Operational notes

Document updated on **August 8, 2022**.

The following colors are **not** part of the final product, but serve as highlights in the editing/review process:

- text that needs attention from the Subject Matter Experts: Mirco, Anna,& Jan
- terms that have not yet been defined in the book
- things that need to be checked only at the very final typesetting stage (and it doesn't make sense to do them before)
- text that needs advice from the communications/marketing team: Aaron & Shane
- text that needs to be completed or otherwise edited (by Sylvia)

NB: This PDF only includes the following chapter(s): Arithmetics.

Todo list

MoonMath manual

TechnoBob and the Least Scruples crew

August 8, 2022

Contents

Chapter 3

Arithmetics

S: This chapter talks about different types of arithmetic, so I suggest using “Arithmetics” as the chapter title.

Pluralize chapter title

3.1 Introduction

The goal of this chapter is to bring a reader with only basic school-level algebra up to speed in arithmetics. We start with a brief recapitulation of basic integer arithmetics, discussing long division, the greatest common divisor and Euclidean Division. After that, we introduce modular arithmetics as **the most important** skill to compute our pen-and-paper examples. We then introduce polynomials, compute their analogs to integer arithmetics and introduce the important concept of Lagrange interpolation.

check if this is already introduced in intro

3.2 Integer arithmetic

In a sense, integer arithmetic is at the heart of large parts of modern cryptography. Fortunately, most readers will probably remember integer arithmetic from school. It is, however, important that you can confidently apply those concepts to understand and execute computations in the many pen-and-paper examples that form an integral part of the MoonMath Manual. We will therefore recapitulate basic arithmetic concepts to refresh your memory and fill any knowledge gaps.

unify addressing the reader

formality of addressing the reader

Even though the terms and concepts in this chapter might not appear in the literature on zero-knowledge proofs directly, understanding them is necessary to follow subsequent chapters and beyond: terms like **groups** or **fields** also crop up very frequently in academic papers on zero-knowledge cryptography.

3.2.1 Integers, natural numbers and rational numbers

Integers are also known as **whole numbers**, that is, numbers that can be written without fractional parts. Examples of numbers that are **not** integers are $\frac{2}{3}$, 1.2 and -1280.006.

Throughout this book, we use the symbol $\mathbb{Z}$ as a shorthand for the set of all **integers**:

$$\mathbb{Z} := \{\ldots,-3,-2,-1,0,1,2,3,\ldots\} \tag{3.1}$$

If $a \in \mathbb{Z}$ is an integer, then we write $|a|$ for the **absolute value** of a, that is, the the non-negative value of a without regard to its sign:

$$|4| = 4 \tag{3.2}$$

$$|-4| = 4 \tag{3.3}$$

We use the symbol $\mathbb{N}$ for the set of all positive integers, usually called the set of **natural numbers**. Furthermore, we use $\mathbb{N}_0$ for the set of all non-negative integers. This means that $\mathbb{N}$ does not contain the number 0, while $\mathbb{N}_0$ does:

$$\mathbb{N} := \{1,2,3,\dots\} \qquad\qquad \mathbb{N}_0 := \{0,1,2,3,\dots\}$$

SB: Talking about the binary representation seems way to complex at this stage, and the concepts introduced here are not used for several chapters. Let $n \in \mathbb{N}_0$ be a non-negative integer and $(b_0, b_1, \dots b_k)$ a string of **bits** $b_j \in \{0,1\} \subset \mathbb{N}_0$ for some non negative integer $k \in \mathbb{N}$, such that the following equation holds:

Move content on binary representation

$$n = \sum_{j=0}^{k} b_j \cdot 2^j \tag{3.4}$$

In this case, we call $Bits(n) :=< b_0, b_1, \dots b_k >$ the **binary representation** of n, say that n is a k-bit number and call $k := |n|_2$ the **bit length** of n. It can be shown, that the binary representation of any non negative integer is unique. We call b_0 the **least significant bit** and b_k the **most significant** bit and define the **Hamming weight** of an integer as the number of 1s in its binary representation.

In addition, we use the symbol $\mathbb{Q}$ for the set of all **rational numbers**, which can be represented as the set of all fractions $\frac{n}{m}$, where $n \in \mathbb{Z}$ is an integer and $m \in \mathbb{N}$ is a natural number, such that there is no other fraction $\frac{n'}{m'}$ and natural number $k \in \mathbb{N}$ with $k \neq 1$ and

$$\frac{n}{m} = \frac{k \cdot n'}{k \cdot m'} \tag{3.5}$$

The sets $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{Q}$ have a notion of addition and multiplication defined on them. Most of us are probably able to do many integer computations in our head, but this gets more and more difficult as these increase in complexity. We will frequently invoke the SageMath system (2.7.1) for more complicated computations (we define rings and fields later in this book):SB: I would delete lines 12-18 form the Sage example below, unnecessarily confusing at this point

simplify Sage ex.

```
1  sage: ZZ #  Sage notation for the set of integers
2  Integer Ring
3  sage: NN # Sage notation for the set of natural numbers
4  Non negative integer semiring
5  sage: QQ # Sage notation for the set of rational numbers
6  Rational Field
7  sage: ZZ(5) # Get an element from the set of integers
8  5
9  sage: ZZ(5) + ZZ(3)
10 8
11 sage: ZZ(5) * NN(3)
```

```
15
sage: ZZ.random_element(10**50)
13724561743067967200720145809981686812557288427 93
sage: ZZ(27713).str(2) # Binary string representation
110110001000001
sage: NN(27713).str(2) # Binary string representation
110110001000001
sage: ZZ(27713).str(16) # Hexadecimal string representation
6c41
```

A set of numbers of particular interest to us is the set of **prime numbers**, which are natural numbers $p \in \mathbb{N}$ with $p \geq 2$ that are only divisible by themself and by 1. All prime numbers apart from the number 2 are called **odd** prime numbers. We use $\mathbb{P}$ for the set of all prime numbers and $\mathbb{P}_{\geq 3}$ for the set of all odd prime numbers. The set of prime numbers $\mathbb{P}$ is an infinite set, and it can be ordered according to size. This means that, for any prime number $p \in \mathbb{P}$, one can always find another prime number $p' \in \mathbb{P}$ with $p < p'$. Consequently, there is no largest prime number. Since prime numbers can be ordered by size, we can write them as follows:

$$2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,\ldots \tag{3.6}$$

As the **fundamental theorem of arithmetic** tells us, prime numbers are, in a certain sense, the basic building blocks from which all other natural numbers are composed. To see that, let $n \in \mathbb{N}$ be any natural number with $n > 1$. Then there are always prime numbers $p_1, p_2, \ldots, p_k \in \mathbb{P}$, such that the following equation hold:

To see that

$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_k \tag{3.7}$$

This representation is unique for each natural number (except for the order of the **factors** $p_1, p_2, \ldots, p_k$) and is called the **prime factorization** of n.

Example 1 (Prime Factorization). To see what we mean by the prime factorization of a number, let's look at the number $504 \in \mathbb{N}$. To get its prime factors, we can successively divide it by all prime numbers in ascending order starting with 2:

let's

$$504 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 7$$

We can double check our findings invoking Sage, which provides an algorithm for factoring natural numbers:

```
sage: n = NN(504)
sage: factor(n)
2^3 * 3^2 * 7
```

The computation from the previous example reveals an important observation: computing the factorization of an integer is computationally expensive, because we have to divide repeatedly by all prime numbers smaller than the number itself until all factors are prime numbers themself. From this, an important question arises: how fast can we compute the prime factorization of a natural number? This question is the famous **integer factorization problem** and, as far as we know, there is currently no known method that can factor integers much faster then the naive approach of just dividing the given number by all prime numbers in ascending order.

“themselves” is more common?

On the other hand, computing the product of a given set of prime numbers is fast: you just multiply all factors. This simple observation implies that the two processes “prime number

you

multiplication" on the one side and its inverse process "natural number factorization" have very different computational costs. The factorization problem is therefore an example of a so-called **one-way function**: an invertible function that is easy to compute in one direction, but hard to compute in the other direction. [1]

a bit overused in the text?

Exercise 1. What is the absolute value of the integers -123, 27 and 0?

Exercise 2. Compute the factorization of 30030 and double check your results using Sage.

Exercise 3. Consider the following equation:

$$4 \cdot x + 21 = 5.$$

Compute the set of all solutions for x under the following alternative assumptions:

1. The equation is defined over the set of natural numbers.
2. The equation is defined over the set of integers.

Exercise 4. Consider the following equation:

$$2x^3 - x^2 - 2x = -1.$$

Compute the set of all solutions x under the following assumptions:

1. The equation is defined over the set ofnatural numbers.
2. The equation is defined over the set of integers.
3. The equation is defined over the set of rational numbers.

3.2.2 Euclidean Division

As we know from high school mathematics, integers can be added, subtracted and multiplied, and the of these operations result is guaranteed to always be an integer as well. On the contrary, division (in the commonly understood sense) is not defined for integers, as, for example, 7 divided by 3 will not result in an integer. However, it is always possible to divide any two integers if we consider **division with a remainder**. For example, 7 divided by 3 is equal to 2 with a remainder of 1, since $7 = 2 \cdot 3 + 1$.

This section introduces division with a remainder for integers, usually called **Euclidean Division**. It is an essential technique underlying many concepts in this book. The precise definition is as follows:

Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be two integers with $b \neq 0$. Then there is always another integer $m \in \mathbb{Z}$ and a natural number $r \in \mathbb{N}$, with $0 \leq r < |b|$ such that the following holds:

$$a = m \cdot b + r \tag{3.8}$$

This decomposition of a given b is called **Euclidean Division**, where a is called the **dividend**, b is called the **divisor**, m is called the **quotient** and r is called the **remainder**. It can be shown that both the quotient and the remainder always exist and are unique, as long as the divisor is different from 0.

[1] It should be pointed out, however, that the American mathematician Peter W. Shor developed an algorithm in 1994, which can calculate the prime factorization of a natural number in polynomial time on a quantum computer. The consequence of this is that cryptosystems, which are based on the prime factor problem, are unsafe as soon as practically usable quantum computers become available.

Notation and Symbols 1. Suppose that the numbers a, b, m and r satisfy equation (3.8). We can then describe the quotient and the remainder of the Euclidean Division as follows:

$$a \text{ div } b := m, \qquad a \text{ mod } b := r \tag{3.9}$$

We also say that an integer a is **divisible** by another integer b if $a \text{ mod } b = 0$ holds. In this case, we write $b|a$, and call the integer $a \text{ div } b$ the **cofactor** of b in a.

