UNIVERSITY OF BIRMINGHAM

School of Computer Science

Autumn Semester 2021

06-35324

Mathematical Foundations of Computer Science

Course notes

Contents

1 The natural numbers

1.1 The laws of arithmetic

For us, the **natural numbers** begin with zero:

$$0, 1, 2, 3, 4, \ldots$$

It is useful to have a symbol for the collection of *all* natural numbers; traditionally, one uses $\mathbb{N}$. Instead of "collection", we also say "set". So $\mathbb{N}$ denotes the set of all natural numbers, starting with zero. Note that this set has *infinitely* many elements.

On the natural numbers we have the basic arithmetic operations of addition and multiplication. The symbol for addition is written "+", the one for multiplication is "×". Some texts use a simple dot to denote multiplication, as in $3 \cdot 4 = 12$. Also, the multiplication symbol is not always written, as in $2x = x + x$.

The two arithmetic operations satisfy certain laws, which are familiar to you from your early school days:

1: The laws of arithmetic

$$\begin{array}{rclrcll} a+0 & = & a & a \times 1 & = & a & \text{(neutral elements)} \\ a+b & = & b+a & a \times b & = & b \times a & \text{(commutativity)} \\ (a+b)+c & = & a+(b+c) & (a \times b) \times c & = & a \times (b \times c) & \text{(associativity)} \\ & & a \times (b+c) = a \times b + a \times c & & & & \text{(distributivity)} \\ & & a \times 0 = 0 & & & & \text{(annihilation)} \end{array}$$

More laws can be *derived* from the ones listed. Here is an example:

2

You can call this a *proof* of the law $(x+y) \times z = x \times z + y \times z$. Many more useful laws can be derived in this way from the ones given in Box 1. You can try to derive some yourself in the style of Box 2; the exercises at the end of this chapter contain some suggestions.

1.2 Beyond equations

The arithmetic operations on the natural numbers have properties that we can not express as a simple universally valid equation. To give an example, here are two important laws:

3: The cancellation laws

If $a + c = b + c$ is true, then $a = b$ is also true.
If $a \times c = b \times c$ is true, and $c \neq 0$, then $a = b$ is also true.

I am sure these rules are as familiar to you as the ones in Box 1 but you would not be able to derive them from those. In fact, soon we will see a situation where the laws of Box 1 hold but cancellation does not.

Cancellation can be used to derive the annihilation law from the others:

4

Here is a statement that is true about the natural numbers, which is even more involved than the cancellation laws:

5: Division with remainder

Given a number $b \neq 0$, every natural number a can (uniquely) be written in the form $m \times b + r$ where m and r are natural numbers and $r < b$ holds.

Is that it, then? Not at all. But you may begin to wonder whether it is possible to write down laws *from which all other laws that hold for the natural numbers can be derived*. This question was tackled by mathematicians towards the end of the 19th century. The most successful answer was given by Giuseppe Peano (1858–1932), and it goes back to origins of the natural numbers in *counting*. Rather than use addition and multiplication straightaway, this theory uses only the operation **successor**, which you could also call "next number", or "$a+1$", or "`a++`"; we will write it as "$s(a)$".

6: The Peano Axioms, part 1

1. 0 is a natural number.
2. If a is a natural number then so is $s(a)$.
3. A number of the form $s(a)$ is always different from 0.
4. If $s(a)$ and $s(b)$ are equal, then a and b are equal.

These look harmless and they are. It is the next (and final) one that makes Peano's approach so powerful:

7: Peano's Axiom of Induction

5. If $P(x)$ is a property of natural numbers that

(ground case)	holds of 0, and	[so $P(0)$ is true]
(inductive step)	holds of $s(x)$ whenever it holds of x,	[so $P(x)$ implies $P(s(x))$]

then P holds of all the natural numbers.

Peano then defined addition by reducing it to counting. Here are his two clauses for addition:

$$\begin{aligned} a+0 &= a \\ a+s(b) &= s(a+b) \end{aligned}$$

Let's use these two equations plus his axioms to prove that his addition is associative, $(a+b)+c = a+(b+c)$:

8

We ask again, are the Peano axioms sufficient to prove every valid property of the natural numbers? The (negative) answer was given by Kurt Gödel (1906–1978), 50 years after Peano published his axioms.

1.3 Place value systems

According to Peano, every natural number can be written as $s(s(s(\ldots(s(0))\ldots)))$ but this would take a lot of space. Instead we use the **place value system**, gifted to us by the Arabs: Given a **base** $b > 1$ and **digits** $0, 1, \ldots, b-1$, we can write every natural number as a string of digits. The value of such a string $d_n\, d_{n-1} \ldots d_0$ is given by adding multiples of powers of b:

$$d_n \times b^n + d_{n-1} \times b^{n-1} + \ldots + d_1 \times b^1 + d_0 \times b^0 \quad \text{or} \quad d_n\, b^n + d_{n-1}\, b^{n-1} + \ldots + d_1\, b + d_0 \text{ for short.}$$

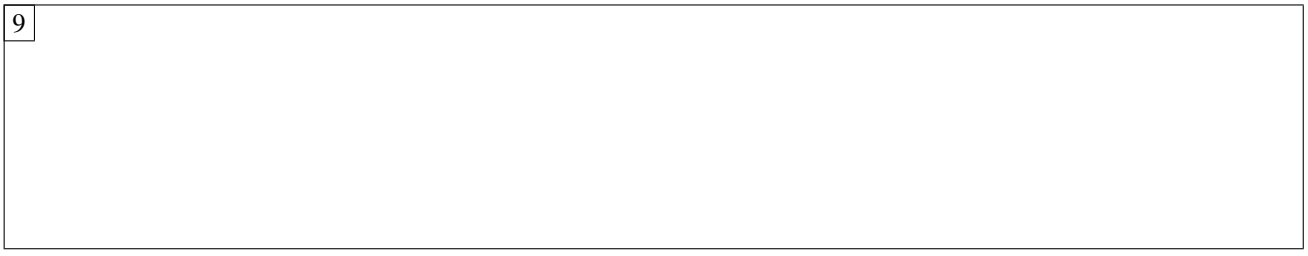
This notation is highly economical because short strings can be used to denote very large numbers. (As an aside, the number of digits required to write out the number a in such a system is "very close" to the **logarithm** of a with regards to base b.) Furthermore, by memorising the sums/products of individual digits we can add/multiply numbers of arbitrary size very efficiently. You spent a good part of your primary school years doing just that.

If we write the expression $d_n \times b^n + d_{n-1} \times b^{n-1} + \ldots + d_1 \times b + d_0$ for the number a slightly differently (using the distributivity law)

$$(d_n \times b^{n-1} + d_{n-1} \times b^{n-2} + \ldots + d_2 \times b + d_1) \times b + d_0$$

then we see that the lowest-value digit d_0 is obtained from a as the *remainder* from dividing a by b, as expressed in Box 5 above. We can then continue the process with the quotient $d_n \times b^{n-1} + d_{n-1} \times b^{n-2} + \ldots + d_2 \times b + d_1$ etc and get all the digits of a for the base b representation. As an example let us represent 59 in base $b = 7$:

9



and we get the representation 113 in base 7, sometimes written as 113_7.

1.4 Natural number representation in a computer

A computer knows nothing about numbers (or anything else for that matter). How do we use its hardware to represent the mathematical concept of a natural number? A common unit in a cpu is a register of 32 bits, where each bit can be in one of two states, traditionally called "0" and "1". Thus a register can be in any one of 2^{32} different states, depending on the state of each of its 32 bits. It is now up to us to interpret these "bit patterns" as natural numbers. The usual way for doing so is to view the bit pattern as the digit representation of a number in base 2. An example:

10

This works very well and there is circuitry (in the ALU, or arithmetic logical unit) to implement addition, multiplication, and a number of other elementary operations. However, we have a big problem arising from the fact that registers are of a fixed size, that is, there are only 32 bits to represent binary digits. What happens when the result of an operation requires more than 32 digits? The answer has two parts:

1. The excess digits are available for only a short moment in the cpu.
2. Java ignores them.

The consequence of this is that the answer you get from an arithmetic operation on Java `int` variables is only correct up to multiples of 2^{32}. Because of this, the numbers available in a cpu are better imagined as sitting on a circle rather than on the number line you are familiar with from school:

11

Nevertheless, and this is extremely important, the arithmetic laws listed in Box 1 are still valid for the 2^{32} numbers we do have. This means that the algebraic manipulations you learned in school can still be applied when writing computer programs, and will not change the outcome of any calculations. However, the fact remains that the answer from a calculation can be off by multiples of 2^{32}.

Finally, Java does not actually support this "sign-less" interpretation of bit patterns, but its "ancestor" C does. When you define a variable as `unsigned int` in C, then you get a block of memory consisting of four bytes (so 32 bits) the contents of which are interpreted exactly as described above.

Exercises

1. Derive the equation $(x+y)^2 = x^2 + 2xy + y^2$ from the basic laws of arithmetic in the style of Box 2. (Recall that x^2 is just shorthand for $x \times x$.)
2. Prove by induction that the sum $2^0 + 2^1 + 2^2 + \cdots + 2^a$ equals $2^{a+1} - 1$.
3. Write out the number 2020 in base 3.
4. What is the largest number that can be represented in a 32-bit `unsigned int` variable?
5. Which of the Peano axioms is *not* valid for 32-bit unsigned integers?

Practical advice

In the exam, I expect you to be able to

- derive an equation from the basic laws of arithmetic in the style of Box 2;
- prove a statement about natural numbers by induction;
- represent a natural number in an arbitrary base.

I will also expect you

- to know the notation $\mathbb{N}$ for the set of natural numbers;
- to understand the consequences of computer arithmetic being limited to numbers that can be represented as 32 binary digits.

However, I will **not** expect you to memorise the various laws discussed in this chapter; if needed, they will be given on the exam question paper.

More information

Most textbooks on *Discrete Mathematics* start with a discussion of the natural numbers. You will find a wide choice of such books in the University Library. If there is one that you like particularly well, then you may want to consider purchasing a copy for yourself.

At the end of Part 1.2 I mentioned Gödel's result about the impossibility to axiomatise the natural numbers, his "First Incompleteness Theorem". You could have a look at `https://en.wikipedia.org/wiki/G%C3%B6del's_incompleteness_theorems` and `https://plato.stanford.edu/entries/goedel-incompleteness/` to get a glimpse of what this is all about, and why Gödel's work became so famous.

1 The natural numbers

1: The laws of arithmetic

$$\begin{array}{rclrcll} a+0 & = & a & a\times 1 & = & a & \text{(neutral elements)}\\ a+b & = & b+a & a\times b & = & b\times a & \text{(commutativity)}\\ (a+b)+c & = & a+(b+c) & (a\times b)\times c & = & a\times(b\times c) & \text{(associativity)}\\ & & a\times(b+c)=a\times b+a\times c & & & & \text{(distributivity)}\\ & & a\times 0=0 & & & & \text{(annihilation)} \end{array}$$

2

$$\begin{array}{rcll} (x+y)\times z & = & z\times(x+y) & \text{(commutativity of } \times)\\ & = & z\times x+z\times y & \text{(distributivity)}\\ & = & x\times z+y\times z & \text{(commutativity of } \times) \end{array}$$

3: The cancellation laws

If $a+c=b+c$ is true, then $a=b$ is also true.
If $a\times c=b\times c$ is true, and $c\neq 0$, then $a=b$ is also true.

4

$$\begin{array}{rcll} a\times 0 & = & a\times(0+0) & \text{(0 is a neutral element for } +)\\ & = & a\times 0+a\times 0 & \text{(distributivity)} \end{array}$$

Using the neutrality of 0 once more on the left-hand side, we have thus derived

$$0+a\times 0=a\times 0+a\times 0$$

Applying cancellation for addition, we obtain $0=a\times 0$, the annihilation law.

5: Division with remainder

Given a number $b\neq 0$, every natural number a can (uniquely) be written in the form $m\times b+r$ where m and r are natural numbers and $r<b$ holds.

6: The Peano Axioms, part 1

1. 0 is a natural number.
2. If a is a natural number then so is $s(a)$.
3. A number of the form $s(a)$ is always different from 0.
4. If $s(a)$ and $s(b)$ are equal, then a and b are equal.

7: Peano's Axiom of Induction

5. If $P(x)$ is a property of natural numbers that

(ground case)	holds of 0, and	[so $P(0)$ is true]
(inductive step)	holds of $s(x)$ whenever it holds of x,	[so $P(x)$ implies $P(s(x))$]

then P holds of all the natural numbers.

8

We use induction over c, by which I mean that $P(x)$ is the statement, "For all a, b natural numbers, $(a+b)+x = a+(b+x)$."

(ground case) $P(0)$ is the statement $(a+b)+0 = a+(b+0)$ and this is true by using the first defining equation for addition on both sides to obtain $a+b$.

(inductive step) Let's assume that $P(c)$ is true. This means that $(a+b)+c = a+(b+c)$ for some fixed number c. On this basis, let us argue that $(a+b)+s(c) = a+(b+s(c))$ is also true. Using the second defining equation for addition, we can re-write the left-hand side as follows:

$$(a+b)+s(c) = s((a+b)+c)$$

Using the induction hypothesis $P(c)$, we can re-write further:

$$= s(a+(b+c))$$

On the right hand side we use the second defining equation for addition twice:

$$a+(b+s(c)) = a+s(b+c) = s(a+(b+c))$$

and we see that we arrive at the same expression as we did on the left-hand side.

9

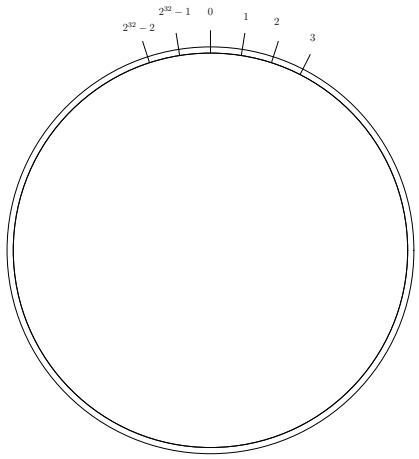
$$\begin{aligned} 59 &= 8\times 7+3 && \text{so } d_0 = 3 \\ 8 &= 1\times 7+1 && \text{so } d_1 = 1 \\ 1 &= 0\times 7+1 && \text{so } d_2 = 1 \end{aligned}$$

10

0	0	$\cdots$	0	1	1	0	1

is interpreted as $1101_2 = 13_{10}$

11



Answers to exercises

1.

$$
\begin{array}{rcll}
(x+y)^2 & = & (x+y)\times(x+y) & \text{(definition of } ()^2\text{)} \\
 & = & (x+y)\times x+(x+y)\times y & \text{(distributivity)} \\
 & = & x\times x+y\times x+x\times y+y\times y & \text{(distributivity as shown in Box 2)} \\
 & = & x\times x+x\times y+x\times y+y\times y & \text{(commutativity of } \times\text{)} \\
 & = & x\times x+1\times(x\times y)+1\times(x\times y)+y\times y & \text{(1 is neutral for } \times\text{)} \\
 & = & x\times x+(1+1)\times(x\times y)+y\times y & \text{(distributivity)} \\
 & = & x\times x+2\times(x\times y)+y\times y & \text{(2 is an abbreviation for } 1+1\text{)} \\
 & = & x^2+2\times(x\times y)+y^2 & \text{(definition of } ()^2\text{)}
\end{array}
$$

2. **Base case.** $a=0$: Need to show that $2^0=2^1-1$ which is indeed true: both sides evaluate to 1.

