,6,6,0) \quad \in F^4.$$

We wish to study the subspace $\mathcal{W} = \langle w_1, w_2, w_3, w_4 \rangle$ of F^4, and determine whether or not a given vector belongs to it.

We first use the algorithm `Echelonize` to determine a sequence of vectors $(u_1, \ldots, u_j)$ in echelon form, such that $\langle u_1, \ldots, u_j \rangle = \mathcal{W}$.

$$\underset{}{\texttt{Echelonize}}\underset{\quad w_1 \qquad w_2 \qquad w_3 \qquad w_4}{((5,3,1,3),(3,6,2,6),(6,5,4,1),(2,6,6,0)))}$$

k	U	i	$i \leq k$	`Sift`(w_i, U)	u	$u \neq 0$
4	( )	1	T	((5,3,1,3),( ))	(5,3,1,3)	T
					(1,2,3,2)	
	((1,2,3,2))	2	T	((0,0,0,0),(3))	(0,0,0,0)	F
		3	T	((0,0,0,3),(6))	(0,0,0,3)	T
					(0,0,0,1)	
	((1,2,3,2), (0,0,0,1))	4	T	((0,2,0,3),(2,0))	(0,2,0,3)	T
					(0,1,0,5)	
	((1,2,3,2), (0,1,0,5), (0,0,0,1))	5	F			

`return` $((1,2,3,2),(0,1,0,5),(0,0,0,1))$

$$\underset{}{\texttt{Sift}}\underset{v \qquad W}{((5,3,1,3),\ (\,))}$$

k	L	u	ldindx$(u) \in L$	i
0	{ }	(5,3,1,3)	F	

`return` $((5,3,1,3),(\,))$

Note that there are no columns for the a_j in the tracing of `Sift`, because when $k = 0$, the assignment statement $a_j := 0;, j = 1, \ldots, k$ is empty.

$$\underset{}{\texttt{Sift}}\underset{v \qquad w_1}{((3,6,2,6),((1,2,3,2)))}$$

k	L	a_1	u	ldindx$(u) \in L$	i
1	$\{1\}$	0	$(3,6,2,6)$	T	1
		3	$(0,0,0,0)$	F	

`return` $((0,0,0,0),(3))$

`Sift`$((6,5,4,1),((1,2,3,2)))$
v w_1

k	L	a_1	u	ldindx$(u) \in L$	i
1	$\{1\}$	0	$(6,5,4,1)$	T	1
		6	$(0,0,0,3)$	F	

`return` $((0,0,0,3),(6))$

`Sift`$((2,6,6,0),\ ((1,2,3,2),(0,0,0,1)))$
v w_1 w_2

k	L	a_1	a_2	u	ldindx$(u) \in L$	i
2	$\{1,4\}$	0	0	$(2,6,6,0)$	T	1
		2		$(0,2,0,3)$	F	

`return` $((0,2,0,3),(2,0))$

Thus,

$$\langle (1,2,3,2),\ (0,1,0,5),\ (0,0,0,1)\rangle = \mathcal{W} = \langle w_1, w_2, w_3, w_4\rangle.$$

So $\dim(\mathcal{W}) = 3$. Is $(6,0,4,4) \in \mathcal{W}$? To answer this question we compute

`Sift`$((6,0,4,4,),((1,2,3,2),(0,1,0,5),(0,0,0,1)))$
v w_1 w_2 w_3

k	L	a_1	a_2	a_3	u	ldindx$(u) \in L$	i
3	$\{1,2,4\}$	0	0	0	$(6,0,4,4)$	T	1
		6			$(0,2,0,6)$	T	2
			2		$(0,0,0,3)$	T	3
				3	$(0,0,0,0)$	F	

`return` $((0,0,0,0),(6,2,3))$

Since the first element of the output sequence is $(0,0,0,0)$, we have that $(6,0,4,4) \in \mathcal{W}$. Next, we determine whether $(2,4,1,5)$ is in $\mathcal{W}$.

`Sift`$((2,4,1,5),\ ((1,2,3,2),(0,1,0,5),(0,0,0,1)))$
v w_1 w_2 w_3

k	L	a_1	a_2	a_3	u	ldindx$(u) \in L$	i
3	$\{1,2,4\}$	0	0	0	$(2,4,1,5)$	T	1
		2			$(0,0,2,1)$	F	

`return` $((0,0,2,1),(2,0,0))$

Since $(0,0,2,1) \neq (0,0,0,0)$ we conclude that $(2,4,1,5) \notin \mathcal{W}$.