So, in a nutshell, Euclidean Division is the process of dividing one integer by another in a way that produces a quotient and a non-negative remainder, the latter of which is smaller than the absolute value of the divisor.

Example 2. Applying Euclidean Division and the notation defined in 3.9 to the dividend -17 and the divisor 4, we get the following:

$$-17 \text{ div } 4 = -5, \qquad -17 \text{ mod } 4 = 3 \tag{3.10}$$

$-17 = -5 \cdot 4 + 3$ is the Euclidean Division of -17 by 4. The remainder, by definition, is a non-negative number. In this case, 4 does not divide -17, as the remainder is not zero. The truth value of the expression $4|-17$ therefore is FALSE. On the other hand, the truth value of $4|12$ is TRUE, since 4 divides 12, as $12 \text{ mod } 4 = 0$. If we invoke Sage to do the computation for us, we get the following:

```
sage: ZZ(-17) // ZZ(4) # Integer quotient
-5
sage: ZZ(-17) % ZZ(4) # remainder
3
sage: ZZ(4).divides(ZZ(-17)) # self divides other
False
sage: ZZ(4).divides(ZZ(12))
True
```

Remark 1. In 3.9, we defined the notation of a **div** b and a **mod** b in terms of Euclidean Division. It should be noted, however, that many programing languages (like Python and Sage) implement both the operator $(/)$ amd the operator $(\%)$ differently. Programers should be aware of this, as the discrepancy between the mathematical notation and the implementation in programing languages might become the source of subtle bugs in implementations of cryptographic primitives.

To give an example, consider the the dividend -17 and the divisor -4. Note that, in contrast to the previous example 2, we now have a negative divisor. According to our definition we have the following:

$$-17 \text{ div } -4 = 5, \qquad -17 \text{ mod } -4 = 3 \tag{3.11}$$

$-17 = 5 \cdot (-4) + 3$ is the Euclidean Division of -17 by -4 (the remainder is, by definition, a non-negative number). However, using the operators $(/)$ and $(\%)$ in Sage, we get a different result:

```
sage: ZZ(-17) // ZZ(-4) # Integer quotient
4
sage: ZZ(-17) % ZZ(-4) # remainder
-1
sage: ZZ(-17).quo_rem(ZZ(-4)) # not Euclidean Division
```

```
(4, -1)
```

Methods to compute Euclidean Division for integers are called **integer division algorithms**. Probably the best known algorithm is the so-called **long division**, which most of us might have learned in school.

In a nutshell, the long division algorithm loops through the digits of the dividend from the left to right, subtracting the largest possible multiple of the divisor (at the digit level) at each stage. The multiples then become the digits of the quotient, and the remainder is the first digit of the dividend.

As long division is the standard method used for pen-and-paper division of multi-digit numbers expressed in decimal notation, we use it throughout this book when we do simple pen-and-paper computations, so readers should become familiar with it. However, instead of defining the algorithm formally, we provide some examples instead, as this will hopefully make the process more clear.

readers

Example 3 (Integer Long Division). To give an example of integer long division algorithm, let's divide the integer $a = 143785$ by the number $b = 17$. Our goal is therefore to find solutions to equation 3.8, that is, we need to find the quotient $m \in \mathbb{Z}$ and the remainder $r \in \mathbb{N}$ such that $143785 = m \cdot 17 + r$. Using a notation that is mostly used in Commonwealth countries, we compute as follows:SB: I think a more detailed explanation is needed for those unfamiliar with this notation/algorithm

Commonwealth countries

Add explanation

$$
\begin{array}{r}
8457 \\
17\,\overline{)\,143785} \\
\underline{136} \\
77 \\
\underline{68} \\
98 \\
\underline{85} \\
135 \\
\underline{119} \\
16
\end{array}
\tag{3.12}
$$

We calculated $m = 8457$ and $r = 16$, and, indeed, the equation $143785 = 8457 \cdot 17 + 16$ holds. We can double check this invoking Sage:

```
sage: ZZ(143785).quo_rem(ZZ(17))
(8457, 16)
sage: ZZ(143785) == ZZ(8457)*ZZ(17) + ZZ(16) # check
True
```

Exercise 5 (Integer Long Division). Find an $m \in \mathbb{Z}$ and an $r \in \mathbb{N}$ with $0 \leq r < |b|$ such that $a = m \cdot b + r$ holds for the following pairs:

- $(a,b) = (27,5)$
- $(a,b) = (27,-5)$
- $(a,b) = (127,0)$
- $(a,b) = (-1687,11)$
- $(a,b) = (0,7)$

In which cases are your solutions unique?

Exercise 6 (Long Division Algorithm). Using the programming language of your choice, write an algorithm that computes integer long division and handles all edge cases properly.

Exercise 7 (Binary Representation). Using the programming language of your choice, write an algorithm that computes the binary representation 3.4 of any non-negative integer.

3.2.3 The Extended Euclidean Algorithm

One of the most critical parts of this book is the modular arithmetic, defined in section 3.3, and its application in the computations of **prime fields**, defined in section4.3.1. To be able to do computations in modular arithmetic, we have to get familiar with the so-called **Extended Euclidean Algorithm**, used to calculate the **greatest common divisor** (GCD) of integers.

The greatest common divisor of two non-zero integers a and b is defined as the largest non-zero natural number d such that d divides both a and b, that is, $d|a$ as well as $d|b$ are true. We use the notation $gcd(a,b) := d$ for this number. Since the natural number 1 divides any other integer, 1 is always a common divisor of any two non-zero integers, but it is not necessarily the greatest.

A common method for computing the greatest common divisor is the so-called Euclidean Algorithm. However, since we don't need that algorithm in this book, we will introduce the Extended Euclidean Algorithm, which is a method for calculating the greatest common divisor of two natural numbers a and $b \in \mathbb{N}$, as well as two additional integers $s,t \in \mathbb{Z}$, such that the following equation holds:

$$gcd(a,b) = s \cdot a + t \cdot b \tag{3.13}$$

The pseudocode in algorithm 1 shows in detail how to calculate the greatest common divisor and the numbers s and t with the Extended Euclidean Algorithm:In example 4, the computation stops when $r_k = 0$ (at k_4), not when $r_{k-1} = 0$ (which would be k_5). Also the GCD is $r_3 = r_{k-1} = 1$, not r_{k-2}. Same for s and t.

unify with example 4

Algorithm 1 Extended Euclidean Algorithm

Require: $a,b \in \mathbb{N}$ with $a \geq b$
  procedure EXT-EUCLID(a,b)
    $r_0 \leftarrow a$ and $r_1 \leftarrow b$
    $s_0 \leftarrow 1$ and $s_1 \leftarrow 0$
    $t_0 \leftarrow 0$ and $t_1 \leftarrow 1$
    $k \leftarrow 2$
    while $r_{k-1} \neq 0$ **do**
      $q_k \leftarrow r_{k-2}$ div r_{k-1}
      $r_k \leftarrow r_{k-2}$ mod r_{k-1}
      $s_k \leftarrow s_{k-2} - q_k \cdot s_{k-1}$
      $t_k \leftarrow t_{k-2} - q_k \cdot t_{k-1}$
      $k \leftarrow k+1$
    end while
    return $gcd(a,b) \leftarrow r_{k-2}$, $s \leftarrow s_{k-2}$ and $t \leftarrow t_{k-2}$
  end procedure
Ensure: $gcd(a,b) = s \cdot a + t \cdot b$

The algorithm is simple enough to be used effectively in pen-and-paper examples. It is commonly written as a table where where the rows represent the while-loop and the columns represent the values of the the array r, s and t with index k. The following example provides a simple execution.

Example 4. To illustrate algorithm 1, we apply it to the numbers $a = 12$ and $b = 5$. Since $12, 5 \in \mathbb{N}$ and $12 \geq 5$, all requirements are met, and we compute as follows: check if extended table is understandable

check additional explanation

check if extended table is understandable

k	r_k	s_k	t_k
0	12	1	0
1	5	0	1
2	2	1	-2
3	1	-2	5
4	0		

k	r_k	s_k	t_k	q_k				
0	12	1	0	–	$r_0 \leftarrow a = 12$	$s_0 \leftarrow 1$	$t_0 \leftarrow 0$	
1	5	0	1	–	$r_1 \leftarrow b = 5$	$s_1 \leftarrow 0$	$t_1 \leftarrow 1$	
2	2	1	-2	2	$r_2 \leftarrow r_0 \text{ mod } r_1 =$ $12 \text{ mod } 5 = 2$	$s_2 \leftarrow s_0 - q_2 \cdot s_1 =$ $1 - 2 \cdot 0 = 1$	$t_2 \leftarrow t_0 - q_2 \cdot t_1 =$ $0 - 2 \cdot 1 = -2$	$q_2 \leftarrow r_0 \text{ div } r_1 =$ $12 \text{ div } 5 = 2$
3	1	-2	5	2	$r_3 \leftarrow r_1 \text{ mod } r_2 =$ $5 \text{ mod } 2 = 1$	$s_3 \leftarrow s_1 - q_3 \cdot s_2 =$ $0 - 2 \cdot 1 = -2$	$t_3 \leftarrow t_1 - q_3 \cdot t_2 =$ $1 - 2 \cdot -2 = 5$	$q_3 \leftarrow r_1 \text{ div } r_2 =$ $5 \text{ div } 2 = 2$
4	0				$r_4 \leftarrow r_2 \text{ mod } r_3 =$ $2 \text{ mod } 1 = 0$			

From this we can see that the greatest common divisor of 12 and 5 is $gcd(12, 5) = 1$ and that the equation $1 = (-2) \cdot 12 + 5 \cdot 5$ holds. We can also invoke Sage to double check our findings:

```
sage: ZZ(12).xgcd(ZZ(5)) # (gcd(a,b),s,t)
(1, -2, 5)
```

Exercise 8 (Extended Euclidean Algorithm). Find integers $s, t \in \mathbb{Z}$ such that $gcd(a, b) = s \cdot a + t \cdot b$ holds for the following pairs:

- $(a, b) = (45, 10)$
- $(a, b) = (13, 11)$
- $(a, b) = (13, 12)$

Exercise 9 (Towards Prime fields). Let $n \in \mathbb{N}$ be a natural number and p a prime number, such that $n < p$. What is the greatest common divisor $gcd(p, n)$?

Exercise 10. Find all numbers $k \in \mathbb{N}$ with $0 \leq k \leq 100$ such that $gcd(100, k) = 5$.

Exercise 11. Show that $gcd(n, m) = gcd(n + m, m)$ for all $n, m \in \mathbb{N}$.