   Inductive step. We *assume* that $2^0+2^1+2^2+\cdots+2^a$ equals $2^{a+1}-1$ for some given $a\in\mathbb{N}$.

   We *want to show* that $2^0+2^1+2^2+\cdots+2^a+2^{a+1}$ equals $2^{a+1+1}-1$.

   Argument: The left-hand side equals $(2^{a+1}-1)+2^{a+1}$ by *induction assumption*. This rewrites to $2\times 2^{a+1}-1$ and this is $2^{a+2}-1$, which is the right-hand side we desire.

3. 2202211

4. In binary, this number is $1111,1111,1111,1111,1111,1111,1111,1111$ (32 ones). In decimal, this is $2^{31}+2^{30}+\ldots+2^1+2^0$ and this is, according to Exercise 2, $2^{32}-1$, or $4,294,967,295$.

5. The third one which says that $s(n)$ is always different from zero. For fixed size computer integers this is not true. If you add 1 to $4,294,967,295$ then you get zero.

2 The integers

2.1 The arithmetic laws of integers

With the integers we have *negative numbers* as well as all the natural numbers from before:

$$\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$$

We denote the collection (or set) of all integers with $\mathbb{Z}$.

As before, we try to express the properties of the integers in precise form, so that we can compare the mathematical concept with integers on a computer. And also as before, we try to be as economical as possible. With this in mind, we don't try to write down the properties of subtraction but just the those of the **negative** $-a$ of an integer a. There is just one law needed (in addition to those of Box 1):

12: The law of negation

$$a + (-a) \quad = \quad 0 \qquad \text{(additive inverse)}$$

That's very economical indeed! Can we really derive all the other algebraic rules about negation and subtraction from this one axiom? Let's see. First we use the new law to show that cancellation for addition is true for the integers (whereas we had to state it explicitly for the natural numbers):

13

By the calculation in Box 4, we now get annihilation for free. Using again cancellation we can derive the "law of double negation":

14

Subtraction $a - b$ can be introduced as a shorthand for $a + (-b)$ and the usual rules for subtraction can be easily derived. What isn't quite so easy, is the rule "minus times minus is plus", or as an equation $(-a) \times (-b) = a \times b$. Let's see whether we can get it from the rules we have stated. We begin with the simpler equation $a \times (-b) = -(a \times b)$:

15

Using this law twice gives us the desired result: $(-a)\times(-b) = -((-a)\times b) = -(b\times(-a)) = -(-(b\times a)) = b\times a = a\times b$.

2.2 Rings

It is not uncommon to have operations interacting in the way addition, negation, and multiplication do on the integers. This is the reason why mathematicians have introduced a special name for this situation. They say that the integers form a **ring**. Let's write out the laws of rings all in one place:

16: The laws of rings

$$
\begin{array}{rclrcll}
a+0 & = & a & a\times 1 & = & a & \text{(neutral elements)} \\
a+b & = & b+a & a\times b & = & b\times a & \text{(commutativity)} \\
a+(-a) & = & 0 & & & & \text{(additive inverse)} \\
(a+b)+c & = & a+(b+c) & (a\times b)\times c & = & a\times(b\times c) & \text{(associativity)} \\
 & & a\times(b+c) = a\times b + a\times c & & & & \text{(distributivity)}
\end{array}
$$

There are many variations on this theme. Sometimes, one lacks a neutral element for multiplication, so then people speak of "unital rings" when there is one. At other times, multiplication is not commutative, so by saying "commutative ring" we can emphasise that it is. Finally, the integers also satisfy the the cancellation rule for multiplication (which we stated in Box 3) and this can be highlighted by speaking of a "cancellative ring". All together, then, the integers form a *unital commutative cancellative ring*.

You may wonder whether there is something like Peano's axioms for the integers which, after all, were both simpler *and* more powerful than the arithmetic laws on the natural numbers. The answer is no. On the other hand, it *is* possible to construct the integers from the natural numbers, and then their ring properties can be *proven* rather than postulated as we did here. We'll see a glimpse of this approach later in the course.

2.3 Integers in computers

Given a computer register of 32 bits, how can we use the 2^{32} available bit patterns to represent negative as well as positive numbers? A first thought might be to reserve the highest value bit for representing the sign of the number and the remaining 31 bits for representing a natural number as before. This, however, has several disadvantages. Firstly, it would give us two versions of zero: $+0$ and -0. Secondly, the implementation of the arithmetic operations would require new circuitry, as we would have to take the sign bit into account.

The mathematician and computer pioneer John von Neumann (1903-1957) realised in 1945 that there is a much better coding scheme. It makes a virtue out of the (otherwise irritating) phenomenon of overflow. The idea is that if multiples of 2^{32} have no effect on the pattern in the register, then we can freely add them to stay with positive numbers *at all times*. As an example, instead of trying to define a special pattern for -1, one simply works with the representation of the positive number $2^{32}-1$. This trick should be familiar from the way we read a traditional clock: Instead of "11 o'clock" we also say "one hour before noon", etc.

It is now easily seen that this alteration of the inputs to an arithmetic operation affects the result only by adding or subtracting a multiple of 2^{32}, in other words, it is invisible to us since we only keep the 32 lower bits anyway. It's good to see how this works in an example:

17

It is entirely up to us to decide which bit patterns should correspond to negative numbers under this translation. The common approach is to take those bit patterns as negative for which the highest value bit equals 1.

18

As with the natural numbers and the unsigned integers in C, we now find that the bit patterns as described above satisfy all the ring laws. The cancellation law for multiplication, on the other hand, holds for the mathematical integers but not for computer integers.

Numbers in Java. When we declare a variable to be of type `int` in Java then we are committing ourselves to the integer interpretation of bit patterns just described. The range is from -2^{31} to $2^{31}-1$. Alternatively we can ask for a `long` in which case 64 bit registers are used. The principles of the representation remain the same, however, so the range is now from -2^{63} to $2^{63}-1$.

If we need integers outside these ranges then we can use the Java class `BigInteger` which accommodates integers of (almost) unbounded size.

2.4 Modulo arithmetic

The limited size of cpu registers forced us to adopt an approach that ignores multiples of 2^{32}. In fact, this method is very general. Given any "**modulus**" $m \neq 0$ we can consider numbers "equal up to multiples of m", or as this is more commonly called, "congruent mod m". For example, modulo 7 there are only seven different numbers:

19

In order to calculate modulo m we can choose representatives that are most convenient for the task at hand. For example, modulo 7 we have

20

The ring laws are all valid when we consider numbers modulo some fixed number m. We get what is called **modulo arithmetic** or the **ring** $\mathbb{Z}_m$ (which in general is not cancellative). It is a fundamental tool in mathematics and also in cryptography, where m is typically a very large number, say with 500 decimal digits. To implement cryptographic routines in Java we therefore definitely need the `BigInteger` class.

Exercises

1. Using the definition of subtraction $a - b = a + (-b)$, derive the law $a \times (b - c) = a \times b - a \times c$.
2. Find an example that illustrates that multiplication is *not* cancellative for 32-bit integers.
3. Modulo 2 there are only two numbers, 0 and 1. How do addition and multiplication work for these?

Practical advice

In the exam, I expect you to be able to

- derive a statement about integers from the basic ring laws;
- perform a computation in a ring $\mathbb{Z}_m$.

I will expect you to know

- the notations $\mathbb{Z}$ and $\mathbb{Z}_m$;
- that integer arithmetic in Java is performed modulo 2^{32} in the case of `int` and modulo 2^{64} in the case of `long`.

I do **not** expect you to memorise the ring laws.

More information

Negative numbers are a relatively recent invention in Western mathematics; you can read up on the history of this concept on Wikipedia, `https://en.wikipedia.org/wiki/Negative_number`.

Many textbooks on *Discrete Mathematics* show how the integers can be constructed from the natural numbers. It is not too difficult and we will see some elements of this later in the course.

2 The integers

12: The law of negation

$$a+(-a) \;=\; 0 \qquad \text{(additive inverse)}$$

13

We assume we have integers a, b, c such that $a+c = b+c$.
Then we can add $-c$ on both sides to get $(a+c)+(-c) = (b+c)+(-c)$.
This can be written as $a+(c+(-c)) = b+(c+(-c))$ because addition is associative.
Using the law of the additive inverse, we get $a+0 = b+0$ and using the fact that 0 is a neutral element, we are finished: $a = b$.

14

We have $a+(-a) = 0$ by the law of the additive inverse.
We also have $(-a)+(-(-a)) = 0$, by invoking the law of the additive inverse for $-a$.
Putting these two equations together, we obtain $a+(-a) = (-a)+(-(-a))$.
Cancellation allows us to remove $(-a)$ on both sides and we get $a = -(-a)$.

15

The trick is to use cancellation again:
On the one hand, we have $a \times b + (-(a \times b)) = 0$ by the basic property of the additive inverse.
On the other hand, we have $a \times b + a \times (-b) = a \times (b+(-b)) = a \times 0 = 0$ by distributivity, inverse, and annihilation.
These two equations yield $a \times b + (-(a \times b)) = a \times b + a \times (-b)$ and cancelling $a \times b$ on both sides gives us the desired equality.

16: The laws of rings

$a+0$	$=$	a		$a \times 1$	$=$	a	(neutral elements)
$a+b$	$=$	$b+a$		$a \times b$	$=$	$b \times a$	(commutativity)
$a+(-a)$	$=$	0					(additive inverse)
$(a+b)+c$	$=$	$a+(b+c)$		$(a \times b) \times c$	$=$	$a \times (b \times c)$	(associativity)
		$a \times (b+c) = a \times b + a \times c$					(distributivity)

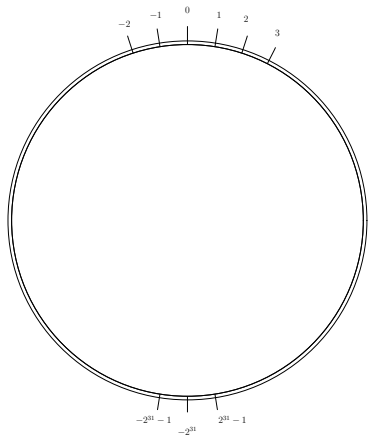
17

$6-1$ becomes $6+(2^{32}-1) = 2^{32}+(6-1) = 2^{32}+5$

In the register:

	0	…	0	0	1	1	0
	1	…	1	1	1	1	1
1	0	…	0	0	1	0	1

18



19

$$\begin{array}{cccccccccccc}
\ldots & -14 & \equiv & -7 & \equiv & 0 & \equiv & 7 & \equiv & 14 & \equiv & \ldots \\
\ldots & -13 & \equiv & -6 & \equiv & 1 & \equiv & 8 & \equiv & 15 & \equiv & \ldots \\
 & & & & & \vdots & & & & & & \\
\ldots & -8 & \equiv & -1 & \equiv & 6 & \equiv & 13 & \equiv & 20 & \equiv & \ldots
\end{array}$$

20

$$6^{10} \equiv (-1)^{10} = ((-1)^2)^5 = 1^5 = 1$$

Answers to exercises

1.

$$\begin{array}{lcll}
a \times (b-c) & = & a \times (b+(-c)) & \text{(definition of } -\text{)} \\
 & = & a \times b + a \times (-c) & \text{(distributivity)} \\
 & = & a \times b + (-a \times c) & \text{(as shown in Box 15)} \\
 & = & a \times b - a \times c & \text{(definition of } -\text{)}
\end{array}$$

2. Choose $a = 2^{12}$, $b = 2^{13}$, and $c = 2^{20}$. Then $a \times c = 2^{32}$ which is the same as zero for `int` integers. Likewise, $b \times c = 2^{33} = 2 \times 2^{32}$ and this is also zero. So we have $a \times c = b \times c$ but $a \neq b$.

3.

$+$	0	1
0	0	1
1	1	0

$\times$	0	1
0	0	0
1	0	1

3 The rational numbers

3.1 Fractions

You are familiar with fractions from your school days; they are pairs of integers $\frac{a}{b}$, where $b \neq 0$, expressing the idea of a number a of entities shared equally among b individuals. Fractions figure prominently in the oldest mathematical texts that have survived from ancient times, the Rhind Papyrus (kept in the British Museum) and the Moscow Mathematical Papyrus. So unlike negative numbers, fractions appear very early in the historical record.

In school you also learned how to perform the arithmetic operations with fractions, for example, the rules for addition and multiplication are

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+cb}{bd} \quad \text{and} \quad \frac{a}{b}\times\frac{c}{d}=\frac{ac}{bd}$$

We also have a zero: $\frac{0}{1}$ and a one: $\frac{1}{1}$, and these behave as neutral elements for addition and multiplication, respectively. The ring laws can now be checked and they are all valid — or are they? Actually, no, the distributivity law almost always fails:

21

What's going on?

3.2 The rational numbers

Well, what is going on is that the same *ratio* can be expressed in many different ways: Whether we share one chocolate bar among two friends, or share two among four gives the same ratio; everyone gets half a bar. Mathematically we say that the two fractions $\frac{1}{2}$ and $\frac{2}{4}$ represent the same *rational number*. Here is a simple test to check whether two fractions represent the same ratio:

$$\frac{a}{b}=\frac{c}{d} \quad \text{if and only if} \quad a\times d = b\times c$$

We can use it to check that the two fractions we got from testing the distributivity law are equal as rational numbers:

22

Now we look again at all the ring laws and we find that the rational numbers, usually denoted by $\mathbb{Q}$, do indeed form a ring. In fact, rational numbers have a further property, namely, every rational $q \neq 0$ has a **multiplicative inverse** q^{-1}. If q is given as the fraction $\frac{a}{b}$ then the inverse is $\frac{b}{a}$ since $\frac{a}{b}\times\frac{b}{a}=\frac{ab}{ab}=\frac{1}{1}$.

Whenever a (commutative unital) ring has a multiplicative inverse for every element other than zero, we speak of a **field**. For ease of reference we list all the laws again:

23: The field laws

$a+0$	$=$	a	$a\times 1$	$=$	a	(neutral elements)
$a+b$	$=$	$b+a$	$a\times b$	$=$	$b\times a$	(commutativity)
$(a+b)+c$	$=$	$a+(b+c)$	$(a\times b)\times c$	$=$	$a\times(b\times c)$	(associativity)
$a+(-a)$	$=$	0	$a\times a^{-1}$	$=$	$1\ (a\neq 0)$	(inverses)
		$a\times(b+c)=a\times b+a\times c$				(distributivity of $\times$ over $+$)

In Box 13 we showed that the cancellation law for addition is automatically valid once we have an additive inverse for every number; essentially the same derivation shows that the cancellation law for multiplication follows from the existence of multiplicative inverses (for non-zero numbers).

3.3 A normal form and the Euclidean algorithm

Among the infinitely many fractions that represent the same rational number, there is one which is particularly simple. It is the fraction $\frac{a}{b}$ where a and b are as small as possible, and b is positive. We call this simplest fraction **reduced** or a **normal form** for the corresponding rational number.

Given any fraction $\frac{a}{b}$ we can compute the normal form by cancelling out common factors from a and b, and by replacing $\frac{a}{b}$ with $\frac{-a}{-b}$ should b be negative. To find the largest common factor $\mathsf{lcf}(a,b)$ of two natural numbers a and $b \neq 0$, we employ **Euclid's algorithm**.[1] It starts by writing a as $m \times b + r$ with $0 \leq r < b$, which is always possible as we stated in Box 5. If it turns out that $r = 0$, then b divides a and clearly b is the largest common factor of the two. If $r \neq 0$ then we repeat the process with b and r. Since $r < b$, the process will eventually end.