6.4 Testing subspaces

Let F be a field, and

$$v_1,\ldots,v_l,w_1,\ldots,w_k \in F^n, \qquad \mathcal{V} = \langle v_1,\ldots,v_l\rangle, \qquad \mathcal{W} = \langle w_1,\ldots,w_k\rangle.$$

How can we determine whether or not $\mathcal{V} \subseteq \mathcal{W}$?

We proceed as follows. First, let $U :=$ `Echelonize`$((w_1,\ldots,w_k))$. Then U is in echelon form, and $\langle U\rangle = \mathcal{W}$. Suppose $U = (u_1,\ldots,u_j)$. Now $\mathcal{V} \subseteq \mathcal{W}$ iff $v_i \in \mathcal{W}$ for each $i = 1,\ldots,l$. Thus, $\mathcal{V} \subseteq \mathcal{W}$ iff `Sift`$(v_i, (u_1,\ldots,u_j))$ returns $(\underline{0}, A_i)$, for each $i = 1,\ldots,l$. (We do not care what is A_i for this application.)

Thus, from our previous example (section 6.3), we conclude that

$$\langle (6,0,4,4)\rangle \subseteq \mathcal{W} \qquad \langle (6,0,4,4),(2,4,1,5)\rangle \not\subseteq \mathcal{W}.$$

We can also test whether or not $\mathcal{V} = \mathcal{W}$, since $\mathcal{V} = \mathcal{W}$ iff $\mathcal{V} \subseteq \mathcal{W}$ and $\mathcal{W} \subseteq \mathcal{V}$. Alternatively, $\mathcal{V} = \mathcal{W}$ iff $\mathcal{V} \subseteq \mathcal{W}$ and $\dim(\mathcal{V}) = \dim(\mathcal{W})$.

Exercises

Exercise 6.1. Let $F = \mathbb{Z}/(2)$.

(a) Write down all the elements of F^3.

(b) Write down all the sequences in echelon form of elements of F^2.

Exercise 6.2. Let $w_1 = (1,1,0)$, $w_2 = (2,2,2)$ be vectors in $(\mathbb{Z}/(3))^3$, and let $W = \langle w_1, w_2\rangle$.

(a) Write down all the elements of W.

(b) For each $w \in W$, determine ldindx(w), and if $w \neq 0$, determine ldterm(w).

Exercise 6.3. Let $v, w_1, w_2, w_3 \in \mathbb{Q}^3$, with

$$v = (4/7, -3/5, 5/7); \quad w_1 = (1, 1/2, 1), \quad w_2 = (0,1,2), \quad w_3 = (0,0,1).$$

(a) Trace `Sift`$(v, (w_1, w_2, w_3))$.

(b) Write v as a linear combination of w_1, w_2, w_3, verifying your calculation explicitly.

Exercise 6.4. Let $F = \mathbb{Z}/(11)$, and let

$$w_1 = (1, 0, 7, 3) \qquad w_2 = (0, 1, 3, 4) \qquad w_3 = (0, 0, 0, 1) \in F^4.$$

(*a*) Use the algorithm `Sift` to prove that $(5, 2, 4, 3) \notin \langle w_1, w_2, w_3 \rangle$.

(*b*) Use the algorithm `Sift` to prove that $v = (6, 2, 4, 3) \in \langle w_1, w_2, w_3 \rangle$. Hence write v as a linear combination of w_1, w_2, w_3, verifying your calculation explicitly.

Exercise 6.5. Let $F = \mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, or $\mathbb{Z}/(p)$, p a prime (or indeed let F be any field), and let $v_1, \ldots, v_k \in F^n$.

(*a*) Let $0 \neq \alpha \in F$. Prove that

$$\langle v_1, \ldots, v_{k-1}, \alpha v_k \rangle = \langle v_1, \ldots, v_k \rangle.$$

(*b*) Prove that if the sequence $(v_1, \ldots, v_k)$ is in echelon form, then $v_1, \ldots, v_k$ are linearly independent.

(*c*) Let $F = \mathbb{Z}/(p)$, p a prime. Using `Echelonize`, prove that if $\mathcal{W}$ is the subspace of F^n generated by the vectors $W = (w_1, \ldots, w_m)$, then $\#\mathcal{W} = p^s$, for some s, with $0 \leq s \leq n$. Explain what is s.