3.2.4 Coprime Integers

Coprime integers are integers that do not share a prime number as a factor. As we will see in 3.3, coprime integers are important for our purposes, because, in modular arithmetic, computations that involve coprime numbers are substantially different from computations on non-coprime numbers 3.3.2.

The naive way to decide if two integers are coprime would be to divide both numbers successively by all prime numbers smaller than those numbers, to see if they share a common prime factor. However, two integers are coprime if and only if their greatest common divisor is 1, which is why computing the *gcd* is the preferred method.

Example 5. Consider example 4 again. As we have seen, the greatest common divisor of 12 and 5 is 1. This implies that the integers 12 and 5 are coprime, since they share no divisor other than 1, which is not a prime number.

Exercise 12. Consider exercise 8 again. Which pairs (a,b) from that exercise are coprime?

3.3 Modular arithmetic

Modular arithmetic is a system of integer arithmetic where numbers “wrap around” when reaching a certain value, much like calculations on a clock wrap around whenever the value exceeds the number 12. For example, if the clock shows that it is 11 o’clock, then 20 hours later it will be 7 o’clock, not 31 o’clock. The number 31 has no meaning on a normal clock that shows hours.

The number at which the wrap occurs is called the **modulus**. Modular arithmetic generalizes the clock example to arbitrary moduli, and studies equations and phenomena that arise in this new kind of arithmetic. It is of central importance for understanding most modern cryptographic systems, in large parts because modular arithmetic provides the computational infrastructure for algebraic types that have cryptographically useful examples of one-way functions.

Although modular arithmetic appears very different from ordinary integer arithmetic that we are all familiar with, we encourage the interested reader to work through the examples and discover that, once they get used to the idea that this is a new kind of calculation, it will seem much less daunting.

the interested reader

3.3.1 Congruence

In what follows, let $n \in \mathbb{N}$ with $n \geq 2$ be a fixed natural number that we will call the **modulus** of our modular arithmetic system. With such an n given, we can then group integers into classes: two integers are in the same class whenever their Euclidean Division (3.2.2) by n will give the same remainder. We two numbers that are in the same class are called **congruent**.

Example 6. If we choose $n = 12$ as in our clock example, then the integers -7, 5, 17 and 29 are all congruent with respect to 12, since all of them have the remainder 5 if we perform Euclidean Division on them by 12. Imagining the picture of an analog 12-hour clock, starting at 5 o’clock and adding 12 hours, we are at 5 o’clock again, representing the number 17. Indeed, in many countries, 5:00 in the afternoon is written as 17:00. On the other hand, when we subtract 12 hours, we are at 5 o’clock again, representing the number -7.

We can formalize this intuition of what congruence should be into a proper definition utilizing Euclidean Division (as explained previously in 3.2). Let a, $b \in \mathbb{Z}$ be two integers, and

$n \in \mathbb{N}$ be a natural number such that $n \geq 2$. The integers a and b are said to be **congruent with respect to the modulus** n if and only if the following equation holds:

$$a \bmod n = b \bmod n \tag{3.14}$$

If, on the other hand, two numbers are not congruent with respect to a given modulus n, we call them **incongruent** w.r.t. n.

In other words, **congruence** is an equation "up to congruence", which means that the equation only needs to hold if we take the modulus of both sides. In which case we write

$$a \equiv b \pmod{n} \tag{3.15}$$

Exercise 13. Which of the following pairs of numbers are congruent with respect to the modulus 13:

- $(5, 19)$
- $(13, 0)$
- $(-4, 9)$
- $(0, 0)$

Exercise 14. Find all integers x, such that the congruence $x \equiv 4 \pmod{6}$ is satisfied.

3.3.2 Computational Rules

Having defined the notion of a congruence as an equation "up to a modulus", a follow-up question is if we can manipulate a congruence similarly to an equation. Indeed, we can almost apply the same substitution rules to a congruency as to an equation, with the main difference being that, for some non-zero integer $k \in \mathbb{Z}$, the congruence $a \equiv b \pmod{n}$ is equivalent to the congruence $k \cdot a \equiv k \cdot b \pmod{n}$ only if k and the modulus n are coprime (see 3.2.4).

Suppose that integers $a_1, a_2, b_1, b_2, k \in \mathbb{Z}$ are given. Then the following arithmetic rules hold for congruences:

- $a_1 \equiv b_1 \pmod{n} \Leftrightarrow a_1 + k \equiv b_1 + k \pmod{n}$ (compatibility with translation)
- $a_1 \equiv b_1 \pmod{n} \Rightarrow k \cdot a_1 \equiv k \cdot b_1 \pmod{n}$ (compatibility with scaling)
- $gcd(k, n) = 1$ and $k \cdot a_1 \equiv k \cdot b_1 \pmod{n} \Rightarrow a_1 \equiv b_1 \pmod{n}$
- $k \cdot a_1 \equiv k \cdot b_1 \pmod{k \cdot n} \Rightarrow a_1 \equiv b_1 \pmod{n}$
- $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n} \Rightarrow a_1 + a_2 \equiv b_1 + b_2 \pmod{n}$ (compatibility with addition)
- $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n} \Rightarrow a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod{n}$ (compatibility with multiplication)

Other rules, such as compatibility with subtraction, follow from the rules above. For example, compatibility with subtraction follows from compatibility with scaling by $k = -1$ and compatibility with addition.

Another property of congruences not found in the traditional arithmetic of integers is **Fermat's Little Theorem**. Simply put, it states that, in modular arithmetic, every number raised to the power of a prime number modulus is congruent to the number itself. Or, to be more precise, if $p \in \mathbb{P}$ is a prime number and $k \in \mathbb{Z}$ is an integer, then the following holds:

$$k^p \equiv k \pmod{p} \tag{3.16}$$

If k is coprime to p, then we can divide both sides of this congruence by k and rewrite the expression into the following equivalent form:

$$k^{p-1} \equiv 1 \pmod{p} \tag{3.17}$$

The Sage code below computes examples of Fermat's Little Theorem and highlights the effects of the exponent k being coprime to p (as in the case of 137 and 64) and not coprime to p (as in the case of 1918 and 137):

```
sage: ZZ(137).gcd(ZZ(64))
1
sage: ZZ(64)^ ZZ(137) % ZZ(137) == ZZ(64) % ZZ(137)
True
sage: ZZ(64)^ ZZ(137-1) % ZZ(137) == ZZ(1) % ZZ(137)
True
sage: ZZ(1918).gcd(ZZ(137))
137
sage: ZZ(1918)^ ZZ(137) % ZZ(137) == ZZ(1918) % ZZ(137)
True
sage: ZZ(1918)^ ZZ(137-1) % ZZ(137) == ZZ(1) % ZZ(137)
False
```

The following example contains most of the concepts described in this section.

Example 7. Let us solve the following congruence for $x \in \mathbb{Z}$ in modular 6 arithmetic:

$$7 \cdot (2x + 21) + 11 \equiv x - 102 \pmod{6}$$

As many rules for congruences are more or less same as for equations, we can proceed in a similar way as we would if we had an equation to solve. Since both sides of a congruence contain ordinary integers, we can rewrite the left side as follows:

$$7 \cdot (2x + 21) + 11 = 14x + 147 + 11 = 14x + 158$$

We can therefore rewrite the congruence into the equivalent form:

$$14x + 158 \equiv x - 102 \pmod{6}$$

In the next step, we want to shift all instances of x to the left and every other term to the right. So we apply the"compatibility with translation" rules twice. In the first step, we choose $k = -x$, and in a second step, we choose $k = -158$. separate steps 1 and 2 Since "compatibility with translation" transforms a congruence into an equivalent form, the solution set will not change, and we get the following:

SB: let's separate these two steps in the equivalence below

$$\begin{aligned} 14x + 158 \equiv x - 102 \pmod{6} \Leftrightarrow \\ 14x - x + 158 - 158 \equiv x - x - 102 - 158 \pmod{6} \Leftrightarrow \\ 13x \equiv -260 \pmod{6} \end{aligned}$$

If our congruence was just a regular integer equation, we would divide both sides by 13 to get $x = -20$ as our solution. However, in case of a congruence, we need to make sure that the modulus and the number we want to divide by are coprime to ensure that get an equivalent expression (see rule 3.17). Consequently, we need to find the greatest common divisor $gcd(13,6)$. Since 13 is prime and 6 is not a multiple of 13, we know that$gcd(13,6) = 1$, so these numbers are indeed coprime. We therefore compute as follows:

check reference

$$13x \equiv -260 \pmod 6 \Leftrightarrow x \equiv -20 \pmod 6$$

Our task now is to find all integers x such that x is congruent to -20 with respect to the modulus 6. In other words, we have to find all x such that the following equation holds:

$$x \bmod 6 = -20 \bmod 6$$

Since $-4 \cdot 6 + 4 = -20$, we know that $-20 \bmod 6 = 4$, and hence we know that $x = 4$ is a solution to this congruence. However, 22 is another solution, since $22 \bmod 6 = 4$ as well. Another solution is -20. In fact, there are infinitely many solutions given by the following set:

$$\{\ldots,-8,-2,4,10,16,\ldots\} = \{4 + k \cdot 6 \mid k \in \mathbb{Z}\}$$

Putting all this together, we have shown that every x from the set $\{x = 4 + k \cdot 6 \mid k \in \mathbb{Z}\}$ is a solution to the congruence $7 \cdot (2x + 21) + 11 \equiv x - 102 \pmod 6$. We double check for two arbitrary numbers from this set, $x = 4$ and $x = 4 + 12 \cdot 6 = 76$ using Sage:

```
sage: (ZZ(7)* (ZZ(2)*ZZ(4) + ZZ(21)) + ZZ(11)) % ZZ(6) == (ZZ
    (4) - ZZ(102)) % ZZ(6)
True
sage: (ZZ(7)* (ZZ(2)*ZZ(76) + ZZ(21)) + ZZ(11)) % ZZ(6) == (
    ZZ(76) - ZZ(102)) % ZZ(6)
True
```

Readers who had not been familiar with modular arithmetic until now and who might be discouraged by how complicated modular arithmetic seems at this point should keep two things in mind. First, computing congruences in modular arithmetic is not really more complicated than computations in more familiar number systems (e.g. rational numbers), it is just a matter of getting used to it. Second, once we introduce the idea of remainder class representations in 3.3.4, computations become conceptually cleaner and easier to handle.

Readers who

Exercise 15. Consider the modulus 13 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$5x + 4 \equiv 28 + 2x \pmod{13}$$

Exercise 16. Consider the modulus 23 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$69x \equiv 5 \pmod{23}$$

Exercise 17. Consider the modulus 23 and find all solutions $x \in \mathbb{Z}$ to the following congruence:

$$69x \equiv 46 \pmod{23}$$

Exercise 18. Let a,b,k be integers, such that $a \equiv b \pmod n$ holds. Show $a^k \equiv b^k \pmod n$.