Using pseudocode instead of words, here is the algorithm again. It uses "registers" `x` and `y` to hold the (changing) pair of numbers of which we want to compute the largest common factor:

```
x <- a
y <- b
while ( y != 0 ) {
   r <- x mod y
   x <- y
   y <- r }
return x
```

Why does this work? For this we consider a **loop invariant**, i.e., a logical statement that is true at the beginning of the `while`-loop and also after each of its iterations:

$$\mathsf{lcf}(x,y) = \mathsf{lcf}(a,b)$$

It holds at the start of the `while`-loop because at that point we just initialised `x` with `a` and `y` with `b`. Assuming now that it holds at the start of the execution of the body, we want to show that it holds afterwards, in other words, we need to prove that

$$\mathsf{lcf}(x,y) = \mathsf{lcf}(y,r) \quad \text{where} \quad x = m \times y + r$$

Here is the argument:

24

There is also a **loop variant**, i.e., something which changes in each iteration of the `while`-loop, namely, the variable y; each time the body of the loop is executed, y is replaced with $r = x \bmod y$ and by Box 5 (on page 4) the new value (the remainder of the division of x by y) is strictly smaller than y. The same theorem tells us that y is always greater than or equal to zero, so the sequence of values stored in the variable y is strictly decreasing but bounded from below by zero. It must therefore become equal to zero after finitely many iterations. In other words, the `while`-loop will eventually stop.

3.4 The extended Euclidean algorithm and finite fields

With a little bit more book-keeping we can obtain a very useful statement about the largest common factor, namely, that it can be written as a *linear combination* of the original numbers a and b (named after Étienne Bézout, 1730-1783.):

[1] Named after the ancient Greek mathematician Euclid who lived around 300 BC.

Theorem 1 (Bézout's identity) *Let a and b be natural numbers not both of which are zero. Then there exist integers u and v such that*

$$\mathsf{lcf}(a,b) = u \times a + v \times b\ .$$

The proof is *constructive* in that we give an algorithm to compute u and v, an extension of Euclid's algorithm:

```
x <- a
y <- b
u_x <- 1; v_x <- 0
u_y <- 0; v_y <- 1
while ( y != 0 ) {
   r <- x mod y; k <- x div y
   u <- u_x; v <- v_x
   u_x <- u_y; v_x <- v_y
   u_y <- u - k*u_y; v_y <- v - k*v_y
   x <- y
   y <- r }
return x, u_x, v_x
```

Here is an example run of this algorithm, for $a = 288$ and $b = 150$. We record the contents of each variable at the beginning of the `while`-loop:

25

run	x	y	u_x	v_x	u_y	v_y
0	288	150	1	0	0	1
1						
2						
3						
4						

We obtain $\mathsf{lcf}(288,150) = 6 = 12 \times 288 - 23 \times 150$.

As an exercise you may now show that the following is an invariant for the `while`-loop of the extended algorithm:

$$x = u_x \times a + v_x \times b \qquad \text{and} \qquad y = u_y \times a + v_y \times b\ .$$

Let us now use Bézout's identity to show that $\mathbb{Z}_m$ is actually a field whenever m is a prime number. Remember that we already know that $\mathbb{Z}_m$ is a ring, and the difference between a ring and a field is that in the latter every non-zero element a has a multiplicative inverse, that is, there is an element b such that $a \times b \equiv 1 \bmod m$. The proof is very simple:

26

As we demonstrated, the multiplicative inverse in such a **finite field** can be calculated efficiently with the extended Euclidean algorithm. For small prime numbers m we can also read them off the multiplication table of $\mathbb{Z}_m$. Let's look at two examples and one non-example:

27

It's important to remember that the multiplicative inverse that exists for every $a \neq 0$ in a finite field $\mathbb{Z}_m$, has *nothing at all* to do with the inverse $\frac{1}{a}$ in the field $\mathbb{Q}$.

For $m = 2$ we get the field $\mathbb{Z}_2$, with its two elements 0 and 1 (which we could also call "even" and "odd"). It is also called GF(2) , or "**Galois field** with two elements" in honour of Évariste Galois (1811–32). It is obviously of great interest to computer scientists.

3.5 Fractions and rational numbers on a computer

It is easy to represent fractions in Java; all we need to do is define a class such that every instance of that class has two `BigInteger` variables in which to store numerator and denominator. The arithmetic operations can be defined as methods along the lines indicated in Section 3.1. The test for equality becomes a method with result type `boolean`, and in it we employ the criterion discussed in Section 3.2. We should also have a method `normalise` which uses the Euclidean algorithm to reduce the fraction stored in the class instance.

The implementation suggested here is an example of an **abstract datatype**. On the outside, objects in this class behave like *rationals* while the implementation (invisible to the programmer) works with *fractions*.

Exercises

1. Let $\frac{a}{b}$ be a fraction which is *not* reduced, which means that $\mathsf{lcf}(a,b) > 1$. Let $\frac{c}{d}$ be any fraction. Show that $\frac{a}{b} + \frac{c}{d}$ is also not reduced.
2. What happens in the first round of the Euclidean algorithm when the initial values a and b are such that $a < b$?
3. Find the multiplicative inverse of 4 in the finite field $\mathbb{Z}_{17}$.

Practical advice

In the exam, I expect you to be able to

- prove a statement that follows from the field laws;
- find the multiplicative inverse in a finite field $\mathbb{Z}_m$ where m is prime.

I will also expect you to know

- the difference between fractions and rational numbers;
- the notation $\mathbb{Q}$ for the rational numbers.

More information

Finite rings and finite fields $\mathbb{Z}_m$ are central to cryptography. Every book on cryptography starts with a discussion of their (many and fascinating) properties. Here is a taster, known as *Fermat's Little Theorem*: If $\mathbb{F}$ is a finite field with m elements and $a \neq 0$ is an element of $\mathbb{F}$, then $a^{m-1} = 1$. (Check it in $\mathbb{Z}_5$.)

Mathematics books on the topic are not so easy to read, unfortunately, since fields appear only after a lengthy (albeit useful) discussion of "groups" (yet another concept) and rings.

3 The rational numbers

21

$$(\frac{1}{2}+\frac{3}{4})\times\frac{5}{6}=\frac{10}{8}\times\frac{5}{6}=\frac{50}{48} \quad \text{but} \quad \frac{1}{2}\times\frac{5}{6}+\frac{3}{4}\times\frac{5}{6}=\frac{5}{12}+\frac{15}{24}=\frac{300}{288}$$

22

$$50\times 288=14400=48\times 300 \quad \text{so indeed} \quad \frac{50}{48}=\frac{300}{288}$$

23: The field laws

$a+0$	$=$	a	$a\times 1$	$=$	a	(neutral elements)
$a+b$	$=$	$b+a$	$a\times b$	$=$	$b\times a$	(commutativity)
$(a+b)+c$	$=$	$a+(b+c)$	$(a\times b)\times c$	$=$	$a\times(b\times c)$	(associativity)
$a+(-a)$	$=$	0	$a\times a^{-1}$	$=$	$1\ (a\neq 0)$	(inverses)
		$a\times(b+c)=a\times b+a\times c$				(distributivity of $\times$ over $+$)

24

We show that any number that divides both x and y, also divides y and r, and vice versa.

If p is a factor of both x and y, then we can write $x=s\times p$ and $y=t\times p$ for some natural numbers s and t.

For $r=m\times y-x$ this gives us $r=m\times t\times p-s\times p=(m\times t-s)\times p$ and we see that p is also a factor of r. (We already know that it is a factor of y.)

The argument for the other direction works in the same way, every factor of both y and r is also a factor of x.

25

run	x	y	u_x	v_x	u_y	v_y
0	288	150	1	0	0	1
1	150	138	0	1	1	-1
2	138	12	1	-1	-1	2
3	12	6	-1	2	12	-23
4	6	0	12	-23	-25	48

We obtain $\mathsf{lcf}(288,150)=6=12\times 288-23\times 150$.

26

Because m is prime and $a\not\equiv 0$, $\mathsf{lcf}(a,m)=1$.

By Bézout's identity, there are integers u and v such that $1=u\times a+v\times m$.

If we look at the identity modulo m, then it reads $1\equiv u\times a$, and we see that u is the multiplicative inverse of a that we were after.

27

$m = 2$

$\times$	0	1
0	0	0
1	0	1

$m = 5$

$\times$	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

$m = 4$

$\times$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

$a = 2$ has no inverse

Answers to exercises

1. If $\frac{a}{b}$ is not reduced then a and b contain a common factor $k > 1$, so we can write $a = k \times a'$ and $b = k \times b'$. Then we get
$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd} = \frac{k \times a' \times d + k \times b' \times d}{k \times b' \times d} = \frac{k \times (a' \times d + b' \times d)}{k \times (b' \times d)}$$
and we see that the same common factor k appears in the sum $\frac{a}{b} + \frac{c}{d}$.
2. Not much: a and b get exchanged so that b is in register x and a is in register y.
3. The answer is 13 as $4 \times 13 = 52 = 1$ in $\mathbb{Z}_{17}$.

4 The real numbers

4.1 The reals and scientific notation

Rationals are not very useful as a numeric datatype because numerator and denominator tend to grow large very rapidly as the computation progresses. Another problem is that not every mathematically interesting function yields rational results, for example, $\sqrt{2}$ is *irrational*, that is, it can not be represented as a fraction (a fact already known to the ancient Greeks).

In mathematics we address this problem by extending the number system once again, from the rationals to the reals. We think of the reals as infinite expansions

$$d_n d_{n-1} \ldots d_1 d_0 \bullet d_{-1} d_{-2} \ldots$$

which are a shorthand for *infinite sums* of powers of the base

$$d_n \times b^n + d_{n-1} \times b^{n-1} + \ldots + d_1 \times b + d_0 + d_{-1} \times b^{-1} + d_{-2} \times b^{-2} \ldots$$

This is a good representation except that there are real numbers which have more than one representation: $1.000\ldots$ is the same real number as $0.999\ldots$, but these identifications are not as irritating as the ones between fractions. The set of all real numbers is denoted with $\mathbb{R}$. Like $\mathbb{Q}$, the real numbers form a field.

In practice we don't have the patience to write out all the infinitely many digits of a real number, so we have to make do with finitely many of them, for example, we work with 3.1416 as an approximation to π. It may be tempting to think of such a finite approximation as representing a decimal fraction, for example, $3.1416 = \frac{31416}{10000}$, but this would obscure the fact that we obtained 3.1416 from the real π by rounding. If we don't know the actual π but trust that someone did the rounding properly, then we know that π lies somewhere between 3.14155 and 3.14165. Thus we should think of finite decimal expansions as representing *intervals* of reals, not actual numbers.

Every finite decimal expansion (except zero) can be written in the form

$$d_0 \bullet d_{-1} d_{-2} \ldots d_{-m} \ \times \ b^n$$

where $d_0 \neq 0$ and n is an integer. This is often shortened to

$$d_0 \bullet d_{-1} d_{-2} \ldots d_{-m} \, \mathrm{E}n$$

For example, Avogadro's constant is 6.022 E23. This way of writing (approximations to) real numbers is called **scientific notation** where the part in front of E is called **mantissa** and the part following E is the **exponent**.

4.2 Floating point numbers

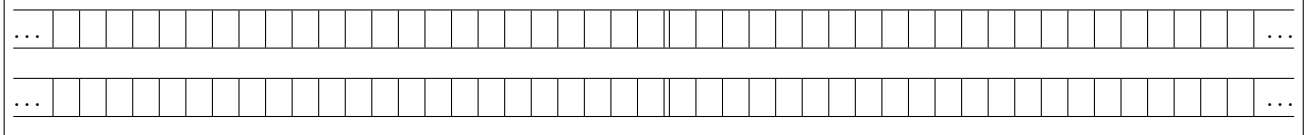
It is now a small step from scientific notation to floating point numbers stored in registers of 32 bits:

- We use base 2 for both mantissa and exponent.
- Since always $d_0 \neq 0$, we know that $d_0 = 1$, so this part of the mantissa does not need to be stored, only the digits following the binary point. (We have to find a representation for zero, though.)
- We fix the number of binary digits after the binary point to 23.
- We use 8 bits for the exponent which we interpret as an unsigned integer from which 127 has to be subtracted. (In other words, we don't use the clever encoding of integers that is employed for `int` variables.)
- We use 1 bit to indicate the sign.
- We pack the three parts in the order sign–exponent–mantissa into 32 bit registers.

There are more aspects to this representation, which you can read up on by searching for IEEE-754, the universally accepted standard for floating point representation. Rather than delving into these details, I want to present a visualisation for floating point numbers. For this we imagine a fixed scale of 256 positions in the middle of which we place the binary point:

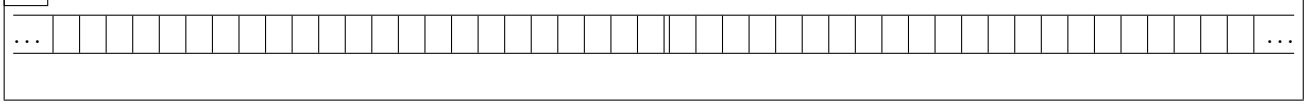
The mantissa always has 24 bits; where those 24 bits are located on the scale is determined by the exponent. However, to make the pictures more manageable, let's pretend that the mantissa has only 8 bits. As an example, we fill in the floating point approximation to $\frac{1}{5}$, and below it the same for $8 \times \frac{1}{5}$:

28

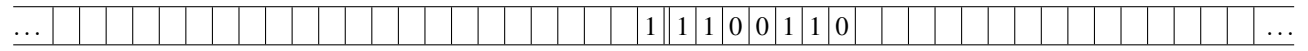


If we apply our usual algorithm for addition, then we get

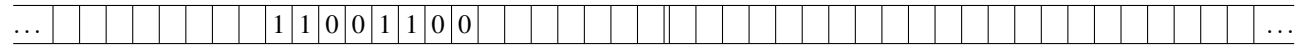
29



but now there are *too many digits*. We must cut back to the maximum that is allowed for a mantissa, in our illustration 8 bits. This is done by rounding:

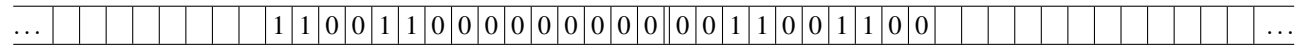


Almost every arithmetic operation requires rounding at the end. Even worse, if a is a large number and b is small, then their sum equals a again since all the digits of b are lost to rounding. Here is an illustration of this:



plus

equals



which rounds to

As a consequence, almost *no arithmetic laws* are valid for the floating point numbers.

Rather than focus on the *digit representation* of floating point numbers, we can also try to visualise their *value* on the number line. Given a fixed exponent n, recall that the mantissa can hold 2^{23} different patterns, representing the numbers

$$1.000\ldots0\mathrm{E}n \quad \text{to} \quad 1.111\ldots1\mathrm{E}n$$

The "next" floating point number after $1.111\ldots1\mathrm{E}n$ would be $10.000\ldots0\mathrm{E}n$ but we drop the last zero and write this as $1.000\ldots0\mathrm{E}n+1$. This means that the 2^{23} patterns in the mantissa represent numbers that lie between 2^n and 2^{n+1}, evenly spaced. The next 2^{23} patterns similarly represent numbers evenly spaced between 2^{n+1} and 2^{n+2}, and so on. The key observation is that the step from 2^{n+1} to 2^{n+2} is *twice* as large as that from 2^n to 2^{n+1}. And from 2^{n+2} to 2^{n+3} it is again twice what it was before.