Exercise 6.6. Let $F = \mathbb{Z}/(2)$, and let $v_1 = (0, 1, 0, 1)$, $v_2 = (0, 0, 0, 0)$ $v_3 = (1, 1, 0, 1)$, $v_4 = (1, 0, 1, 0)$, $v_5 = (0, 1, 1, 1)$ be in F^4. Use the algorithm `Echelonize` to determine a sequence U, in echelon form, of vectors in F^4, such that $\langle U \rangle = \langle v_1, \ldots, v_5 \rangle$.

Exercise 6.7. Let $F = \mathbb{Z}/(5)$, and let

$$v_1 = (4, 3, 1, 3) \qquad v_2 = (1, 3, 4, 1) \qquad v_3 = (2, 3, 3, 2) \in F^4.$$

(*a*) Use the algorithm `Echelonize` to determine a sequence U, in echelon form, of vectors in F^4, such that $\langle U \rangle = \langle v_1, v_2, v_3 \rangle$.

(*b*) Let $\mathcal{V} = \langle v_1, v_2, v_3 \rangle$.
Which of the following vectors belong to $\mathcal{V}$?

$$(2, 1, 3, 1) \qquad (3, 1, 4, 2) \qquad (2, 2, 4, 3).$$

In each case, explain why.

Exercise 6.8.

(*a*) Let A be any set. Write an algorithm to the following specifications

`Algorithm Match`

`INPUT`: a, S where $a \in A$ and $S = (s_1, \ldots, s_n) \in A^n$

`OUTPUT`: i, where i is the smallest integer such that $a = s_i$,

if such i exist, or the empty sequence otherwise.

(*b*) Using `Match`, write the algorithm `MatchLeadingIndices`, introduced at page 71 in connection with the algorithm `Sift`.

Exercise 6.9. Prove that the algorithm `InsertSort` is correct. (This algorithm was introduced at page 76 in connection with the algorithm `Echelonize`.)

Exercise 6.10. Write an algorithm to the following specifications

`Algorithm Echelon`

`INPUT`: W, a finite sequence of n-dimensional vectors, over the same field.

`OUTPUT`: TRUE, if W is in echelon form, FALSE otherwise.

Explain what you are doing. Use the notation $W = (W_1, W_2, \ldots)$, and $W_k = (W_{k,1}, W_{k,2}, \ldots)$, and the operator # to access input data. Assume that the algorithm `ldindx` and `ldterm` are available. Decide how `ldterm` behaves for the zero vector, and design the algorithm accordingly.

Exercise 6.11. Write an algorithm to the following specifications

`Algorithm Subspace`

`INPUT`: V, W, where V and W are finite sequences of
n-dimensional vectors over the same field.

`OUTPUT`: TRUE, if $\langle V \rangle \subseteq \langle W \rangle$, FALSE otherwise.

Exercise 6.12. Prove that theorem 28 holds also for the algorithm `Sift2`.

Chapter 7

Some Proofs*

7.1 A note on ring theory

Let R be a commutative ring with identity, and $a, b, \in R$. In our proofs we have been using the fact that

(i) $a0 = 0$

(ii) $(-a)b = -(ab)$.

We now prove these facts using the definition of a commutative ring with identity. (As usual $a - b$ means $a + (-b)$.)

Proof of (i). We have $a0 = a(0 + 0) = a0 + a0$ (distributive law). Thus, $a0 - (a0) = (a0 + a0) - (a0) = a0 + (a0 - (a0))$ (associative law), hence $0 = a0 + 0 = a0$.

Proof of (ii). We have, using the distributive law $ab + (-a)b = (a - a)b = 0b = 0$, the last step deriving from (i). Thus, $(-a)b = -(ab)$, since additive inverses are unique in R, for if $s, t \in R$ and $a + s = a + t = 0$ then

$$s + a + s = s + a + t \quad \Longrightarrow \quad 0 + s = 0 + t \quad \Longrightarrow \quad s = t.$$

[Note also that additive identities are unique in R, as are multiplicative identities, for if $r, s \in R$, $0 + r = 0 \Longrightarrow r = 0$ and $1s = 1 \Longrightarrow s = 1$.]

7.2 Uniqueness of quotient and remainder

We prove the uniqueness of $a \,\mathrm{DIV}\, b$ and $a \,\mathrm{MOD}\, b$ for integers and polynomials.

Theorem 30 *Suppose $a, b, \in \mathbb{Z}$, $b \neq 0$, and $a = bq_1 + r_1 = bq_2 + r_2$ such that $q_1, q_2, r_1, r_2 \in \mathbb{Z}$ and $0 \leq r_1, r_2 < |b|$. Then $q_1 = q_2$ and $r_1 = r_2$.*

Proof: From the fact that $bq_1 + r_1 = bq_2 + r_2$ we have that $b(q_1 - q_2) = r_2 - r_1$.