Exercise 19. Let a,n be integers, such that a and n are not coprime. For which $b \in \mathbb{Z}$ does the congruence $a \cdot x \equiv b \pmod n$ have a solution x and how does the solution set look in that case?

3.3.3 The Chinese Remainder Theorem

We have seen how to solve congruences in modular arithmetic. In this section, we look at how to solve systems of congruences with different moduli using the **Chinese Remainder Theorem**. This states that, for any $k \in \mathbb{N}$ and coprime natural numbers $n_1, \ldots n_k \in \mathbb{N}$, as well as integers $a_1, \ldots a_k \in \mathbb{Z}$, the so-called **simultaneous congruences** (in 3.18 below) have a solution, and all possible solutions of this congruence system are congruent modulo the product $N = n_1 \cdot \ldots \cdot n_k$.[2]

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\ldots \\ x &\equiv a_k \pmod{n_k} \end{aligned} \tag{3.18}$$

The following algorithm computes the solution set:

check algorithm floating

Algorithm 2 Chinese Remainder Theorem

Require: , $k \in \mathbb{Z}$, $j \in \mathbb{N}_0$ and $n_0, \ldots, n_{k-1} \in \mathbb{N}$ coprime
  procedure CONGRUENCE-SYSTEMS-SOLVER($a_0, \ldots, a_{k-1}$)
    $N \leftarrow n_0 \cdot \ldots \cdot n_{k-1}$
    while $j < k$ **do**
      $N_j \leftarrow N/n_j$
      $(\_, s_j, t_j) \leftarrow EXT - EUCLID(N_j, n_j)$ ▷ $1 = s_j \cdot N_j + t_j \cdot n_j$
    end while
    $x' \leftarrow \sum_{j=0}^{k-1} a_j \cdot s_j \cdot N_j$
    $x \leftarrow x' \bmod N$
    return $\{x + m \cdot N \mid m \in \mathbb{Z}\}$
  end procedure
Ensure: $\{x + m \cdot N \mid m \in \mathbb{Z}\}$ is the complete solution set to 3.18.

Example 8. To illustrate how to solve simultaneous congruences using the Chinese Remainder Theorem, let's look at the following system of congruences:

$$\begin{aligned} x &\equiv 4 \pmod{7} \\ x &\equiv 1 \pmod{3} \\ x &\equiv 3 \pmod{5} \\ x &\equiv 0 \pmod{11} \end{aligned}$$

Clearly, all moduli are coprime (since they are all prime numbers). Now we calculate as follows:

$$\begin{aligned} N &= 7 \cdot 3 \cdot 5 \cdot 11 = 1155 \\ N_1 &= 1155/7 = 165 \\ N_2 &= 1155/3 = 385 \\ N_3 &= 1155/5 = 231 \\ N_4 &= 1155/11 = 105 \end{aligned}$$

From this, we calculate with the Extended Euclidean Algorithm:

add more explanation

[2] This is the classical Chinese Remainder Theorem as it was already known in ancient China. Under certain circumstances, the theorem can be extended to non-coprime moduli $n_1, \ldots, n_k$ but this is beyond the scope of this book. Interested readers should consult XXX add references

$$
\begin{aligned}
1 &= 2\cdot 165 + -47\cdot 7\\
1 &= 1\cdot 385 + -128\cdot 3\\
1 &= 1\cdot 231 + -46\cdot 5\\
1 &= 2\cdot 105 + -19\cdot 11
\end{aligned}
$$

Consequently, we get $x = 4\cdot 2\cdot 165 + 1\cdot 1\cdot 385 + 3\cdot 1\cdot 231 + 0\cdot 2\cdot 105 = 2398$ as one solution. Because $2398 \bmod 1155 = 88$, the set of all solutions is $\{\ldots,-2222,-1067,88,1243,2398,\ldots\}$. We can invoke Sage's computation of the Chinese Remainder Theorem (CRT) to double check our findings:

```
sage: CRT_list([4,1,3,0], [7,3,5,11])
88
```

3.3.4 Remainder Class Representation

As we have seen in various examples before, computing congruences can be cumbersome, and solution sets are large in general. It is therefore advantageous to find some kind of simplification for modular arithmetic.

Fortunately, this is possible and relatively straightforward once we identify each set of numbers that have equal remainders with that remainder itself, and call this set the **remainder class** or **residue class** representation in modulo n arithmetic.

It then follows from the properties of Euclidean Division that there are exactly n different remainder classes for every modulus n, and that integer addition and multiplication can be projected to a new kind of addition and multiplication on those classes.

Informally speaking, the new rules for addition and multiplication are then computed by taking any element of the first remainder class and some element of the second remainder class, then add or multiply them in the usual way and see which remainder class the result is contained in. The following example makes this abstract description more concrete.

Example 9 (Arithmetic modulo 6). Choosing the modulus $n = 6$, we have six remainder classes of integers which are congruent modulo 6, that is, they have the same remainder when divided by 6. When we identify each of those remainder classes with the remainder, we get the following identification:

$$
\begin{aligned}
0 &:= \{\ldots,-6,0,6,12,\ldots\}\\
1 &:= \{\ldots,-5,1,7,13,\ldots\}\\
2 &:= \{\ldots,-4,2,8,14,\ldots\}\\
3 &:= \{\ldots,-3,3,9,15,\ldots\}\\
4 &:= \{\ldots,-2,4,10,16,\ldots\}\\
5 &:= \{\ldots,-1,5,11,17,\ldots\}
\end{aligned}
$$

To compute the new addition law of those remainder class representatives, say $2+5$, one chooses an arbitrary element from each class, say 14 and -1, adds those numbers in the usual way, and then looks at the remainder class of the result.

one chooses

Adding 14 and (-1), we get 13, and 13 is in the remainder class (of) 1. Hence, we find that $2+5 = 1$ in modular 6 arithmetic, which is a more readable way to write the congruence $2+5 \equiv 1 \pmod 6$.

Applying the same reasoning to all remainder classes, addition and multiplication can be transferred to the representatives of the remainder classes. The results for modulus 6 arithmetic are summarized in the following addition and multiplication tables:

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

·	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

(3.19)

This way, we have defined a new arithmetic system that contains just 6 numbers and comes with its own definition of addition and multiplication. We call it **modular 6 arithmetic** and write the associated type as $\mathbb{Z}_6$.

To see why identifying a remainder class with its remainder is useful and actually simplifies congruence computations significantly, let's go back to the congruence from example 7:

$$7 \cdot (2x+21)+11 \equiv x-102 \pmod{6} \tag{3.20}$$

As shown in example 7, the arithmetic of congruences can deviate from ordinary arithmetic: for example, division needs to check whether the modulus and the dividend are coprimes, and solutions are not unique in general.

We can rewrite the congruence in (3.20) as an **equation** over our new arithmetic type $\mathbb{Z}_6$ by **projecting onto the remainder classes**: since $7 \bmod 6 = 1$, $21 \bmod 6 = 3$, $11 \bmod 6 = 5$ and $102 \bmod 6 = 0$, we get the following:

$$7 \cdot (2x+21)+11 \equiv x-102 \pmod{6} \text{ over } \mathbb{Z}$$
$$\Leftrightarrow\ 1 \cdot (2x+3)+5 = x \text{ over } \mathbb{Z}_6$$

We can use the multiplication and addition table in (3.19) above to solve the equation on the right like we would solve normal integer equations:

$$\begin{aligned}
1 \cdot (2x+3)+5 &= x \\
2x+3+5 &= x && \text{\# addition table: } 3+5=2 \\
2x+2 &= x && \text{\# add 4 and } -x \text{ on both sides} \\
2x+2+4-x &= x+4-x && \text{\# addition table: } 2+4=0 \\
x &= 4
\end{aligned}$$

As we can see, despite the somewhat unfamiliar rules of addition and multiplication, solving congruences this way is very similar to solving normal equations. And, indeed, the solution set is identical to the solution set of the original congruence, since 4 is identified with the set $\{4+6 \cdot k \mid k \in \mathbb{Z}\}$.

We can invoke Sage to do computations in our modular 6 arithmetic type. This is particularly useful to double-check our computations:

```
sage: Z6 = Integers(6)
sage: Z6(2) + Z6(5)
1
sage: Z6(7)*(Z6(2)*Z6(4)+Z6(21))+Z6(11) == Z6(4) - Z6(102)
True
```

Remark 2 (k-bit modulus). In cryptographic papers, we sometimes read phrases like "[...] using a 4096-bit modulus". This means that the underlying modulus n of the modular arithmetic used in the system has a binary representation with a length of 4096 bits. In contrast, the number 6 has the binary representation 110 and hence our example 9 describes a 3-bit modulus arithmetic system.

Exercise 20. Define $\mathbb{Z}_{13}$ as the the arithmetic modulo 13 analogously to example 9. Then consider the congruence from exercise 15 and rewrite it into an equation in $\mathbb{Z}_{13}$.

modulo/ modulus/ modular? unify throughout

3.3.5 Modular Inverses

As we know, integers can be added, subtracted and multiplied so that the result is also an integer, but this is not true for the division of integers in general: for example, $3/2$ is not an integer. To see why this is so from a more theoretical perspective, let us consider the definition of a multiplicative inverse first. When we have a set that has some kind of multiplication defined on it, and we have a distinguished element of that set that behaves neutrally with respect to that multiplication (doesn't change anything when multiplied with any other element), then we can define **multiplicative inverses** in the following way:

Definition 3.3.5.1. Let S be our set that has some notion $a \cdot b$ of multiplication and a **neutral element** $1 \in S$, such that $1 \cdot a = a$ for all elements $a \in S$. Then a **multiplicative inverse** a^{-1} of an element $a \in S$ is defined as follows:

$$a \cdot a^{-1} = 1 \tag{3.21}$$

Informally speaking, the definition of a multiplicative inverse is means that it "cancels" the original element, so that multiplying the two results in 1.

Numbers that have multiplicative inverses are of particular interest, because they immediately lead to the definition of division by those numbers. In fact, if a is number such that the multiplicative inverse a^{-1} exists, then we define **division** by a simply as multiplication by the inverse:

$$\frac{b}{a} := b \cdot a^{-1} \tag{3.22}$$

Example 10. Consider the set of rational numbers, also known as fractions, $\mathbb{Q}$. For this set, the neutral element of multiplication is 1, since $1 \cdot a = a$ for all rational numbers. For example, $1 \cdot 4 = 4$, $1 \cdot \frac{1}{4} = \frac{1}{4}$, or $1 \cdot 0 = 0$ and so on.