At the other end, if we look at the largest floating point number just below $1.000\ldots0\mathrm{E}n$ we get $1.111\ldots1\mathrm{E}n-1$ and the numbers with the exponent $n-1$ are squeezed into *half* the space, i.e., they lie between 2^{n-1} and 2^n.

To summarise, as we go towards zero, the floating point numbers are closer and closer together. As we go towards infinity, they are further and further apart. Let's try to draw a picture:

30

In Java the datatype `float` corresponds to 32 bit floating point numbers as described above, the datatype `double` corresponds to 64 bit numbers. For the latter the mantissa has 52 bits (where again the leading one is kept implicit) and the exponent occupies 11 bits. In graphics co-processors one also finds half-precision floating point numbers which require only 16 bits, but Java does not support these.

Summary. This concludes our discussion of the basic number types in mathematics and computer science. We summarise the operations and laws in the following table:

	$+$	$\times$	$-$	$(\ )^{-1}$	add. canc.	mult. canc.
$\mathbb{N}$	✓	✓			✓	✓
$\mathbb{Z}$	✓	✓	✓		✓	✓
$\mathbb{Z}_m$	✓	✓	✓		✓	
$\mathbb{Z}_p$, $\mathbb{Q}$, $\mathbb{R}$	✓	✓	✓	✓	✓	✓

Exercises

1. Which field laws would you expect to hold for floating point numbers and which may fail?
2. Why should we never use the comparison "`x = y`" in a Java program where `x` and `y` are `float` variables?
3. A Java `float` variable can hold (about) 2^{32} different bit patterns. How many of these denote numbers between -1 and 1?

Practical advice

In the exam, I expect you to be able to

- draw practical conclusions from the finite precision afforded by floating point numbers.

I will also expect you to know

- the notation $\mathbb{R}$ for the reals.

I will *not* ask you about the details of the IEEE-754 standard.

More information

You may ask, can't we do better? Indeed, every real number can be thought of as an infinite *stream* of digits which contains precisely one decimal/binary point and we can write computer programs that read and produce such streams, digit by digit. Such programs can be said to implement **exact real arithmetic**, but it is not at all obvious how one might implement efficiently even elementary functions such as multiplication. In fact, exact real arithmetic is still an active area of research. If you are interested, have a look at `https://www.cs.bham.ac.uk/~mhe/papers/index.html` or `https://github.com/norbert-mueller/iRRAM`

4 The real numbers

Answers to exercises

1. Addition and multiplication are still commutative. 0 is neutral for addition and 1 is neutral for multiplication. $-a$ is still the negative to a. Everything else usually fails.
2. Because of round-off errors, you can never be quite sure what exact value you get in a floating point computation.
3. About half of them, 2^{31}.

5 Sets

5.1 The intuitive idea of a set

A **set** is a collection of things. The "things" can be concrete (such as "people") or abstract (such as "colours"). No reason for collecting the things into a set needs to be given but often there is a shared attribute. Examples:

- the set of undergraduate students at the University of Birmingham in the academic year 2021/22 (there are about 25,000);
- the seasons (there are four);
- the set of numbers that can be stored exactly as a `float` in Java (there are slightly fewer than 2^{32} many).

The "things" in the set are called the **elements** or the **members**. The following are important aspects of the idea of a set:

1. The elements of a set are identifiable and can be distinguished from each other.

   (So the "waves on the Atlantic" do not form a set.)

2. There is a clear criterion that defines when a "thing" is a member of the set and when it is not.

   (So we cannot form the set of "tall people in Britain.")

3. A member of a set is counted only once.

   (So the "set of staff members who serve on Staff-Student Committee or Teaching Committee" contains Mark Lee only once, although he satisfies both criteria.)

4. As far as the set is concerned, its elements are not ordered in any way.

   (So although it may appear natural to list the elements of the set of "books by Suzanne Collins" in the order

   - The Hunger Games
   - Catching Fire
   - Mockingjay

   we are not required to do so; listing them in any other order would define the same set.)

5. The set is defined by which members it has and nothing else. If two collections have the same elements, then they form the same set, independently of how the collections were defined.

   (So the "set of triangles which have three equal sides" is the same as the "set of triangles which have three equal angles.")

5.2 Notation

To indicate that x is an element of the set A, one writes

$$x \in A$$

In mathematics the symbol "$\in$" is used for this purpose and this purpose alone (it originates from the Greek letter *epsilon*, written "ε"). Principle 5 from the previous item can now be written in the form

31

Sets with few elements can often be written out explicitly, for example, the set of seasons:

$$\{\text{Spring}, \text{Summer}, \text{Autumn}, \text{Winter}\}$$

Infinite sets can be indicated by suggesting a formation principle:

$$\{1, 4, 9, 16, 25, \ldots\}$$

Sets in Computer Science. Computer scientists use the language of sets for formal and precise descriptions, just like mathematicians, scientists, and engineers. They even developed it into a full-blown *specification language*, called Z.

Set theory has also influenced the design of systems and programming languages. A particularly striking example are "relational databases," suggested in 1970 by the British computer scientist Edgar F. "Ted" Codd. The principles of set theory, listed in Item 5.1, all have immediate relevance here, in particular 1, 3, and 4:

- 1: this is related to the issue of "keys" in a database table;
- 3: storing information about the same object more than once is wasteful and in many cases causes real errors;
- 4: giving the database management system the freedom to arrange the entries of a table in any order allows many optimizations in storage, processing, and retrieval.

Sets are also related to the programming language idea of a "type." Indeed, finite sets can be defined directly in C as "enumerated types," for example

```
enum Seasons {Spring, Summer, Autumn, Winter};
```

Java has supported enumerated types since version 1.5; the syntax is the same as in C. Underneath, however, `enum`'s are actually special `class` definitions and the Java compiler automatically creates several methods for it.

You should also know that Java supports the collection type `Set` (as an `interface`) and various concrete implementations, for example, the very useful `HashSet`.

5.3 Sets of numbers

Let us consider the various systems of numbers which we studied in the first two weeks. The **set of natural numbers**, as you may remember, is denoted by $\mathbb{N}$. You can imagine the elements of $\mathbb{N}$ to be given as strings of the form $s(s(\ldots s(0)\ldots))$, or as numbers in some place-value system such as the decimal one, or you can think of them as *abstract* concepts. In any case, for mathematicians there is only one set $\mathbb{N}$.

To describe the set of **integers**, we add to $\mathbb{N}$ the set of negative whole numbers. For this we use the operator $\cup$ which joins two sets into one. It is pronounced "**union**". Using it, we can write the set of integers as

32

where the elements of the new component can be thought of as written out in the usual decimal system with a minus sign in front.

Having both natural numbers and integers, we can define the set of **fractions** as **pairs** of an integer (the numerator) and a non-zero natural number (the denominator). In general, if A and B are sets (possibly the same one) we write $A \times B$ for the set whose elements are pairs (x,y) where x is an element of A and y is an element of B. If $A = B$ then we may abbreviate $A \times A$ to A^2. We call $A \times B$ the **product** of A and B. So the set of fractions could be written as the product $\mathbb{Z} \times P$ where P is the set of strictly positive natural numbers.

How do we define P? For this we use the **set difference** operator $\setminus$. In general, $A \setminus B$ is the set of all elements of A which do **not** belong to B. So:

33

All this is quite straightforward but how would we construct $\mathbb{Q}$, the set of rational numbers? Remember from Chapter 3 that the rationals are to be viewed as fractions *with a different equality*. At the moment our set theory does not provide a facility for imposing an alternate equality. We will deal with this systematically in Chapter 7. As a stop gap, we could restrict the set of fractions to those where numerator and denominator do not have a common factor greater than 1. In general, we say that A is a **subset** of a set B if all elements of A are also elements of B. In symbols, this is written as $A \subseteq B$. So our definition of $\mathbb{Q}$ is as a subset of the set of fractions. We'll have a lot more to say about subsets in the next chapter.

A similar issue arises with the finite rings $\mathbb{Z}_m$. On the one hand, each of these can be directly realised as the finite set $\{0,1,\ldots,m-1\}$, i.e., the possible remainders of dividing a natural number by m; on the other hand, it is much better to view $\mathbb{Z}_m$ as the set of integers but with the alternate equality "congruent modulo m".

The bit patterns that can be stored in a 32 bit register can be seen as 32-tuples, i.e., elements of the set B^{32} where $B = \{0,1\}$. This is a simple generalisation of the idea of a product. More interesting is the data type `String`. For this we introduce a new set formation operator: If A is a set then A^* is the set whose elements are finite sequences of elements of A. In computer science, the set A is often finite and called the "alphabet"; the elements of A^* are then the "words over A". The shortest element of A^* is the "empty word," or the "empty sequence," often denoted by ε (the real Greek letter epsilon). The alphabet for Java's strings is called **Unicode**; it has $2^{16} = 65,536$ characters.

In order to describe $\mathbb{R}$ as a set we also allow the construction of *infinite streams*. More precisely, if A is a set then A^ω is the set of infinite sequences of elements of A. With this device we can define a candidate for the set $\mathbb{R}$ of **real numbers**. Every element of $\mathbb{R}$ consists of an integer part and an infinite sequence of digits (the fractional part), so

34: The reals — not quite right

where $D = \{0,1,\ldots,9\}$. However, in doing so we are ignoring the fact that a decimal fraction that ends in an infinite sequence of nines is considered equal to one which ends in an infinite sequence of ones. For example, $0.999\ldots = 1.000\ldots$ We can remedy this by excluding from $\mathbb{Z} \times D^\omega$ those pairs where the second component ends in an infinite sequence of nines. However, as with the rational numbers and $\mathbb{Z}_m$, this does not reflect our day-to-day working with reals and we would much rather be able to postulate the equality between these special elements.

5.4 Counting the elements of a set

We all know how to do this, of course: If A is the set of seasons from Section 5.1 then we can say that it has four elements. This is written in set theory as

$$|A| = 4 \quad \text{or} \quad \text{card}(A) = 4$$

where the function "card" is short for **cardinality**. We could also call it the "size" of a set.

Given two sets A and B, we don't need to count the elements in order to check that they have the same number of elements; instead we can line up the element of A and B in such a way that each element of A is paired with exactly one element of B (and vice versa):

35

which would persuade us that $\text{card}(\text{Seasons}) = \text{card}(\text{Compass})$, even if we had never heard of the number 4.

5.5 Cardinality of infinite sets

At first sight, the sets $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$ all have infinitely many elements and we could say that their cardinality, therefore, is "infinity." It was Georg Cantor's great discovery that in fact not all infinite sets have the same cardinality. The argument is easy and does not use much from the theory of sets.

The first infinite set that comes to mind is that of the natural numbers, $\mathbb{N}$. Let us call a set **countable**, or more precisely, **countably infinite**, if it has the same cardinality as $\mathbb{N}$. There are more countable sets than you might think. We go through a sequence of examples.

- $A = \mathbb{N} \cup \{-1\}$ is countable:

36

$\mathbb{N}$

- $\mathbb{Z}$ is countable

37

$$\mathbb{N} = \{ \quad 0, \qquad 1, \qquad 2, \qquad 3, \qquad 4, \qquad \ldots \}$$

- $\mathbb{N}^2$ is countable

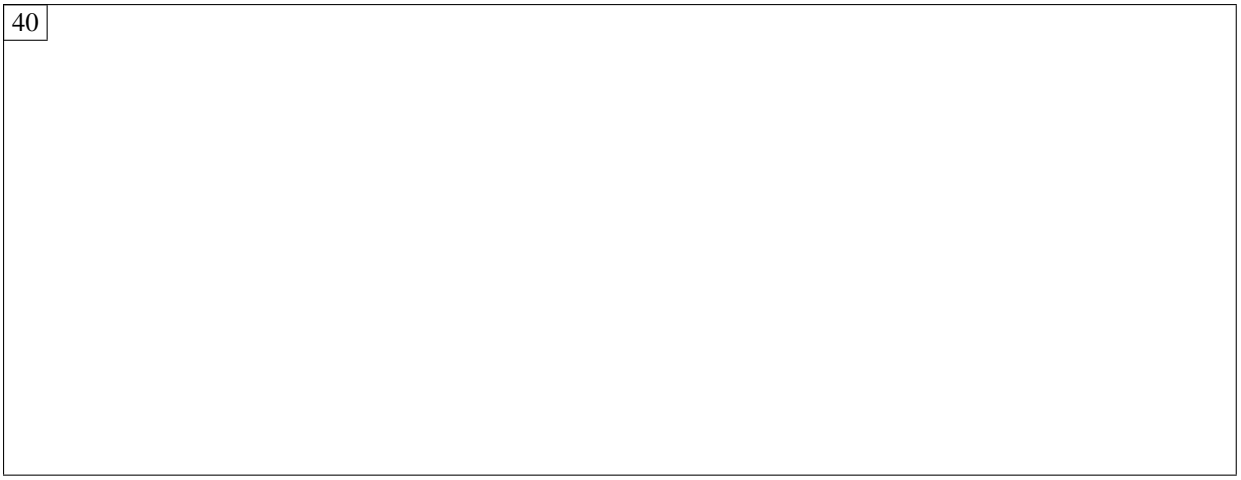
38

$$\mathbb{N} = \quad \{ \quad 0, \qquad 1, \qquad 2, \qquad 3, \qquad 4, \qquad 5, \qquad \ldots \}$$

This enumeration trick is more clearly illustrated with a two-dimensional picture:

39

- If A is a finite set (an "alphabet") then A^* is countable. A good picture to illustrate this is the following (for $A = \{a, b\}$):



which is nothing other than a "breadth-first traversal" of the binary tree of all the words over $\{a,b\}$. You can see that it would work for any alphabet size (as long as the alphabet is finite).

- Java programs: These form a subset of A^* where A is the 65536 letter alphabet of Unicode. So it follows from the last example that:

  Theorem 2 *The set of valid Java programs is countable.*

  This is an important fact. Make sure you remember it!

5.6 Uncountability

We can think of the countable sets as being "infinite but still relatively small." A set that is not "small" in this sense is $\mathbb{R}$, the set of real numbers. The proof is by **contradiction**.[2] *Assume* we could list in the fashion of the previous item all the real numbers, that is, assume we had a listing that looks like the following:

	0	1	2	3	4	5
r_0	3.	1	4	1	5	9…
r_1	2.	7	1	8	2	8…
r_2	6.	0	2	2	1	4…
r_3	22.	7	1	0	9	8…
r_4	9.	8	0	6	6	5…
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Now consider the real number

$$a = a_0 \,.\, a_1\, a_2\, a_3\, \ldots$$

where

$$a_i = \begin{cases} 0 & \text{if } r_i[i] \neq 0 \\ 1 & \text{if } r_i[i] = 0 \end{cases}$$

where $r_i[i]$ means the digit in row i and column i in the assumed listing of all real numbers. For example, the number

$$r_0[0] \,.\, r_1[1]\, r_2[2]\, r_3[3]\, r_4[4]$$

derived from the example listing above starts with

$$3.7206\ldots$$

Consequently, the number a associated with this listing starts with

$$0.0010\ldots$$

As you can see, the number a is different from r_0 because it was constructed so that it differs in its integral part (the part before the decimal point); it differs from r_1 because it was made to differ from it in the first digit after the point; and so on.