Case (i): $q_1 = q_2$. Then $b(q_1 - q_2) = 0$, hence $r_1 - r_2 = 0$ and $r_1 = r_2$.

Case (ii): $q_1 \neq q_2$. Then

$$|r_2 - r_1| = |b(q_1 = q_2)| = |b|\,|q_1 - q_2| \geq |b|.$$

However, $0 \leq r_1, r_2 < |b|$, hence $|r_1 - r_2| < |b|$, a contradiction. Thus, $q_1 = q_2$, so, by case (i), $r_1 - r_2$. ■

Theorem 31 *Let R be a commutative ring with identity, $a, b, \in R[x]$, $b \neq 0$, and* $\mathrm{ldcf}(b)$ *invertible. Suppose $a = bq+r_1 = bq+r_2$ such that $q_1, q_2, r_1, r_2 \in R[x]$ and* $\deg(r_1), \deg(r_2) < \deg(b)$. *Then $q_1 = q_2$ and $r_1 = r_2$.*

Proof: As in the proof of the previous theorem, we have that $b(q_1 - q_2) = r_2 - r_1$.

Case (i): $q_1 = q_2$. Then $b(q_1 - q_2) = 0$, hence $r_1 - r_2 = 0$ and $r_1 = r_2$.

Case (ii): $q_1 \neq q_2$. Then

$$\deg(r_2 - r_1) = \deg(b(q_1 = q_2)) = \deg(b) + \deg(q_1 - q_2) \geq \deg(b)$$

where the first equality follows from the fact that $\mathrm{ldcf}(b)$ is invertible and $q_1 - q_2 = 0$.

But $\deg(r_2 - r_1) < \deg(b)$, a contraction. Thus, $q_1 = q_2$, so, by case (i), $r_1 - r_2$. ■

Chapter 8

Hints for exercises

Chapter 1

Exercise 1.3. Find a counterexample (you do not have to look too far).

Exercise 1.4. Part (f): 47 is the sum of two squares if and only if, for some square $x^2 \leq 47$, we have that $47 - x^2$ is also a square (necessarily in the same range).

Exercise 1.9. Part (b): there is only one thing that can go wrong.

Exercise 1.12. First do exercise 1.4 (f). You may find it convenient to build the sequence of squares not exceeding x.

Chapter 2

Exercise 2.2. Part (c): remember that $a \,\mathrm{MOD}\, b = r$ means $a = qb + r$, for some q. How do you represent an odd integer?
Part (d): treat the case z even and odd separately.

Exercise 2.11. Part (c): do you need to test even integers?

Exercise 2.9. The difficulty lies in turning the sentence 'but not by both' into a boolean expression. What boolean operator corresponds to 'but'?

Exercise 2.10. Part (b): the output of `IntegerFactorization` is a finite sequence P; denote its cardinality by $\#P$, and its elements by P_i, $i = 1, \ldots, \#P$. Treat the case $n = 1$ separately.

Exercise 2.15. Part (b): represent n in base 2.

Chapter 3

Exercise 3.5. Analyze the possible partitions of a set with 5 elements, and for each count the number of elements of the corresponding equivalence relation.

Exercise 3.7. Measure efficiency by the number of evaluations of the function P. How many evaluations are involved in the above formula? How could such number be reduced? Look at concrete examples involving small sets X.

Exercise 3.8. The formulation of this exercise is deliberately cryptic: give yourself time to think about it. You'll need a nested loop.

Chapter 4

Exercise 4.1. Part (b): use Theorem 12, page 40, and induction.

Exercise 4.8. Part (b): examine some examples, then infer from them the general phenomenon. The essay should contain no reference to specific examples.

Chapter 5

Exercise 5.3. Part (b): see how the loop invariant for the while-loop of algorithm `PolynomialQuoRem` was proved.

Exercise 5.4. Part (a) when analyzing the effect of the statement-sequence of `W`, consider the cases b even and b odd separately.

Chapter 6

Exercise 6.5. Part (a): show that the left-hand side is contained in the rigth-hand side, and vice-versa.
Part (b): this is linear algebra.
Part (c): use the result of the previous problem..

Exercise 6.10. The problem is that if $\mathrm{ldterm}(\underline{0}) = ()$, then expressions of the type $() = 1$ make no sense.

Index