Every rational number $a \neq 0$ has a multiplicative inverse, given by $\frac{1}{a}$. For example, the multiplicative inverse of 3 is $\frac{1}{3}$, since $3 \cdot \frac{1}{3} = 1$, the multiplicative inverse of $\frac{5}{7}$ is $\frac{7}{5}$, since $\frac{5}{7} \cdot \frac{7}{5} = 1$, and so on.

Example 11. Looking at the set $\mathbb{Z}$ of integers, we see that the neutral element of multiplication is the number 1 We can also see that no integer other than 1 or -1 has a multiplicative inverse, since the equation $a \cdot x = 1$ has no integer solutions for $a \neq 1$ or $a \neq -1$.

The definition of multiplicative inverse has a parallel for addition called the **additive inverse**. In the case of integers, the neutral element with respect to addition is 0, since $a + 0 = 0$ for all integers $a \in \mathbb{Z}$. The additive inverse always exists, and is given by the negative number $-a$, since $a + (-a) = 0$.

Example 12. Looking at the set $\mathbb{Z}_6$ of residue classes modulo 6 from example 9, we can use the multiplication table in (3.19) to find multiplicative inverses. To do so, we look at the row of the element and find the entry equal to 1. If such an entry exists, the element of that column is the multiplicative inverse. If, on the other hand, the row has no entry equal to 1, we know that the element has no multiplicative inverse.

For example in, $\mathbb{Z}_6$, the multiplicative inverse of 5 is 5 itself, since $5 \cdot 5 = 1$. We can also see that 5 and 1 are the only elements that have multiplicative inverses in $\mathbb{Z}_6$.

Now, since 5 has a multiplicative inverse in modulo 6 arithmetic, we can divide by 5 in $\mathbb{Z}_6$, since we have a notation of multiplicative inverse and division is nothing but multiplication by the multiplicative inverse:

$$\frac{4}{5} = 4 \cdot 5^{-1} = 4 \cdot 5 = 2$$

From the last example, we can make the interesting observation that, while 5 has no multiplicative inverse as an integer, it has a multiplicative inverse in modular 6 arithmetic.

This raises the question of which numbers have multiplicative inverses in modular arithmetic. The answer is that, in modular n arithmetic, a number r has a multiplicative inverse if and only if n and r are coprime. Since $gcd(n,r) = 1$ in that case, we know from the Extended Euclidean Algorithm that there are numbers s and t, such that the following equation holds:

$$1 = s \cdot n + t \cdot r \tag{3.23}$$

If we take the modulus n on both sides, the term $s \cdot n$ vanishes, which tells us that t mod n is the multiplicative inverse of r in modular n arithmetic.

expand on this

Example 13 (Multiplicative inverses in $\mathbb{Z}_6$). In the previous example, we looked up multiplicative inverses in $\mathbb{Z}_6$ from the lookup table in (3.19). In real-world examples, it is usually impossible to write down those lookup tables, as the modulus is way too large, and the sets occasionally contain more elements than there are atoms in the observable universe.

way

Now, trying to determine that $2 \in \mathbb{Z}_6$ has no multiplicative inverse in $\mathbb{Z}_6$ without using the lookup table, we immediately observe that 2 and 6 are not coprime, since their greatest common divisor is 2. It follows that equation 3.23 has no solutions s and t, which means that 2 has no multiplicative inverse in Z_6.

The same reasoning works for 3 and 4, as neither of these are coprime with 6. The case of 5 is different, since $gcd(6,5) = 1$. To compute the multiplicative inverse of 5, we use the Extended Euclidean Algorithm and compute the following:

k	r_k	s_k	$t_k = (r_k - s_k \cdot a)$ div b
0	6	1	0
1	5	0	1
2	1	1	-1
3	0	·	·

We get $s = 1$ as well as $t = -1$ and have $1 = 1 \cdot 6 - 1 \cdot 5$. From this, it follows that -1 mod $6 = 5$ is the multiplicative inverse of 5 in modular 6 arithmetic. We can double check using Sage:

```
sage: ZZ(6).xgcd(ZZ(5))
(1, 1, -1)
```

At this point, the attentive reader might notice that the situation where the modulus is a prime number is of particular interest, because we know from exercise 9 that, in these cases, all remainder classes must have modular inverses, since $gcd(r,n) = 1$ for prime n and any $r < n$. In fact, Fermat's Little Theorem (3.16) provides a way to compute multiplicative inverses in this situation, since, in case of a prime modulus p and $r < p$, we get the following:

$$
\begin{aligned}
r^p &\equiv r \pmod{p} \Leftrightarrow \\
r^{p-1} &\equiv 1 \pmod{p} \Leftrightarrow \\
r \cdot r^{p-2} &\equiv 1 \pmod{p}
\end{aligned}
$$

This tells us that the multiplicative inverse of a residue class r in modular p arithmetic is precisely r^{p-2}.

Example 14 (Modular 5 arithmetic). To see the unique properties of modular arithmetic when the modulus is a prime number, we will replicate our findings from example 9, but this time for the prime modulus 5. For $p = 5$ we have five equivalence classes of integers which are congruent modulo 5. We write this as follows:

$$
\begin{aligned}
0 &:= \{\ldots, -5, 0, 5, 10, \ldots\} \\
1 &:= \{\ldots, -4, 1, 6, 11, \ldots\} \\
2 &:= \{\ldots, -3, 2, 7, 12, \ldots\} \\
3 &:= \{\ldots, -2, 3, 8, 13, \ldots\} \\
4 &:= \{\ldots, -1, 4, 9, 14, \ldots\}
\end{aligned}
$$

Addition and multiplication can be transferred to the equivalence classes, in a way exactly parallel to Example 9. This results in the following addition and multiplication tables:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

(3.24)

Calling the set of remainder classes in modular 5 arithmetic with this addition and multiplication $\mathbb{Z}_5$, we see some subtle but important differences to the situation in $\mathbb{Z}_6$. In particular, we see that in the multiplication table, every remainder $r \neq 0$ has the entry 1 in its row and therefore has a multiplicative inverse. In addition, there are no non-zero elements such that their product is zero.

To use Fermat's Little Theorem in $\mathbb{Z}_5$ for computing multiplicative inverses (instead of using the multiplication table), let's consider $3 \in \mathbb{Z}_5$. We know that the multiplicative inverse is given by the remainder class that contains $3^{5-2} = 3^3 = 3 \cdot 3 \cdot 3 = 4 \cdot 3 = 2$. And indeed $3^{-1} = 2$, since $3 \cdot 2 = 1$ in $\mathbb{Z}_5$.

We can invoke Sage to do computations in our modular 5 arithmetic type to double-check our computations:

```
sage: Z5 = Integers(5)
sage: Z5(3)**(5-2)
2
sage: Z5(3)**(-1)
2
sage: Z5(3)**(5-2) == Z5(3)**(-1)
True
```

Example 15. To understand one of the principal differences between prime number modular arithmetic and non-prime number modular arithmetic, consider the linear equation $a \cdot x + b = 0$ defined over both types $\mathbb{Z}_5$ and $\mathbb{Z}_6$. Since every non-zero element has a multiplicative inverse in $\mathbb{Z}_5$, we can always solve these equations in $\mathbb{Z}_5$, which is not true in $\mathbb{Z}_6$. To see that, consider the equation $3x+3=0$. In $\mathbb{Z}_5$ we have the following:

$$\begin{aligned}
3x+3 &= 0 && \text{\# add 2 and on both sides}\\
3x+3+2 &= 2 && \text{\# addition-table: } 2+3=0\\
3x &= 2 && \text{\# divide by 3 (which equals multiplication by 2)}\\
2\cdot(3x) &= 2\cdot 2 && \text{\# multiplication-table: } 2\cdot 2 = 4\\
x &= 4
\end{aligned}$$

So in the case of our prime number modular arithmetic, we get the unique solution $x = 4$. Now consider $\mathbb{Z}_6$:

$$\begin{aligned}
3x+3 &= 0 && \text{\# add 3 and on both sides}\\
3x+3+3 &= 3 && \text{\# addition-table: } 3+3=0\\
3x &= 3 && \text{\# division not possible (no multiplicative inverse of 3 exists)}
\end{aligned}$$

So, in this case, we cannot solve the equation for x by dividing by 3. And, indeed, when we look at the multiplication table of $\mathbb{Z}_6$ (Example 9), we find that there are three solutions $x \in \{1,3,5\}$, such that $3x+3=0$ holds true for all of them.

Exercise 21. Consider the modulus $n = 24$. Which of the integers 7, 1, 0, 805, -4255 have multiplicative inverses in modular 24 arithmetic? Compute the inverses, in case they exist.

Exercise 22. Find the set of all solutions to the congruence $17(2x+5)-4 \equiv 2x+4 \pmod 5$. Then project the congruence into $\mathbb{Z}_5$ and solve the resulting equation in $\mathbb{Z}_5$. Compare the results.

Exercise 23. Find the set of all solutions to the congruence $17(2x+5)-4 \equiv 2x+4 \pmod 6$. Then project the congruence into $\mathbb{Z}_6$ and try to solve the resulting equation in $\mathbb{Z}_6$.

3.4 Polynomial arithmetic

A polynomial is an expression consisting of variables (also-called indeterminates) and coefficients that involves only the operations of addition, subtraction and multiplication. All coefficients of a polynomial must have the same type, e.g. they must all be integers or they must all be rational numbers, etc.

To be more precise, an **univariate**[3] **polynomial** is an expression as shown below:

$$P(x) := \sum_{j=0}^{m} a_j x^j = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \,, \tag{3.25}$$

In (3.25) x is called the **variable**, and each a is called a **coefficient**. If $\mathbb{R}$ is the type of the coefficients, then the set of all **univariate polynomials with coefficients in** $\mathbb{R}$ is written as $\mathbb{R}[x]$. Univariate polynomials are often simply called polynomials, and written as $P(x) \in \mathbb{R}[x]$. The constant term a_0 as is also written as $P(0)$.

A polynomial is called the **zero polynomial** if all its coefficients are zero. A polynomial is called the **one polynomial** if the constant term is 1 and all other coefficients are zero.

[3]In our context, the term univariate means that the polynomial contains a single variable only.

Given a univariate polynomial $P(x) = \sum_{j=0}^{m} a_j x^j$ that is not the zero polynomial, we call the non-negative integer $deg(P) := m$ the *degree* of P, and define the degree of the zero polynomial to be $-\infty$, where $-\infty$ (negative infinity) is a symbol with the properties that $-\infty + m = -\infty$ and $-\infty < m$ for all non-negative integers $m \in \mathbb{N}_0$.