[2] The proof idea is now truly famous. It was invented by Cantor in 1891, used in 1901 by Bertrand Russell in his "paradox," by Kurt Gödel in 1931 in his "Incompleteness Theorem," and by Alan Turing in 1937 in his proof of the "Undecidability of the *Entscheidungsproblem*."

In general, a differs from r_i in the i-th digit, so is certainly different from r_i. (Of course, it'll probably differ in many other places as well.)

This means that a is a real number that does not appear in the listing; in other words, our assumption that the listing is complete cannot be true. *Contradiction!* Since our construction of a was completely general and could be carried out for any listing of real numbers, we have to conclude that any such listing *by necessity* will miss out some real numbers. We therefore say:

Theorem 3 *The set of real numbers is* ***uncountable****.*

What does this mean for Computer Science? Since there are (only) countably many Java programs but uncountably many real numbers, there must exist real numbers that cannot be computed by a program. We can say:

Theorem 4 *There are* ***non-computable*** *real numbers.*

Comparing uncountable sets. The examples above illustrate how we can establish that two countably infinite sets have the same cardinality. To show that two *uncountable* sets have the same size requires a bit more ingenuity but the principle is the same: We establish a one-to-one relationship between the elements. For example, we can show that the elements of a circle (minus one point) can be lined up against the elements of the real line. The proof is geometric:

41

We see that every line through the "north pole" N links a point P on the circle and a point P' on the real line. This works for all points on the circle except the north pole itself.

Exercises

1. Let A and B be two countable sets. Argue that $A \cup B$ is also countable.
2. Find an arithmetic or a geometric argument that shows that the following two subsets of $\mathbb{R}$ have the same cardinality: $A = \{x \in \mathbb{R} \mid -1 < x < 1\}$ and $B = \{y \in \mathbb{R} \mid -2 < y < 2\}$.

Practical advice

In the exam I expect you to

- know the set-theoretic symbols $\in$, $\subseteq$, $\cup$, and $\setminus$ (more are to come);
- know the traditional symbol for sets of numbers $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, and $\mathbb{R}$;
- know the symbol for the set of strings A^* over an alphabet A, and likewise A^ω for the set of infinite streams;
- know what "countable" means for a set;
- be able to argue that $\mathbb{Z}$, $\mathbb{N}^2$, and A^* are countable;
- to be able to recall that $\mathbb{R}$ is uncountable, and give some hints as to the proof of this fact;
- demonstrate an understanding of Theorem 4 and its consequences.

5 Sets

31

$$A = B \quad \text{if and only if} \quad \forall x.\, x \in A \iff x \in B$$

32

$$\mathbb{Z} = \mathbb{N} \cup \{-1, -2, -3, \ldots\}$$

33

$$P = \mathbb{N} \setminus \{0\}$$

34: The reals — not quite right

$$\mathbb{R} = \mathbb{Z} \times D^{\omega}$$

35

$$\begin{array}{rcccccc} \text{Seasons} & = & \{\text{Spring}, & \text{Summer}, & \text{Autumn}, & \text{Winter}\} \\ & & | & | & | & | \\ \text{Compass} & = & \{\text{East}, & \text{South}, & \text{West}, & \text{North}\} \end{array}$$

36

$$\begin{array}{rcccccccc} \mathbb{N} & = & \{ & 0, & 1, & 2, & 3, & 4, & \ldots\} \\ & & & | & | & | & | & | & \\ A & = & \{ & -1, & 0, & 1, & 2, & 3, & \ldots\} \end{array}$$

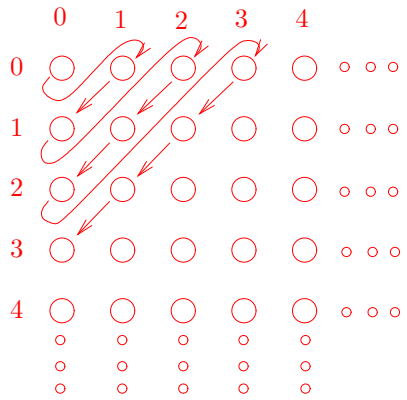
37

$$\begin{array}{rcccccccc} \mathbb{N} & = & \{ & 0, & 1, & 2, & 3, & 4, & \ldots\} \\ & & & | & | & | & | & | & \\ \mathbb{Z} & = & \{ & 0, & 1, & -1, & 2, & -2, & \ldots\} \end{array}$$

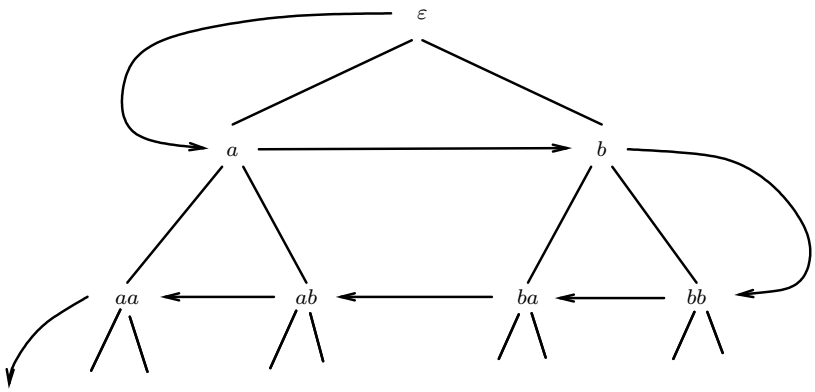
38

$$\begin{array}{rcccccccc} \mathbb{N} & = & \{ & 0, & 1, & 2, & 3, & 4, & 5, \quad \ldots\} \\ & & & | & | & | & | & | & | \\ \mathbb{N}^2 & = & \{ & (0,0), & (1,0), & (0,1), & (2,0), & (1,1), & (0,2), \quad \ldots\} \end{array}$$

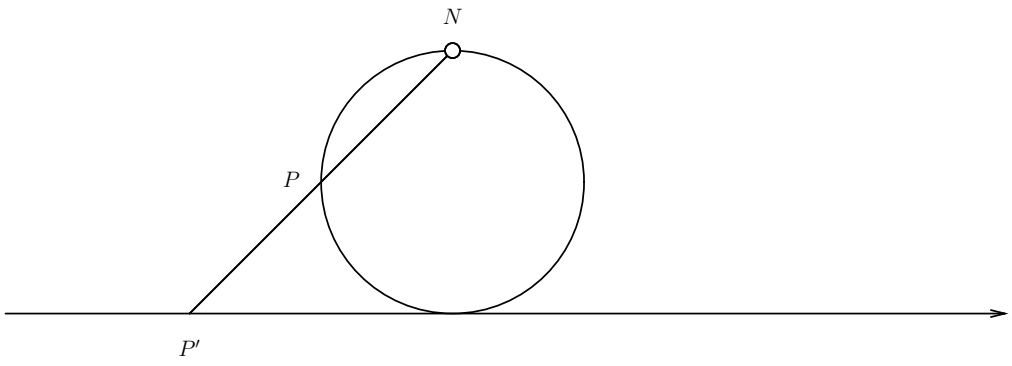
39



40



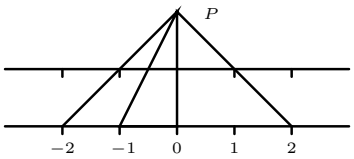
41



Answers to exercises

1. We imagine that we have listings $A = \{a_0, a_1, \ldots\}$ and $B = \{b_0, b_1, \ldots\}$ that establish the countability of A and B. We can then list the elements of $A \cup B$ by interleaving the two series: $a_0, b_0, a_1, b_1, \ldots$, omitting any duplicates.

2. The arithmetic argument is that we can pair each $x \in A$ with $y = 2x \in B$. The geometric argument is the same, really:



Every line from P connects exactly one point of A with one point of B.

6 Power sets and Boolean algebra

6.1 Subsets

If the subset we have in mind does not have many members then we can just list them. For example, here is a subset of $\mathbb{N}$, the set of all known "Fermat prime numbers":

$$\{3,5,17,257,65537\} \subseteq \mathbb{N}$$

Likewise, if we wish to exclude a few members from a set, then we can use the set difference operator introduced in the last chapter. As an example, the expression

$$\mathbb{N} \setminus \{0,1,2,3\}$$

denotes the set of all those natural numbers n for which $2^n < n!$ (as you proved in Exercise 1.1).

We also have two extreme examples of subsets, the **empty set**, written as

$$\{\} \quad \text{or} \quad \emptyset$$

and the whole set viewed as a subset of itself, i.e., for every set A we have

$$A \subseteq A\ .$$

For more sophisticated subsets we employ *logical conditions* to select those members we are interested in. The notation for this is

$$B = \{x \in A \ \mid \ x \text{ satisfies condition } c\}$$

Read this as "B is the set of all elements of A for which the condition c is true." Some examples:

42

When defining a subset in this fashion, it is important that in the first part we state which set we select elements from; this allows us to use negative conditions as well, for example:

$$\{x \in \mathbb{N} \ \mid \ x \text{ is not a perfect square}\} = \{2,3,5,6,7,8,10,\ldots\}$$

In the previous chapter I mentioned that relational databases can be viewed as an implementation of the idea of a set. This is quite apparent in the query language SQL, where a standard query takes the form

```
SELECT name
FROM   staff
WHERE  title = "Dr"
```

and you see the similarity with subset definitions.

Comparing subsets. With the help of the "element of" relation $\in$, we can state the condition for $B \subseteq A$ precisely:

43

Comparing this with the formula that defines the equality of sets (Box 31), we see that we have the following "theorem":

44: Equality of sets

$$A = B \text{ if and only if } A \subseteq B \text{ and } B \subseteq A$$

This is extremely handy in proving that two set expressions are equal. Remember it!

6.2 Operations on subsets

Suppose B, B_1 and B_2 are subsets of a common set A. We have the following operations:

Union. By the "union of B_1 and B_2" we mean taking the elements of B_1 and B_2 together (but not repeating those elements that appear in both). As a formula:

$$B_1 \cup B_2 = \{x \in A \mid x \in B_1 \text{ or } x \in B_2\}$$

As a **Venn diagram**:

45

Intersection. By the "intersection of B_1 and B_2" we mean the set of elements that are common to B_1 and B_2. As a formula:

$$B_1 \cap B_2 = \{x \in A \mid x \in B_1 \text{ and } x \in B_2\}$$

As a Venn diagram:

46

Complement. By the "complement of B" we mean those elements of A which do *not* belong to B. As a formula:

$$\overline{B} = \{x \in A \mid x \notin B\}$$

As a Venn diagram:

47

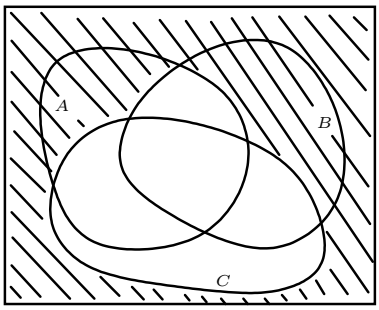
Set difference. This is not really needed because it can be expressed with intersection and complement, but it is a very useful operation:

$$B_1 \setminus B_2 = B_1 \cap \overline{B_2} = \{x \in A \mid x \in B_1 \text{ and } x \notin B_2\}$$

48

As the picture indicates, there is no requirement for B_2 to be a subset of B_1.

Do we have enough operators, i.e., ways of combining subsets? Let's try an example. Find an expression in A, B, and C which describes the shaded region in the following Venn diagram:



49

6.3 Power sets

In set theory it is postulated that the subsets of a set A can again be collected together into a set. The result is called the **power set** of A and written $\mathscr{P}A$. It is a useful exercise to check the conditions for a set given at the beginning of Chapter 5:

1. The elements of a power set are "identifiable" because set theory postulates clearly what it means for two sets to be equal or not equal (the formula in Box 31), and this turns the subsets of A into "identifiable things."
2. The criterion for a set B to be a member of $\mathscr{P}A$ is that B is a subset of A. This in turn has a precise definition: every element of B must also be an element of A.

It is also quite important that we have said that the elements of a set count only once, because typically there are many different ways to specify a subset. These different specifications of a subset are not important from the point of view of the power set, only the elements that each subset contains. Let's see an example:

50

$\mathscr{P}\{a,b,c\} =$

Since we always have

$$\emptyset \subseteq A$$

we get that the empty set is an element of any power set (even the power set of the empty set).

For any element x of A we can form the **singleton set** $\{x\}$, which in turn is clearly a subset of A. Hence the power set of A contains all singleton sets and from this consideration it seems that a power set $\mathscr{P}A$ has many more elements than A itself. If the number of elements in A is small, then we can test this belief directly:

51

Georg Cantor showed that the conjecture is true not only for finite sets but also for infinite ones. Let's formulate this precisely:

Theorem 5 *For any set A, the cardinality of the power set $\mathscr{P}A$ is strictly larger than the cardinality of A.*

52

In the last chapter we showed that $\mathbb{R}$ has greater cardinality than $\mathbb{N}$. From Cantor's theorem, we see that this is not the end of the story; in fact, the sequence

$$\mathbb{N} \quad \mathscr{P}\mathbb{N} \quad \mathscr{P}\mathscr{P}\mathbb{N} \quad \mathscr{P}\mathscr{P}\mathscr{P}\mathbb{N} \quad \mathscr{P}\mathscr{P}\mathscr{P}\mathscr{P}\mathbb{N}\ldots$$

leads to sets of greater and greater cardinality. Mind boggling!

6.4 Propositional logic and Boolean algebras

The power set as an algebra. Above we introduced the three operations $\cup$, $\cap$, and $\overline{\ }$ on subsets of a set X. What laws do these obey? Here is a selection:

53

$A \cup \emptyset$	$=$	A		$A \cap X$	$=$	A	(neutral elements)
$A \cup X$	$=$	X		$A \cap \emptyset$	$=$	$\emptyset$	(annihilators)
$A \cup B$	$=$	$B \cup A$		$A \cap B$	$=$	$B \cap A$	(commutativity)
$(A \cup B) \cup C$	$=$	$A \cup (B \cup C)$		$(A \cap B) \cap C$	$=$	$A \cap (B \cap C)$	(associativity)
$A \cup A$	$=$	A		$A \cap A$	$=$	A	(idempotence)
$(A \cup B) \cap B$	$=$	B		$(A \cap B) \cup B$	$=$	B	(absorption)
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$							(distributivity of $\cap$ over $\cup$)
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$							(distributivity of $\cup$ over $\cap$)
$A \cup \overline{A}$	$=$	X		$A \cap \overline{A}$	$=$	$\emptyset$	(complements)
$\overline{A \cup B}$	$=$	$\overline{A} \cap \overline{B}$		$\overline{A \cap B}$	$=$	$\overline{A} \cup \overline{B}$	(de Morgan laws)
$\overline{\overline{A}} = A$							(double complement)

Too many to remember! but each of them quite easy to prove. Let's do this for the de Morgan law $\overline{A \cup B} = \overline{A} \cap \overline{B}$. We use the method given in Box 44:

54

The subset operations introduced in this chapter are closely connected to logic. If we have subsets B_1 and B_2 defined by selection, as in

$$B_1 = \{x \in A \mid x \text{ satisfies condition } c_1\} \quad \text{and} \quad B_2 = \{x \in A \mid x \text{ satisfies condition } c_2\}$$

Then $B_1 \cup B_2$ is defined by

$$B_1 \cup B_2 = \{x \in A \mid x \text{ satisfies condition } c_1 \textbf{ or } c_2\}$$

Likewise, $B_1 \cap B_2$ is defined by

$$B_1 \cap B_2 = \{x \in A \mid x \text{ satisfies condition } c_1 \textbf{ and } c_2\}$$

Complement, finally, is given by negation:

$$\overline{B_1} = \{x \in A \mid x \text{ does } \textbf{not} \text{ satisfy condition } c_1\}$$

The logical operators are written with symbols very similar to those for the corresponding subset operations: We write $\vee$ for "or", $\wedge$ for "and", and $\neg$ for "not". The laws listed in Box 53 are all valid for the logical connectives as well, if we also remember that the role of the empty set $\emptyset$ is filled by the logical constant `false`, while the ambient set denoted X in Box 53 corresponds to `true`. Let's write them out (again):

55: Laws of Boolean algebras

$A \vee \mathsf{false}$	$=$	A	$A \wedge \mathsf{true}$	$=$	A	(neutral elements)
$A \vee \mathsf{true}$	$=$	true	$A \wedge \mathsf{false}$	$=$	false	(annihilators)
$A \vee B$	$=$	$B \vee A$	$A \wedge B$	$=$	$B \wedge A$	(commutativity)
$(A \vee B) \vee C$	$=$	$A \vee (B \vee C)$	$(A \wedge B) \wedge C$	$=$	$A \wedge (B \wedge C)$	(associativity)
$A \vee A$	$=$	A	$A \wedge A$	$=$	A	(idempotence)
$(A \vee B) \wedge B$	$=$	B	$(A \wedge B) \vee B$	$=$	B	(absorption)
		$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$				(distributivity of $\wedge$ over $\vee$)
		$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$				(distributivity of $\vee$ over $\wedge$)
$A \vee \neg A$	$=$	true	$A \wedge \neg A$	$=$	false	(complements)
$\neg(A \vee B)$	$=$	$\neg A \wedge \neg B$	$\neg(A \wedge B)$	$=$	$\neg A \vee \neg B$	(de Morgan laws)
		$\neg\neg A = A$				(double negation)

Boolean algebras. Generally, whenever we have a set with operations $\vee$, $\wedge$, and $\neg$, and constants false and true, satisfying the laws of Box 55, we speak of a **Boolean algebra**. Thus, every power set can be viewed as a Boolean algebra.

The simplest Boolean algebra consists of nothing else but the two constants false and true. We call it the Boolean algebra of **truth values**. It happens that here the Boolean algebra laws tell us exactly how the operations work. As an example:

56

If we do this for all possible combinations we get the well-known **truth tables**:

57

Boolean circuits. Let us finish this chapter with an application to hardware design. The two electrical states "low" and "high" can be associated with the logical constants false and true, but we write them as "0" and "1". The trick now is to replace the various *arithmetic* operations derived from the binary representation of numbers with *logical* ones, since the logical operations can quite easily be realised with transistors.

As an example, consider the addition of two binary (single) digits x and y with one carry bit c. We can list all possible inputs and the expected outputs s (for the sum) and c' for the new carry bit:

58

For illustration, we can interpret the eight different inputs as the eight regions of a Venn diagram with three subsets. We shade them according to whether the resulting output is 0 or 1. We get two diagrams, one for the new carry bit c', and one for the sum s:

59

Is it clear that these regions have a logical description? Perhaps the table in Box 58 suggests a canonical solution:

$$\begin{aligned} c' &= (x \wedge y \wedge \neg c) \vee (x \wedge \neg y \wedge c) \vee (\neg x \wedge y \wedge c) \vee (x \wedge y \wedge c) \\ s &= (x \wedge \neg y \wedge \neg c) \vee (\neg x \wedge y \wedge \neg c) \vee (\neg x \wedge \neg y \wedge c) \vee (x \wedge y \wedge c) \end{aligned}$$

This is an example of a **disjunctive normal form** of a Boolean expression, since it is a "disjunction of conjunctions of literals", where "literal" means either a variable or a negated variable. It makes it clear that any Boolean function can be realised with our three connectives. For the hardware designer the story doesn't end here, of course, they will now try to find the *simplest* such expression so as to save on the number of transistors required and to minimise the power consumption of the final chip. We don't have time to go into this but the laws of Boolean algebras are clearly a key ingredient of this process.

Exercises

1. Let A be the set $\{x \in \mathbb{N} \mid x < 10\}$ and consider the following subsets of A:

$$\begin{aligned} B &= \{0,2,4,6,8\} \\ C &= \{0,1,2,3,4\} \\ D &= \{0,1,4,9\} \end{aligned}$$

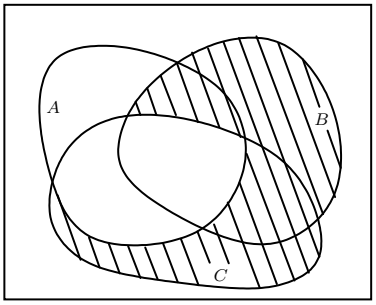
   Write out each of the following sets:

   (i) $B \cup C$ (ii) $B \cap C$ (iii) $\overline{B} \cap D$ (iv) $B \cap \overline{D}$ (v) $B \setminus D$ (vi) $D \setminus B$

2. Let $A = \{x \in \mathbb{N} \mid \exists n \in \mathbb{N}.\ x = 3n\}$ and $B = \{x \in \mathbb{N} \mid \exists n \in \mathbb{N}.\ x = n^3\}$. Which of the following statements are true, which are false? Give (short!) justifications for your answers.

   (i) $9 \in A$ (ii) $9 \in B$ (iii) $A \subseteq B$ (iv) $B \subseteq A$ (v) $A \cap B = \emptyset$

3. Give a Boolean algebra expression for the shaded subset in the Venn diagram below. (In addition to the Boolean algebra operations, you are allowed to use set difference.)



Practical advice

In the exam I expect you to be able to

- read subset definitions written in logic;
- write subset definitions in logic;

- read and evaluate nested expressions that use $\cup$, $\cap$, $\overline{\ }$, and $\setminus$;
- draw Venn diagrams of such expressions;
- use the laws of Boolean algebra to simplify such expressions;
- prove two set expressions to be equal by using either the method of Box 31 or that of Box 44.

6 Power sets and Boolean algebra

42

$$\{x \in \mathbb{N} \mid x \text{ is a prime number}\} = \{2,3,5,7,11,\ldots\}$$

$$\{x \in \mathbb{Z} \mid x \text{ is strictly greater than } -5\} = \{-4,-3,-2,-1,0,1,2,3,\ldots\}$$

$$\{\tfrac{p}{q} \in \mathbb{Q} \mid q \text{ is a power of } 2\}$$

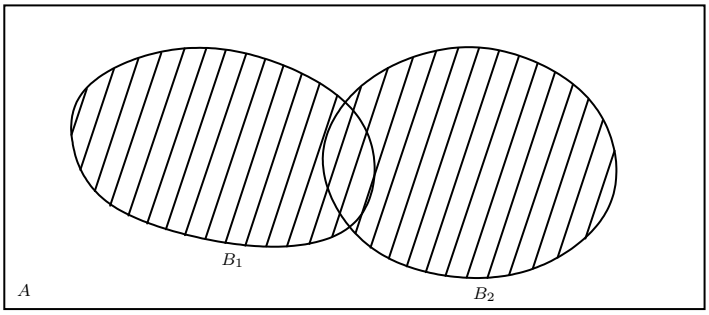
43

$$B \subseteq A \quad \text{if and only if} \quad \forall x.\, x \in B \implies x \in A$$

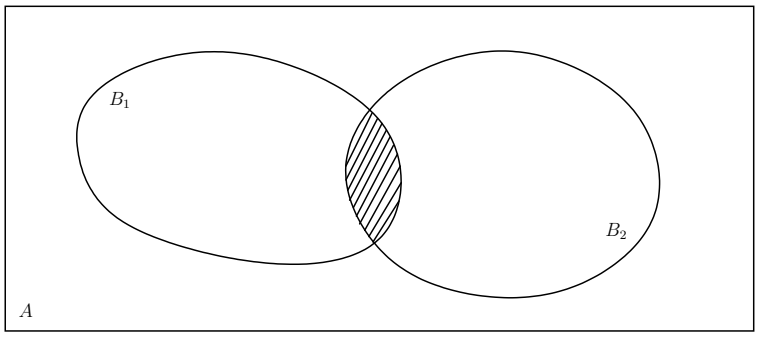
44: Equality of sets

$$A = B \text{ if and only if } A \subseteq B \text{ and } B \subseteq A$$

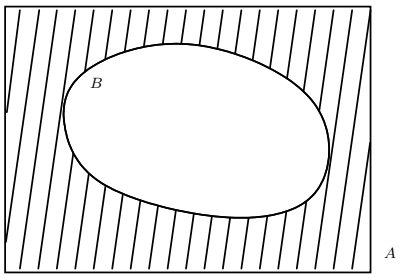
45



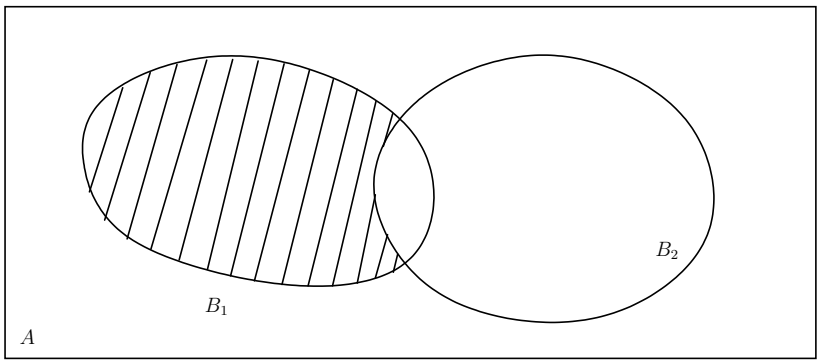
46



47



48



49

Answer. The shaded region corresponds to $\overline{C} \setminus (A \cap B)$ but there are many other expressions.

50

$\mathscr{P}\{a,b,c\} = \{\{\},\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\}$

51

$\operatorname{card}(A) = 0$	1	2	3	4	$\cdots$	n	$\cdots$
$\operatorname{card}(\mathscr{P}A) = 1$	2	4	8	16	$\cdots$	2^n	$\cdots$

52

Since $\mathscr{P}A$ contains the singleton set $\{x\}$ for every element x of A, we are certainly entitled to say that $\text{card}(A) \leq \text{card}(\mathscr{P}A)$. To complete the proof it therefore remains to show that A and $\mathscr{P}A$ can't have the same cardinality.

For the sake of a contradiction, assume that A is some set and that the elements of A can be paired with the elements of $\mathscr{P}A$ (in the style of the last chapter). So for every $x \in A$ there is a subset $S(x)$ which it is paired with, and vice versa, for every $B \subseteq A$, there is an element e_B such B is its partner. As this is a one-to-one pairing, we also have $S(e_B) = B$ and $e_{S(x)} = x$.

For some $x \in A$ it might happen that $x \in S(x)$ and for others it might not be so. It makes sense, therefore, to define the subset

$$C \stackrel{\text{def}}{=\!=} \{x \in A \mid x \notin S(x)\} \subseteq A\,.$$

Let e_C be the element that's partnered with C. We ask, is it the case that $e_C \in C$? Either the answer is yes or it is no.

If the answer is yes, i.e., $e_c \in C$, then by the definition of C it must be the case that $e_C \notin S(e_C) = C$, which is a contradiction.

If the answer is no, i.e., $e_C \notin C$, then by the definition of C, the element e_C belongs to the subset C, which is again a contradiction.

We conclude that a one-to-one pairing between A and $\mathscr{P}A$ can not exist.

53

$A \cup \emptyset$	$=$	A		$A \cap X$	$=$	A	(neutral elements)
$A \cup X$	$=$	X		$A \cap \emptyset$	$=$	$\emptyset$	(annihilators)
$A \cup B$	$=$	$B \cup A$		$A \cap B$	$=$	$B \cap A$	(commutativity)
$(A \cup B) \cup C$	$=$	$A \cup (B \cup C)$		$(A \cap B) \cap C$	$=$	$A \cap (B \cap C)$	(associativity)
$A \cup A$	$=$	A		$A \cap A$	$=$	A	(idempotence)
$(A \cup B) \cap B$	$=$	B		$(A \cap B) \cup B$	$=$	B	(absorption)
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$							(distributivity of $\cap$ over $\cup$)
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$							(distributivity of $\cup$ over $\cap$)
$A \cup \overline{A}$	$=$	X		$A \cap \overline{A}$	$=$	$\emptyset$	(complements)
$\overline{A \cup B}$	$=$	$\overline{A} \cap \overline{B}$		$\overline{A \cap B}$	$=$	$\overline{A} \cup \overline{B}$	(de Morgan laws)
$\overline{\overline{A}} = A$							(double complement)

54

$$\begin{aligned} x \in \overline{A \cup B} &\implies x \notin A \cup B \\ &\implies x \notin A \text{ and } x \notin B \\ &\implies x \in \overline{A} \text{ and } x \in \overline{B} \\ &\implies x \in \overline{A} \cap \overline{B} \end{aligned}$$

This shows $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$. For the other inclusion, we can read the transformations above from bottom to top, and the proof is complete.

55: Laws of Boolean algebras

$A \vee \mathsf{false}$	$=$	A		$A \wedge \mathsf{true}$	$=$	A	(neutral elements)
$A \vee \mathsf{true}$	$=$	true		$A \wedge \mathsf{false}$	$=$	false	(annihilators)
$A \vee B$	$=$	$B \vee A$		$A \wedge B$	$=$	$B \wedge A$	(commutativity)
$(A \vee B) \vee C$	$=$	$A \vee (B \vee C)$		$(A \wedge B) \wedge C$	$=$	$A \wedge (B \wedge C)$	(associativity)
$A \vee A$	$=$	A		$A \wedge A$	$=$	A	(idempotence)
$(A \vee B) \wedge B$	$=$	B		$(A \wedge B) \vee B$	$=$	B	(absorption)
		$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$					(distributivity of $\wedge$ over $\vee$)
		$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$					(distributivity of $\vee$ over $\wedge$)
$A \vee \neg A$	$=$	true		$A \wedge \neg A$	$=$	false	(complements)
$\neg(A \vee B)$	$=$	$\neg A \wedge \neg B$		$\neg(A \wedge B)$	$=$	$\neg A \vee \neg B$	(de Morgan laws)
		$\neg\neg A = A$					(double negation)

56

$$\begin{aligned} \mathsf{false} \vee \mathsf{true} &= \mathsf{true} && \text{annihilation} \\ \neg\mathsf{false} &= \neg\mathsf{false} \vee \mathsf{false} && \text{neutral element} \\ &= \mathsf{true} && \text{complements} \end{aligned}$$

57

$\vee$	false	true
false	false	true
true	true	true

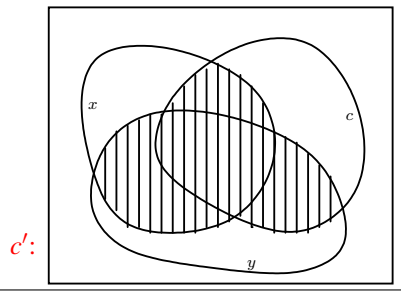
$\wedge$	false	true
false	false	false
true	false	true

$\neg$	
false	true
true	false

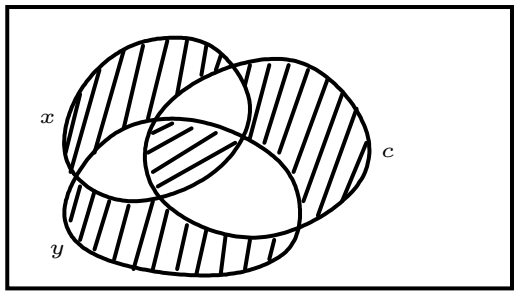
58

x	y	c	c'	s
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

59



c':



s:

Answers to exercises

1. (i) $B \cup C = \{0,1,2,3,4,6,8\}$ (ii) $B \cap C = \{0,2,4\}$ (iii) $\overline{B} \cap D = \{1,9\}$
   (iv) $B \cap \overline{D} = \{2,6,8\}$ (v) $B \setminus D = \{2,6,8\}$ (vi) $D \setminus B = \{1,9\}$

2. (i) Yes, because for $n = 3$ we have $9 = 3 \times 3$.
   (ii) No, because 9 is not a third power of any natural number.
   (iii) No, because $9 \in A$ but $9 \notin B$.
   (iv) No, because $8 = 2^3 \in B$ but $8 \notin A$.
   (v) No, because $27 = 3 \times 9 = 3^3 \in A \cap B$.