In addition, we denote the coefficient of the term with the highest degree, called **leading coefficient**, of the polynomial P as follows:

add explanation on why this is important

$$Lc(P) := a_m \tag{3.26}$$

We can restrict the set $\mathbb{R}[x]$ of **all** polynomials with coefficients in $\mathbb{R}$ to the set of all such polynomials that have a degree that does not exceed a certain value. If m is the maximum degree allowed, we write $\mathbb{R}_{\leq m}[x]$ for the set of all polynomials with a degree less than or equal to m.

Example 16 (Integer Polynomials). The coefficients of a polynomial must all have the same type. The set of polynomials with integer coefficients is written as $\mathbb{Z}[x]$. Some examples of such polynomials are listed below:

$$\begin{aligned}
P_1(x) &= 2x^2 - 4x + 17 && \text{\# with } deg(P_1) = 2 \text{ and } Lc(P_1) = 2 \\
P_2(x) &= x^{23} && \text{\# with } deg(P_2) = 23 \text{ and } Lc(P_2) = 1 \\
P_3(x) &= x && \text{\# with } deg(P_3) = 1 \text{ and } Lc(P_3) = 1 \\
P_4(x) &= 174 && \text{\# with } deg(P_4) = 0 \text{ and } Lc(P_4) = 174 \\
P_5(x) &= 1 && \text{\# with } deg(P_5) = 0 \text{ and } Lc(P_5) = 1 \\
P_6(x) &= 0 && \text{\# with } deg(P_6) = -\infty \text{ and } Lc(P_6) = 0 \\
P_7(x) &= (x-2)(x+3)(x-5)
\end{aligned}$$

Every integer can be seen as an integer polynomial of degree zero. P_7 is a polynomial, because we can expand its definition into $P_7(x) = x^3 - 4x^2 - 11x + 30$, which is a polynomial of degree 3 and leading coefficient 1.

The following expressions are not integer polynomials:

$$\begin{aligned}
Q_1(x) &= 2x^2 + 4 + 3x^{-2} \\
Q_2(x) &= 0.5x^4 - 2x \\
Q_3(x) &= 2^x
\end{aligned}$$

Q_1 is not an integer polynomial because the expression x^{-2} has a negative exponent. Q_2 is not an integer polynomial because the coefficient 0.5 is not an integer. Q_3 is not an integer polynomial because the variable appears in the exponent of a coefficient.

We can invoke Sage to do computations with polynomials. To do so, we have to specify the symbol for the variable and the type for the coefficients. (For the definition of rings see 4.2.) Note, however, that Sage defines the degree of the zero polynomial to be -1.

why is this ref here?

what does this imply?

```
sage: Zx = ZZ['x'] # integer polynomials with variable x
sage: Zt.<t> = ZZ[] # integer polynomials with variable t
sage: Zx
Univariate Polynomial Ring in x over Integer Ring
sage: Zt
Univariate Polynomial Ring in t over Integer Ring
sage: p1 = Zx([17,-4,2])
sage: p1
```

```
2*x^2 - 4*x + 17
sage: p1.degree()
2
sage: p1.leading_coefficient()
2
sage: p2 = Zt(t^23)
sage: p2
t^23
sage: p6 = Zx([0])
sage: p6.degree()
-1
```

Example 17 (Polynomials over $\mathbb{Z}_6$). Recall the definition of modular 6 arithmetics $\mathbb{Z}_6$ from example 9. The set of all polynomials with variable x and coefficients in $\mathbb{Z}_6$ is symbolized as $\mathbb{Z}_6[x]$. Some examples of polynomials from $\mathbb{Z}_6[x]$ are given below:

$$
\begin{aligned}
&P_1(x) = 2x^2 - 4x + 5 && \# \text{ with } deg(P_1) = 2 \text{ and } Lc(P_1) = 2\\
&P_2(x) = x^{23} && \# \text{ with } deg(P_2) = 23 \text{ and } Lc(P_2) = 1\\
&P_3(x) = x && \# \text{ with } deg(P_3) = 1 \text{ and } Lc(P_3) = 1\\
&P_4(x) = 3 && \# \text{ with } deg(P_4) = 0 \text{ and } Lc(P_4) = 3\\
&P_5(x) = 1 && \# \text{ with } deg(P_5) = 0 \text{ and } Lc(P_5) = 1\\
&P_6(x) = 0 && \# \text{ with } deg(P_5) = -\infty \text{ and } Lc(P_6) = 0\\
&P_7(x) = (x-2)(x+3)(x-5)
\end{aligned}
$$

Just like in the previous example, P_7 is a polynomial. However, since we are working with coefficients from $\mathbb{Z}_6$ now, the expansion of P_7 is computed differently, as we have to invoke addition and multiplication in $\mathbb{Z}_6$ as defined in (3.19). We get the following:

check reference

$$
\begin{aligned}
(x-2)(x+3)(x-5) &= (x+4)(x+3)(x+1) && \# \text{ additive inverses in } \mathbb{Z}_6\\
&= (x^2 + 4x + 3x + 3\cdot 4)(x+1) && \# \text{ bracket expansion}\\
&= (x^2 + 1x + 0)(x+1) && \# \text{ computation in } \mathbb{Z}_6\\
&= x^3 + x^2 + x^2 + x && \# \text{ bracket expansion}\\
&= x^3 + 2x^2 + x
\end{aligned}
$$

Again, we can use Sage to do computations with polynomials that have their coefficients in $\mathbb{Z}_6$. (For the definition of rings see 4.2.) To do so, we have to specify the symbol for the variable and the type for the coefficients:

why is this ref here?

```
sage: Z6 = Integers(6)
sage: Z6x = Z6['x']
sage: Z6x
Univariate Polynomial Ring in x over Ring of integers modulo 6
sage: p1 = Z6x([5,-4,2])
sage: p1
2*x^2 + 2*x + 5
sage: p1 = Z6x([17,-4,2])
sage: p1
```

```
2*x^2 + 2*x + 5
sage: Z6x(x-2)*Z6x(x+3)*Z6x(x-5) == Z6x(x^3 + 2*x^2 + x)
True
```

Given some element from the same type as the coefficients of a polynomial, the polynomial can be evaluated at that element, which means that we insert the given element for every occurrence of the variable x in the polynomial expression.

To be more precise, let $P \in \mathbb{R}[x]$, with $P(x) = \sum_{j=0}^{m} a_j x^j$ be a polynomial with a coefficient of type $\mathbb{R}$ and let $b \in \mathbb{R}$ be an element of that type. Then the **evaluation** of P at b is given as follows:

$$P(b) = \sum_{j=0}^{m} a_j b^j \tag{3.27}$$

Example 18. Consider the integer polynomials from example 16 again. To evaluate them at given points, we have to insert the point for all occurences of x in the polynomial expression. Inserting arbitrary values from $\mathbb{Z}$, we get the following:

$$\begin{aligned}
P_1(2) &= 2\cdot 2^2 - 4\cdot 2 + 17 = 17\\
P_2(3) &= 3^{23} = 94143178827\\
P_3(-4) &= -4 = -4\\
P_4(15) &= 174\\
P_5(0) &= 1\\
P_6(1274) &= 0\\
P_7(-6) &= (-6-2)(-6+3)(-6-5) = -264
\end{aligned}$$

Note, however, that it is not possible to evaluate any of those polynomial on values of different type. For example, it is not strictly correct to write $P_1(0.5)$, since 0.5 is not an integer. We can verify our computations using Sage:

```
sage: Zx = ZZ['x']
sage: p1 = Zx([17,-4,2])
sage: p7 = Zx(x-2)*Zx(x+3)*Zx(x-5)
sage: p1(ZZ(2))
17
sage: p7(ZZ(-6)) == ZZ(-264)
True
```

Example 19. Consider the polynomials with coefficients in $\mathbb{Z}_6$ from example 17 again. To evaluate them at given values from $\mathbb{Z}_6$, we have to insert the point for all occurrences of x in the polynomial expression. We get the following: check reference

$$\begin{aligned}
P_1(2) &= 2\cdot 2^2 - 4\cdot 2 + 5 = 2 - 2 + 5 = 5\\
P_2(3) &= 3^{23} = 3\\
P_3(-4) &= P_3(2) = 2\\
P_5(0) &= 1\\
P_6(4) &= 0
\end{aligned}$$

```
sage: Z6 = Integers(6)
sage: Z6x = Z6['x']
sage: p1 = Z6x([5,-4,2])
sage: p1(Z6(2)) == Z6(5)
True
```

Exercise 24. Compare both expansions of P_7 from $\mathbb{Z}[x]$ in example 16 and from from $\mathbb{Z}_6[x]$ in example 17 , and consider the definition of $\mathbb{Z}_6$ as given in example 9. Can you see how the definition of P_7 over $\mathbb{Z}$ projects to the definition over $\mathbb{Z}_6$ if you consider the residue classes of $\mathbb{Z}_6$?

the task could be defined more clearly

3.4.1 Polynomial arithmetic

Polynomials behave like integers in many ways. In particular, they can be added, subtracted and multiplied. In addition, they have their own notion of Euclidean Division. Informally speaking, we can add two polynomials by simply adding the coefficients of the same index, and we can multiply them by applying the distributive property, that is, by multiplying every term of the left factor with every term of the right factor and adding the results together.

To be more precise, let $\sum_{n=0}^{m_1} a_n x^n$ and $\sum_{n=0}^{m_2} b_n x^n$ be two polynomials from $\mathbb{R}[x]$. Then the **sum** and the **product** of these polynomials is defined as follows:

$$\sum_{n=0}^{m_1} a_n x^n + \sum_{n=0}^{m_2} b_n x^n = \sum_{n=0}^{max(\{m_1,m_2\})} (a_n + b_n) x^n \tag{3.28}$$

$$\left(\sum_{n=0}^{m_1} a_n x^n\right) \cdot \left(\sum_{n=0}^{m_2} b_n x^n\right) = \sum_{n=0}^{m_1+m_2} \sum_{i=0}^{n} a_i b_{n-i} x^n \tag{3.29}$$

A rule for polynomial subtraction can be deduced from these two rules by first multiplying the subtrahend with (the polynomial) -1 and then add the result to the minuend.

subtrahend

minuend

Regarding the definition of the degree of a polynomial, we see that the degree of the sum is always the maximum of the degrees of both summands, and the degree of the product is always the degree of the sum of the factors, since we defined $-\infty + m = -\infty$ for every integer $m \in \mathbb{Z}$.