3. One solution is $(C \setminus A) \cup (B \setminus C)$. Another is $((C \cup B) \setminus A) \cup (A \cap B)$.

7 Relations

7.1 The general idea of a relation

This is a very general concept with many different interpretations. Let's look at the definition:

Relation

If $A_1, A_2, \ldots, A_n$ are sets and R is a subset of $A_1 \times A_2 \times \ldots \times A_n$, then R is called a (n**-ary**) **relation**.

Remember that the elements of $A_1 \times A_2 \times \ldots \times A_n$ are n-tuples $(x_1, x_2, \ldots, x_n)$. Also, note that *any subset* is allowed; there are no conditions for a subset to be called a relation.

Here is an example of a 3-ary or ternary relation. Let A_1 be the set Σ^* of strings, and $A_2 = A_3 = \mathbb{N}$, the set of natural numbers. We consider the relation `staff` $\subseteq \Sigma^* \times \mathbb{N} \times \mathbb{N}$ of all those triples (s, n, m) where s is the name of a lecturer in the School of Computer Science, n is their phone number, and m is their office number. For example, the triple (`Achim Jung`, 44776, 213) belongs to the relation `staff` but (`Mickey Mouse`, 999, 777) does not. A picture may be helpful as well:

60

I know of two programming paradigms which are founded on the concept of a general relation: databases and logic programming. In the database language SQL one declares the relation with the command:

61

and then triples can be entered into the relation with the command:

62

In Prolog, the relation does not need to be declared. Individual tuples are entered into it by just stating them as "facts":

63

From the *mathematical* point of view there is not much more to say about general relations, though in database theory we are interested in questions such as whether every element of a set is mentioned in a given relation, or whether for every element in one set there is no more than one element in the other set that is in relation to it.

7.2 Binary relations on a set

In the rest of this chapter we look at the special case of **binary relations** on a **single** set, that is, at subsets of $A \times A$. Apart from being mathematically and computationally interesting, such relations can be interpreted as **directed graphs**, where A is the set of vertices and the relation is the set of edges,[3] and for these we can draw pictures:

64

Binary relations also exist "in nature;" we'll take as a running example the set of all people as A, and the relationship "`likes`" as the relation.

Reflexivity. We call a binary relation $R \subseteq A \times A$ **reflexive** if it contains all pairs (x,x) for x an element of A. As a formula:

Reflexive binary relation.

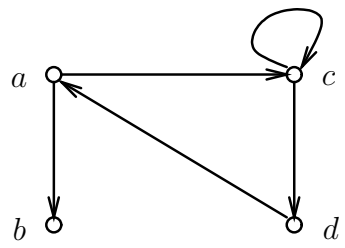
$$R \subseteq A \times A \text{ is reflexive} \iff^{\text{def}} \forall x \in A.\,(x,x) \in R$$

If we are given a relation R that is not reflexive, then we can make it reflexive by adding the missing edges:

$$\text{refl-closure}(R) \stackrel{\text{def}}{=} R \cup \{(x,y) \in A \times A \mid x = y\}$$

The new relation refl-closure(R) is called the **reflexive closure** of R. It is the smallest relation that contains R and is reflexive. Note the simplicity of the formula; we don't have to worry whether an edge is added unnecessarily because sets don't carry duplicates in any case. Let's try the reflexive closure on our example relation from above:

65



The opposite concept to reflexivity is that of an **irreflexive relation**, which is one that does not contain any edges leading from a vertex back to itself. The real-world relationship `likes` is neither reflexive (since there are people who hate themselves) nor irreflexive (as there are people who like themselves very much).

Symmetry. A binary relation $R \subseteq A \times A$ is called **symmetric** if with every pair $(x,y) \in R$ we also have $(y,x) \in R$. As a formula:

Symmetric binary relation.

$$R \subseteq A \times A \text{ is symmetric} \iff^{\text{def}} \forall (x,y) \in A \times A.\,(x,y) \in R \implies (y,x) \in R$$

It is easy to make a relation symmetric; just add the missing links. We develop a bit of notation to write this concisely within the language of sets.

[3] In this interpretation one usually uses the letter V for the set of vertices, and $E \subseteq V \times V$ for the set of edges.

Inverse relation.

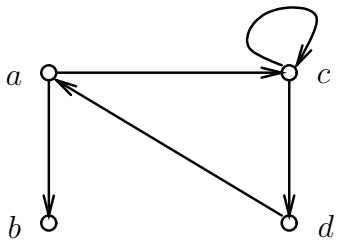
$$R^{-1} \stackrel{\text{def}}{=\!=} \{(x,y) \in A \times A \mid (y,x) \in R\}$$

To make the relation symmetric we add the inverse:

$$\text{symm-closure}(R) \stackrel{\text{def}}{=\!=} R \cup R^{-1}$$

The result is called the **symmetric closure** of R; it is the smallest relation that contains R and is symmetric. Let's apply this to our example relation:

66



As you can see, the drawing of a symmetric graph always has links in both directions between vertices, or none at all.

The opposite concept is that of an **anti-symmetric relation**. Here it is forbidden that any two *different* vertices a and b are connected in both directions. It is allowed, however, for an element to be connected to itself, and also, for two different elements not to have any connection at all.

The real-world relationship `likes` is not symmetric (as there are many instances of person a liking person b but not the other way round) nor is it anti-symmetric (or there wouldn't be any "happily ever afters.").

Transitivity. A binary relation $R \subseteq A \times A$ is called **transitive** if whenever $(x,y) \in R$ and $(y,z) \in R$ then also $(x,z) \in R$. As a formula:

Transitive binary relation.

$$R \subseteq A \times A \text{ is transitive} \iff \forall x,y,z \in A.\, (x,y) \in R \text{ and } (y,z) \in R \implies (x,z) \in R$$

Many relations in mathematics are transitive, for example "less than or equal to" on the set of natural numbers. Also, the relationship "is congruent to" on the set of triangles. Most relationships from the real world are not transitive, and `likes` certainly isn't.

To explain how to change a given relation so that it becomes transitive we first introduce how relations are **composed**.

Composition of binary relations.

Given $R, S \subseteq A \times A$ define:

$$R;S \stackrel{\text{def}}{=\!=} \{(x,z) \in A \times A \mid \exists y \in A.\, (x,y) \in R \text{ and } (y,z) \in S\}$$

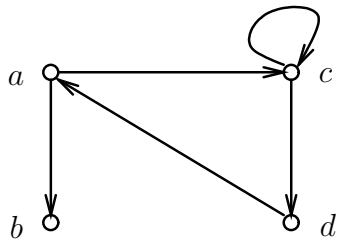
To define the **transitive closure** of a relation R we add all finite compositions of R with itself. As a formula:

$$\text{trans-closure}(R) \stackrel{\text{def}}{=\!=} R \cup R;R \cup R;R;R \cup \ldots$$

The result is the smallest transitive relation that contains R. We note that this is an infinitely long formula. In fact, it can be shown that it is not possible to define transitive closure with the usual means of first-order logic. Since SQL is equivalent to first-order logic, transitive closure is *not definable* in SQL. However, various database vendors have introduced extensions to their systems that allow the user to compute queries such as transitive closure.

In our example we get:

67



The transitive closure operation is related to the idea of a **path**: There is a path from vertex v to vertex v' exactly if (v, v') belongs to the transitive closure of the edge relation.

7.3 Order relations

If we combine the properties introduced in the three items above we get two very important concepts:

Order relation.

A relation $R \subseteq A \times A$ is called an **order relation** if it is reflexive, anti-symmetric, and transitive.

Equivalence relation.

A relation $R \subseteq A \times A$ is called an **equivalence relation** if it is reflexive, symmetric, and transitive.

We take a closer look at each of them.

We first note that the terminology "order relation" is justified, as the relation $\leq$ on the natural numbers (or the integers, or the reals) has the three required properties:

Reflexivity: Indeed, it is true for all numbers x that $x \leq x$.

Anti-symmetry: Yes, if $x \leq y$ and $y \leq x$ then it must be the case that $x = y$. In other words, we cannot have two *different* x and y for which both $x \leq y$ and $y \leq x$ hold.

Transitivity: Again, this is true: $x \leq y$ and $y \leq z$ together imply that $x \leq z$.

The various sets of numbers have another property which is not required for general order relations, namely, for any two numbers x and y we have $x \leq y$ or $y \leq x$. In general, we allow two elements to be **incomparable**, that is, neither (x,y) nor (y,x) belong to the order relation. An example is *divisibility*: Although it is reflexive, anti-symmetric, and transitive, there are incomparable numbers (with respect to divisibility), for example 3 and 5.

An example from computer science is given by *precedence graphs* which indicate which file must be compiled before which other files. "Makefiles" implement this idea.

Small order relations can be drawn as graphs where all edges point upwards and where those edges that follow from transitivity are left out. Here is an example based on divisibility:

68

7.4 Equivalence relations

You can think about an equivalence relation as being a "loose form of equality;" indeed, equality itself is (very trivially) reflexive, symmetric, and transitive. A better example (from mathematics) is the *congruence of natural numbers* which we discussed in Section 2.4. Recall that we say that x and y are *congruent modulo* m if x and y leave the same remainder when divided by m.

For an example more directly from computer science, we can look at ways to compare two implementations P_1 and P_2. Here are some possible equivalences, increasingly liberal:

1. P_1 and P_2 are actually the same file.
2. P_1 and P_2 are in different files, but the files have equal content ("duplicate files").
3. P_1 and P_2 are the same except for formatting ("white space").
4. P_1 and P_2 are the same as before but the order in which variables and methods are declared may now be different.
5. P_1 and P_2 are the same as before but may use different class, variable, and method names ("refactoring").
6. P_1 and P_2 are the same as before but in addition, one may have some methods in-lined ("code optimisation").
7. P_1 and P_2 show exactly the same behaviour when run ("specification" versus "implementation").

(You should know that the School considers every equality of submitted course work, except (7), a case of plagiarism.)

If two elements a and b are related by an equivalence relation, then we say that they are **equivalent**. The symbol that is often used for equivalence relations is $\approx$ (though congruence modulo m is commonly written as $\equiv$ or $\equiv_m$).

For every relation $R \subseteq A \times A$ we can construct the smallest equivalence relation containing it. To do so we simply construct the transitive closure of the symmetric closure of the reflexive closure of R:[4]

$$\text{equiv-closure}(R) \stackrel{\text{def}}{=\!=} \text{trans-closure}(\text{symm-closure}(\text{refl-closure}(R)))$$

Equivalence relations lead to classifications. If $\approx$ is an equivalence relation on a set A and if a is an element of A, we can look at the subset of all elements that are equivalent to a. This is called the **equivalence class** of a. The notation for this is:

Equivalence class of an element

$$[a]_{\approx} \stackrel{\text{def}}{=\!=} \{x \in A \mid x \approx a\}$$

It can be that $[a]_{\approx}$ is all of A, which means that every element of A is equivalent to a, but more often $[a]_{\approx}$ will be a proper subset of A. In that case we can look at an element b that does not belong to $[a]_{\approx}$. It now happens that the equivalence class of b, that is, $[b]_{\approx}$, is *disjoint* from $[a]_{\approx}$. The proof of this is very simple: If $[a]_{\approx}$ and $[b]_{\approx}$ had an element c in common, then we would have $a \approx c \approx b$ and hence $a \approx b$ by transitivity. This would mean that $b \in [a]_{\approx}$ contrary to our assumption. A picture may be helpful:

69

[4] The process that produces an order relation from an arbitrary relation is a bit more complex.

On the other hand, if $b \in [a]_\approx$ then in fact $[a]_\approx$ and $[b]_\approx$ must be the same. This is because for every $x \in [a]_\approx$ we have $x \approx a \approx b$ and hence $x \approx b$ by transitivity, and hence $x \in [b]_\approx$ by symmetry.

So we can conclude:

Theorem

If $\approx$ is an equivalence relation on a set A, and a, b are elements of A, then always one of the following two situations is true; either $a \approx b$ and $[a]_\approx = [b]_\approx$, or $a \not\approx b$ and $[a]_\approx \cap [b]_\approx = \emptyset$.

The consequence of this theorem is that an equivalence relation leads to a **classification** of the elements of A, by which we mean a decomposition of A into subsets (usually called "classes") with the properties:

disjointness No two classes overlap.
coverage Every element of A is contained in some class.

Here is a picture:

70

but I don't like it so much because it suggests that equivalent elements are somehow close to each other and this could be the wrong intuition. For example, if we take "congruent modulo 3" as our equivalence relation on the natural numbers $\mathbb{N}$, then we get the following picture:

71

We end up with three equivalence classes, which we can characterize globally as: the set of all numbers divisible by 3, the set of all numbers that leave remainder 1 when divided by 3, and the set of all numbers that leave remainder 2 when divided by 3.