Example 20. To give an example of how polynomial arithmetic works, consider the following two integer polynomials $P, Q \in \mathbb{Z}[x]$ with $P(x) = 5x^2 - 4x + 2$ and $Q(x) = x^3 - 2x^2 + 5$. The sum of these two polynomials is computed by adding the coefficients of each term with equal exponent in x. This gives the following:

$$\begin{aligned}(P+Q)(x) &= (0+1)x^3 + (5-2)x^2 + (-4+0)x + (2+5)\\ &= x^3 + 3x^2 - 4x + 7\end{aligned}$$

The product of these two polynomials is computed by multiplying each term in the first factor with each term in the second factor. We get the following:

$$\begin{aligned}(P \cdot Q)(x) &= (5x^2 - 4x + 2) \cdot (x^3 - 2x^2 + 5)\\ &= (5x^5 - 10x^4 + 25x^2) + (-4x^4 + 8x^3 - 20x) + (2x^3 - 4x^2 + 10)\\ &= 5x^5 - 14x^4 + 10x^3 + 21x^2 - 20x + 10\end{aligned}$$

```
sage: Zx = ZZ['x']
sage: P = Zx([2,-4,5])
sage: Q = Zx([5,0,-2,1])
sage: P+Q == Zx(x^3 +3*x^2 -4*x +7)
True
sage: P*Q == Zx(5*x^5 -14*x^4 +10*x^3+21*x^2-20*x +10)
True
```

Example 21. Let us consider the polynomials of the previous example 20, but interpreted in modular 6 arithmetic. So we consider $P,Q \in \mathbb{Z}_6[x]$ again with $P(x) = 5x^2-4x+2$ and $Q(x) = x^3-2x^2+5$. This time we get the following:

$$\begin{aligned}(P+Q)(x) &= (0+1)x^3+(5-2)x^2+(-4+0)x+(2+5)\\ &= (0+1)x^3+(5+4)x^2+(2+0)x+(2+5)\\ &= x^3+3x^2+2x+1\end{aligned}$$

$$\begin{aligned}(P\cdot Q)(x) &= (5x^2-4x+2)\cdot(x^3-2x^2+5)\\ &= (5x^2+2x+2)\cdot(x^3+4x^2+5)\\ &= (5x^5+2x^4+1x^2)+(2x^4+2x^3+4x)+(2x^3+2x^2+4)\\ &= 5x^5+4x^4+4x^3+3x^2+4x+4\end{aligned}$$

```
sage: Z6x = Integers(6)['x']
sage: P = Z6x([2,-4,5])
sage: Q = Z6x([5,0,-2,1])
sage: P+Q == Z6x(x^3 +3*x^2 +2*x +1)
True
sage: P*Q == Z6x(5*x^5 +4*x^4 +4*x^3+3*x^2+4*x +4)
True
```

Exercise 25. Compare the sum $P+Q$ and the product $P\cdot Q$ from the previous two examples 20 and 21, and consider the definition of $\mathbb{Z}_6$ as given in example 9. How can we derive the computations in $\mathbb{Z}_6[x]$ from the computations in $Z[x]$?

3.4.2 Euclidean Division with polynomials

The arithmetic of polynomials shares a lot of properties with the arithmetic of integers. As a consequence, the concept of Euclidean Division and the algorithm of long division is also defined for polynomials. Recalling the Euclidean Division of integers 3.2.2, we know that, given two integers a and $b \neq 0$, there is always another integer m and a natural number r with $r < |b|$ such that $a = m\cdot b + r$ holds.

We can generalize this to polynomials whenever the leading coefficient of the dividend polynomial has a notion of multiplicative inverse. In fact, given two polynomials A and $B \neq 0$ from $\mathbb{R}[x]$ such that $Lc(B)^{-1}$ exists in $\mathbb{R}$, there exist two polynomials Q (the quotient) and P (the remainder), such that the following equation holds and $deg(P) < deg(B)$:

$$A = Q\cdot B + P \tag{3.30}$$

Similarly to integer Euclidean Division, both Q and P are uniquely defined by these relations.

Notation and Symbols 2. Suppose that the polynomials A, B, Q and P satisfy equation 3.30. We often use the following notation to describe the quotient and the remainder polynomials of the Euclidean Division:

$$A \text{ div } B := Q, \qquad A \text{ mod } B := P \tag{3.31}$$

We also say that a polynomial A is divisible by another polynomial B if $A \text{ mod } B = 0$ holds. In this case, we also write $B|A$ and call B a *factor* of A.

Analogously to integers, methods to compute Euclidean Division for polynomials are called **polynomial division algorithms**. Probably the best known algorithm is the so-called **polynomial long division** . algorithm-floating

Algorithm 3 Polynomial Euclidean Algorithm

Require: $A, B \in R[x]$ with $B \neq 0$, such that $Lc(B)^{-1}$ exists in R
  procedure POLY-LONG-DIVISION(A, B)
    $Q \leftarrow 0$
    $P \leftarrow A$
    $d \leftarrow deg(B)$
    $c \leftarrow Lc(B)$
    while $deg(P) \geq d$ **do**
      $S := Lc(P) \cdot c^{-1} \cdot x^{deg(P)-d}$
      $Q \leftarrow Q + S$
      $P \leftarrow P - S \cdot B$
    end while
    return (Q, P)
  end procedure
Ensure: $A = Q \cdot B + P$

This algorithm works only when there is a notion of division by the leading coefficient of B. It can be generalized, but we will only need this somewhat simpler method in what follows.

Example 22 (Polynomial Long Division). To give an example of how the previous algorithm works, let us divide the integer polynomial $A(x) = x^5 + 2x^3 - 9 \in \mathbb{Z}[x]$ by the integer polynomial $B(x) = x^2 + 4x - 1 \in \mathbb{Z}[x]$. Since B is not the zero polynomial, and the leading coefficient of B is 1, which is invertible as an integer, we can apply algorithm 1. Our goal is to find solutions to equation XXX, that is, we need to find the quotient polynomial $Q \in \mathbb{Z}[x]$ and the remainder polynomial $P \in \mathbb{Z}[x]$ such that $x^5 + 2x^3 - 9 = Q(x) \cdot (x^2 + 4x - 1) + P(x)$. Using a the long add reference division notation that is mostly used in anglophone countries, we compute as follows:

$$
\begin{array}{r|rrrrrr}
 & & & X^3 & -4X^2 & +19X & -80 \\
\hline
X^2+4X-1 & X^5 & & +2X^3 & & & -9 \\
 & -X^5 & -4X^4 & +X^3 & & & \\
\cline{2-4}
 & & -4X^4 & +3X^3 & & & \\
 & & 4X^4 & +16X^3 & -4X^2 & & \\
\cline{3-5}
 & & & 19X^3 & -4X^2 & & \\
 & & & -19X^3 & -76X^2 & +19X & \\
\cline{4-6}
 & & & & -80X^2 & +19X & -9 \\
 & & & & 80X^2 & +320X & -80 \\
\cline{5-7}
 & & & & & 339X & -89
\end{array}
\tag{3.32}
$$

We therefore get $Q(x) = x^3 - 4x^2 + 19x - 80$ and $P(x) = 339x - 89$, and indeed, the equation $A = Q \cdot B + P$ is true with these valude, since $x^5 + 2x^3 - 9 = (x^3 - 4x^2 + 19x - 80) \cdot (x^2 + 4x - 1) + (339x - 89)$. We can double check this invoking Sage:

```
sage: Zx = ZZ['x']
sage: A = Zx([-9,0,0,2,0,1])
sage: B = Zx([-1,4,1])
sage: Q = Zx([-80,19,-4,1])
sage: P = Zx([-89,339])
sage: A == Q*B + P
True
```

Example 23. In the previous example, polynomial division gave a non-trivial (non-vanishing, i.e non-zero) remainder. Divisions that don't give a remainder are of special interest. In these cases, divisors are called **factors of the dividend**.

For example, consider the integer polynomial P_7 from example 16 again. As we have shown, it can be written both as $x^3 - 4x^2 - 11x + 30$ and as $(x-2)(x+3)(x-5)$. From this, we can see that the polynomials $F_1(x) = (x-2)$, $F_2(x) = (x+3)$ and $F_3(x) = (x-5)$ are all factors of $x^3 - 4x^2 - 11x + 30$, since division of P_7 by any of these factors will result in a zero remainder.

Exercise 26. Consider the polynomial expressions $A(x) := -3x^4 + 4x^3 + 2x^2 + 4$ and $B(x) = x^2 - 4x + 2$. Compute the Euclidean Division of A by B in the following types:

1. $A, B \in \mathbb{Z}[x]$
2. $A, B \in \mathbb{Z}_6[x]$
3. $A, B \in \mathbb{Z}_5[x]$

Now consider the result in $\mathbb{Z}[x]$ and in $\mathbb{Z}_6[x]$. How can we compute the result in $\mathbb{Z}_6[x]$ from the result in $\mathbb{Z}[x]$?

Exercise 27. Show that the polynomial $B(x) = 2x^4 - 3x + 4 \in \mathbb{Z}_5[x]$ is a factor of the polynomial $A(x) = x^7 + 4x^6 + 4x^5 + x^3 + 2x^2 + 2x + 3 \in \mathbb{Z}_5[x]$, that is, show that $B|A$. What is B div A?

3.4.3 Prime Factors

Recall that the fundamental theorem of arithmetic 3.7 tells us that every natural number is the product of prime numbers. In this chapter, we will see that something similar holds for univariate polynomials $R[x]$, too.[4]

The polynomial analog to a prime number is a so-called **irreducible polynomial**, which is defined as a polynomial that cannot be factored into the product of two non-constant polynomials using Euclidean Division. Irreducible polynomials are to polynomials what prime numbers are to integers: they are the basic building blocks from which all other polynomials can be constructed.

To be more precise, let $P \in \mathbb{R}[x]$ be any polynomial. Then there always exist irreducible polynomials $F_1, F_2, \ldots, F_k \in \mathbb{R}[x]$, such that the following holds:

$$P = F_1 \cdot F_2 \cdot \ldots \cdot F_k \,. \tag{3.33}$$

This representation is unique (except for permutations in the factors) and is called the **prime factorization** of P. Moreover, each factor F_i is called a **prime factor** of P.

Example 24. Consider the polynomial expression $P = x^2 - 3$. When we interpret P as an integer polynomial $P \in \mathbb{Z}[x]$, we find that this polynomial is irreducible, since any factorization other then $1 \cdot (x^2 - 3)$, must look like $(x-a)(x+a)$ for some integer a, but there is no integers a with $a^2 = 3$.

```
sage: Zx = ZZ['x']
sage: p = Zx(x^2-3)
sage: p.factor()
x^2 - 3
```

On the other hand, interpreting P as a polynomial $P \in \mathbb{Z}_6[x]$ in modulo 6 arithmetic, we see that P has two factors $F_1 = (x-3)$ and $F_2 = (x+3)$, since $(x-3)(x+3) = x^2 - 3x + 3x - 3 \cdot 3 = x^2 - 3$.

Points where a polynomial evaluates to zero are called **roots** of the polynomial. To be more precise, let $P \in \mathbb{R}[x]$ be a polynomial. Then a root is a point $x_0 \in \mathbb{R}$ with $P(x_0) = 0$ and the set of all roots of P is defined as follows:

$$R_0(P) := \{x_0 \in \mathbb{R} \mid P(x_0) = 0\} \tag{3.34}$$

The roots of a polynomial are of special interest with respect to its prime factorization, since it can be shown that, for any given root x_0 of P, the polynomial $F(x) = (x - x_0)$ is a prime factor of P.

Finding the roots of a polynomial is sometimes called **solving the polynomial**. It is a difficult problem that has been the subject of much research throughout history.

It can be shown that if m is the degree of a polynomial P, then P cannot have more than m roots. However, in general, polynomials can have less than m roots.

Example 25. Consider the integer polynomial $P_7(x) = x^3 - 4x^2 - 11x + 30$ from example 16 again. We know that its set of roots is given by $R_0(P_7) = \{-3, 2, 5\}$.

On the other hand, we know from example 24 that the integer polynomial $x^2 - 3$ is irreducible. It follows that it has no roots, since every root defines a prime factor.

[4] Strictly speaking, this is not true for polynomials over arbitrary types $\mathbb{R}$. However, in this book, we assume $\mathbb{R}$ to be a so-called unique factorization domain for which the content of this section holds.

Example 26. To give another example, consider the integer polynomial $P = x^7 + 3x^6 + 3x^5 + x^4 - x^3 - 3x^2 - 3x - 1$. We can invoke Sage to compute the roots and prime factors of P:

```
sage: Zx = ZZ['x']
sage: p = Zx(x^7 + 3*x^6 + 3*x^5 + x^4 - x^3 - 3*x^2 - 3*x - 1
   )
sage: p.roots()
[(1, 1), (-1, 4)]
sage: p.factor()
(x - 1) * (x + 1)^4 * (x^2 + 1)
```

We see that P has the root 1, and that the associated prime factor $(x-1)$ occurs once in P. We can also see that P has the root -1, where the associated prime factor $(x+1)$ occurs 4 times in P. This gives the following prime factorization:

$$P = (x-1)(x+1)^4(x^2+1)$$

Exercise 28. Show that if a polynomial $P \in \mathbb{R}[x]$ of degree $deg(P) = m$ has less then m roots, it must have a prime factor F of degree $deg(F) > 1$.

Exercise 29. Consider the polynomial $P = x^7 + 3x^6 + 3x^5 + x^4 - x^3 - 3x^2 - 3x - 1 \in \mathbb{Z}_6[x]$. Compute the set of all roots of $R_0(P)$ and then compute the prime factorization of P.

3.4.4 Lagrange interpolation

One particularly useful property of polynomials is that a polynomial of degree m is completely determined on $m+1$ evaluation points, which implies that we can uniquely derive a polynomial of degree m from a set S:

$$S = \{(x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m) \mid x_i \neq x_j \text{ for all indices i and j}\} \tag{3.35}$$

Polynomials therefore have the property that $m+1$ pairs of points (x_i, y_i) for $x_i \neq x_j$ are enough to determine the set of pairs $(x, P(x))$ for all $x \in \mathbb{R}$. This "few too many" property of polynomials is widely used, including in SNARKs. Therefore, we need to understand the method to actually compute a polynomial from a set of points.

If the coefficients of the polynomial we want to find have a notion of multiplicative inverse, it is always possible to find such a polynomial using a method called **Lagrange interpolation**, which works as follows. Given a set like 3.35, a polynomial P of degree m with $P(x_i) = y_i$ for all pairs (x_i, y_i) from S is given by the following algorithm:

check algorithm floating

Example 27. Let us consider the set $S = \{(0,4), (-2,1), (2,3)\}$. Our task is to compute a polynomial of degree 2 in $\mathbb{Q}[x]$ with coefficients from the set of rational numbers $\mathbb{Q}$. Since $\mathbb{Q}$ has multiplicative inverses, we can use method of Lagrange interpolation from Algorithm 4 to

Algorithm 4 Lagrange Interpolation

Require: R must have multiplicative inverses
Require: $S = \{(x_0,y_0),(x_1,y_1),\ldots,(x_m,y_m) \mid x_i,y_i \in R, x_i \neq x_j$ for all indices i and j$\}$
  procedure LAGRANGE-INTERPOLATION(S)
    for $j \in (0\ldots m)$ **do**
      $l_j(x) \leftarrow \Pi_{i=0;i\neq j}^{m} \frac{x-x_i}{x_j-x_i} = \frac{(x-x_0)}{(x_j-x_0)} \cdots \frac{(x-x_{j-1})}{(x_j-x_{j-1})} \frac{(x-x_{j+1})}{(x_j-x_{j+1})} \cdots \frac{(x-x_m)}{(x_j-x_m)}$
    end for
    $P \leftarrow \sum_{j=0}^{m} y_j \cdot l_j$
    return P
  end procedure
Ensure: $P \in R[x]$ with $deg(P) = m$
Ensure: $P(x_j) = y_j$ for all pairs $(x_j,y_j) \in S$

compute the polynomial:

$$
\begin{aligned}
l_0(x) &= \frac{x-x_1}{x_0-x_1}\cdot\frac{x-x_2}{x_0-x_2} = \frac{x+2}{0+2}\cdot\frac{x-2}{0-2} = -\frac{(x+2)(x-2)}{4} \\
&= -\frac{1}{4}(x^2-4) \\
l_1(x) &= \frac{x-x_0}{x_1-x_0}\cdot\frac{x-x_2}{x_1-x_2} = \frac{x-0}{-2-0}\cdot\frac{x-2}{-2-2} = \frac{x(x-2)}{8} \\
&= \frac{1}{8}(x^2-2x) \\
l_2(x) &= \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1} = \frac{x-0}{2-0}\cdot\frac{x+2}{2+2} = \frac{x(x+2)}{8} \\
&= \frac{1}{8}(x^2+2x) \\
P(x) &= 4\cdot(-\frac{1}{4}(x^2-4)) + 1\cdot\frac{1}{8}(x^2-2x) + 3\cdot\frac{1}{8}(x^2+2x) \\
&= -x^2+4+\frac{1}{8}x^2-\frac{1}{4}x+\frac{3}{8}x^2+\frac{3}{4}x \\
&= -\frac{1}{2}x^2+\frac{1}{2}x+4
\end{aligned}
$$

And, indeed, evaluation of P on the x-values of S gives the correct points, since $P(0) = 4$, $P(-2) = 1$ and $P(2) = 3$. Sage confirms this result:

```
sage: Qx = QQ['x']
sage: S=[(0,4),(-2,1),(2,3)]
sage: Qx.lagrange_polynomial(S)
-1/2*x^2 + 1/2*x + 4
```

Example 28. To give another example more relevant to the topics of this book, let us consider the same set as in the previous example, $S = \{(0,4),(-2,1),(2,3)\}$. This time, the task is to compute a polynomial $P \in \mathbb{Z}_5[x]$ from this data. Since we know from example 14 that multiplicative inverses exist in $\mathbb{Z}_5$, algorithm 4 is applicable and we can compute a unique polynomial of degree 2 in $\mathbb{Z}_5[x]$ from S. We can use the lookup tables from (3.24) for computations in $\mathbb{Z}_5$

and get the following:

$$\begin{aligned}
l_0(x) &= \frac{x-x_1}{x_0-x_1}\cdot\frac{x-x_2}{x_0-x_2} = \frac{x+2}{0+2}\cdot\frac{x-2}{0-2} = \frac{(x+2)(x-2)}{-4} = \frac{(x+2)(x+3)}{1} \\
&= x^2+1 \\
l_1(x) &= \frac{x-x_0}{x_1-x_0}\cdot\frac{x-x_2}{x_1-x_2} = \frac{x-0}{-2-0}\cdot\frac{x-2}{-2-2} = \frac{x}{3}\cdot\frac{x+3}{1} = 2(x^2+3x) \\
&= 2x^2+x \\
l_2(x) &= \frac{x-x_0}{x_2-x_0}\cdot\frac{x-x_1}{x_2-x_1} = \frac{x-0}{2-0}\cdot\frac{x+2}{2+2} = \frac{x(x+2)}{3} = 2(x^2+2x) \\
&= 2x^2+4x \\
P(x) &= 4\cdot(x^2+1)+1\cdot(2x^2+x)+3\cdot(2x^2+4x) \\
&= 4x^2+4+2x^2+x+x^2+2x \\
&= 2x^2+3x+4
\end{aligned}$$

And, indeed, evaluation of P on the x-values of S gives the correct points, since $P(0)=4$, $P(-2)=1$ and $P(2)=3$. We can double check our findings using Sage:

```
sage: F5 = GF(5)
sage: F5x = F5['x']
sage: S=[(0,4),(-2,1),(2,3)]
sage: F5x.lagrange_polynomial(S)
2*x^2 + 3*x + 4
```

Exercise 30. Consider modular 5 arithmetic from example 14, and the set $S=\{(0,0),(1,1),(2,2),(3,2)\}$. Find a polynomial $P\in\mathbb{Z}_5[x]$ such that $P(x_i)=y_i$ for all $(x_i,y_i)\in S$.

Exercise 31. Consider the set S from the previous example. Why is it not possible to apply algorithm 4 to construct a polynomial $P\in\mathbb{Z}_6[x]$ such that $P(x_i)=y_i$ for all $(x_i,y_i)\in S$?

Bibliography

Jens Groth. On the size of pairing-based non-interactive arguments. *IACR Cryptol. ePrint Arch.*, 2016:260, 2016. URL `http://eprint.iacr.org/2016/260`.

P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.

David Fifield. The equivalence of the computational diffie–hellman and discrete logarithm problems in certain groups, 2012. URL `https://web.stanford.edu/class/cs259c/finalpapers/dlp-cdh.pdf`.

Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46766-3. URL `https://fmouhart.epheme.re/Crypto-1617/TD08.pdf`.

Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. Cryptology ePrint Archive, Report 2016/492, 2016. `https://ia.cr/2016/492`.