Working with classifications via representatives. In a computer it is difficult to work with classifications directly because there is no immediate mechanism for representing sets. Also, the equivalence classes could be infinite sets and then there is no chance at all to represent the whole class. Instead, one works with **representatives** of the classes, that is, with individual elements at the level of the set A, and keeps in mind the notion of equality given by the equivalence relation. A picture:

72

The best example to illustrate this technique is the set $\mathbb{Q}$ of rational numbers which we discussed in Chapter 3. Every rational number can be written as a fraction in infinitely many ways: the fraction $\frac{2}{3}$ is the same rational number as the fraction $\frac{4}{6}$ etc. We can interpret this as saying that the rationals are constructed from the set of fractions (which is just a certain way of writing elements of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$) via the equivalence relation $\approx$ that is defined as

$$\frac{a}{b} \approx \frac{c}{d} \iff^{\text{def}} ad = bc$$

The arithmetic operations on rational numbers are defined via operations on the fractions:

$$\begin{array}{ccc} \frac{a}{b} + \frac{c}{d} & \stackrel{\text{def}}{=\!=} & \frac{ad+bc}{bd} \\ \frac{a}{b} \times \frac{c}{d} & \stackrel{\text{def}}{=\!=} & \frac{ac}{bd} \\ \frac{a}{b} \Big/ \frac{c}{d} & \stackrel{\text{def}}{=\!=} & \frac{ad}{bc} \end{array}$$

This works, because the operations are independent of the chosen representatives. For example, for addition the following is true (and it is a nice exercise in algebra to check this):

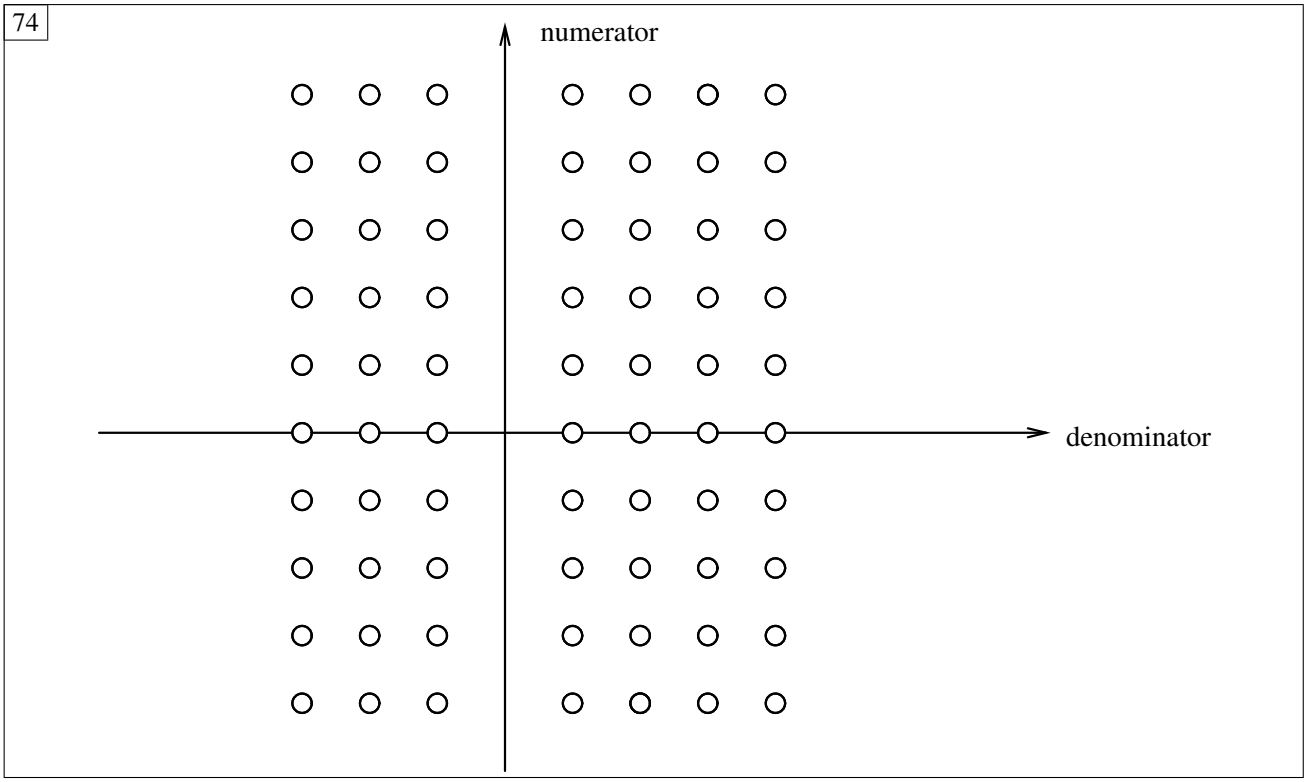
$$\frac{a}{b} \approx \frac{a'}{b'},\ \frac{c}{d} \approx \frac{c'}{d'} \quad \Longrightarrow \quad \frac{ad+bc}{bd} \approx \frac{a'd'+b'c'}{b'd'}$$

A picture of the situation may also be helpful:

73

An implementation of true rational numbers in Java (or any programming language) follows exactly the same pattern. Rationals are stored as two integers of arbitrary size (using the class `BigInteger`) and the operations are implemented using the equations above. In addition, one would implement the Euclidean algorithm as a method `normalise` for cancelling any common factor in numerator and denominator. When printing a rational, one would first normalise and then print so that the fraction displayed uses numbers as small as possible.

There is a nice 2D illustration of this construction:



Exercises

1. Consider the set $V = \{a,b,c,d,e,f\}$ and the relation $E = \{(a,d),(b,c),(c,f),(d,e),(e,a)\}$.
   (a) Draw a picture of this as a directed graph with vertices V and edges E.
   (b) Compute (separately) the edge-sets for the reflexive, symmetric, and transitive closure of E (no need to draw the resulting graphs).
   (c) Compute the edge-set for the equivalence relation generated by E, and draw this as a graph.
   (d) Write out the classes of the classification which is derived from this equivalence relation.
2. On the set of strings Σ^* over some alphabet Σ consider the binary relation $\lhd$ which holds between two strings s and t if s is a substring of t. Demonstrate that this is an order relation by spelling out what the defining properties mean in this context. Draw the order diagram in the style of Box 68 for all strings over the alphabet $\Sigma = \{a,b\}$ (including the empty string ε) up to length 2.

Practical advice

In the exam I expect you to be able to

- recognize when a given relation is reflexive, irreflexive, symmetric, anti-symmetric, and transitive;
- compute the reflexive, symmetric, or transitive closure of a given relation;
- recognize when a given relation is an order relation, or an equivalence relation (of course, it could be neither);
- draw the graph of an order relation as in Box 68;
- construct the classification that arises from an equivalence relation.

7 Relations

60

61

```
CREATE TABLE staff (name VARCHAR, phone INT, office INT);
```

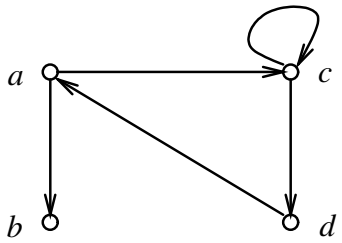
62

```
INSERT INTO staff VALUES ('Achim Jung', 44776, 213);
```

63

```
staff('Achim Jung', 44776, 213).
```

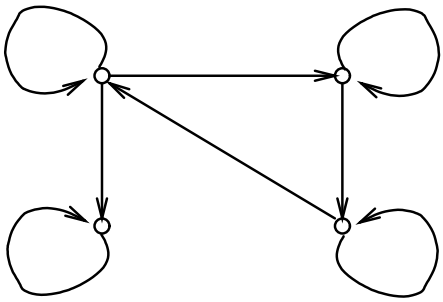
64



$A = \{a,b,c,d\}$

$R = \{(a,c),(a,b),(c,d),(c,c),(d,a)\}$

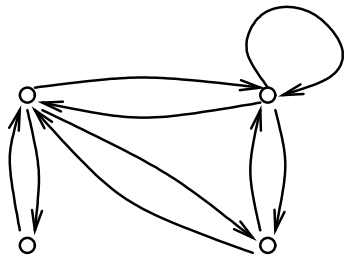
65



refl-closure(R) =

$\{(a,c),(a,b),(c,d),(d,a),(a,a),(b,b),(c,c),(d,d)\}$

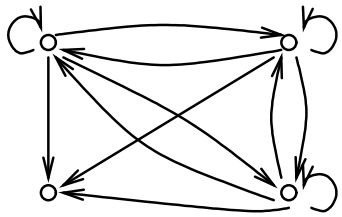
66



symm-closure(R) =

$\{(a,c),(a,b),(c,d),(c,c),(d,a),(c,a),(b,a),(d,c),(a,d)\}$

67

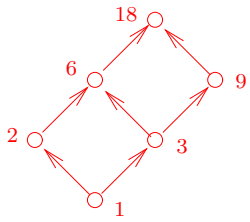


trans-closure(R) =

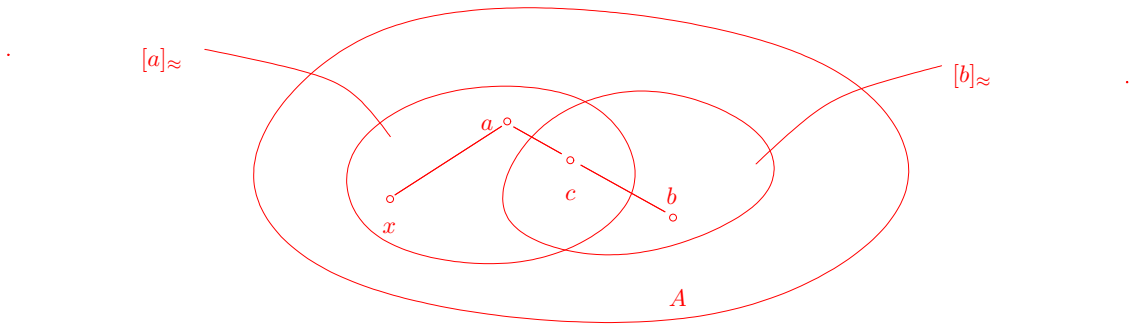
$\{(a,c),(a,b),(c,d),(d,a),$

$(a,a),(c,c),(d,d),(c,a),(d,c),(a,d).(c,b),(d,b)\}$

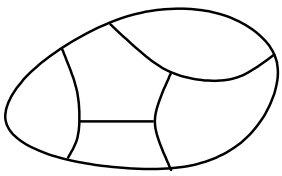
68



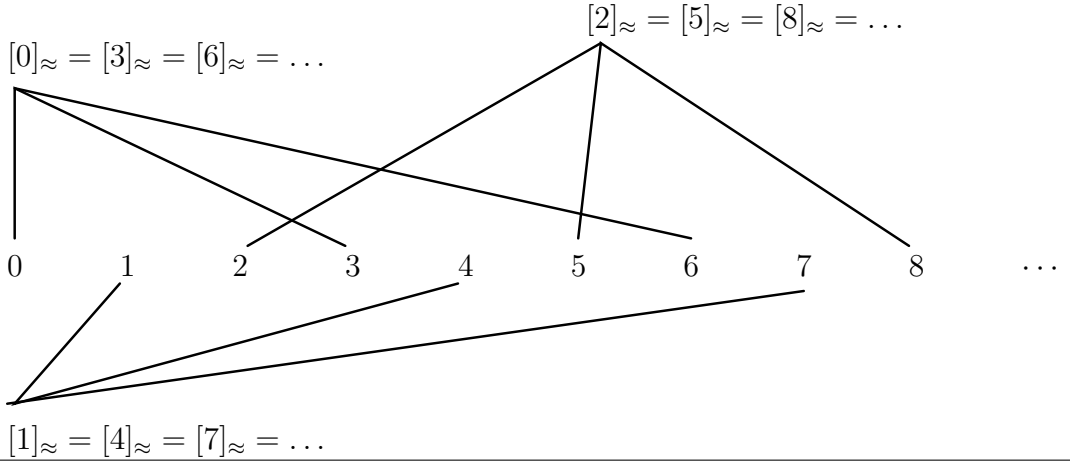
69



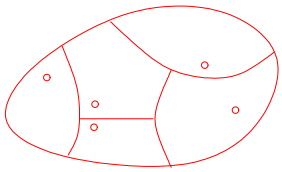
70



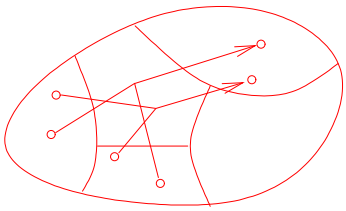
71


$[2]_\approx = [5]_\approx = [8]_\approx = \ldots$
$[0]_\approx = [3]_\approx = [6]_\approx = \ldots$
0 1 2 3 4 5 6 7 8 ...
$[1]_\approx = [4]_\approx = [7]_\approx = \ldots$

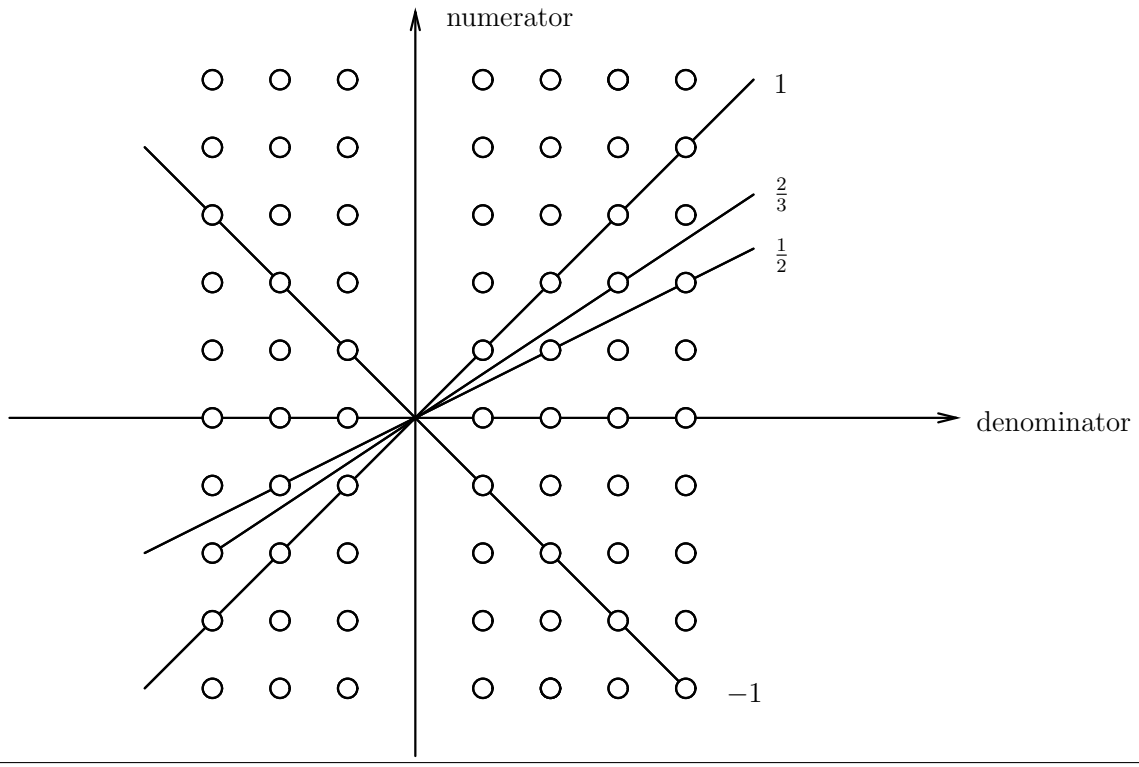
72



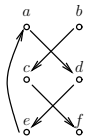
73



74

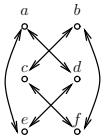

numerator
1
$\frac{2}{3}$
$\frac{1}{2}$
denominator
-1

Answers to exercises

1. (a) 

   (b) refl-closure$(E) = \{(a,d),(b,c),(c,f),(d,e),(e,a),(a,a),(b,b),(c,c),(d,d),(e,e),(f,f)\}$.
   symm-closure$(E) = \{(a,d),(b,c),(c,f),(d,e),(e,a),(d,a),(c,b),(f,c),(e,d),(a,e)\}$.
   trans-closure$(E) = \{(a,d),(b,c),(c,f),(d,e),(e,a),(d,a),(b,f),(e,d),(a,e),(a,a),(d,d),(e,e)\}$

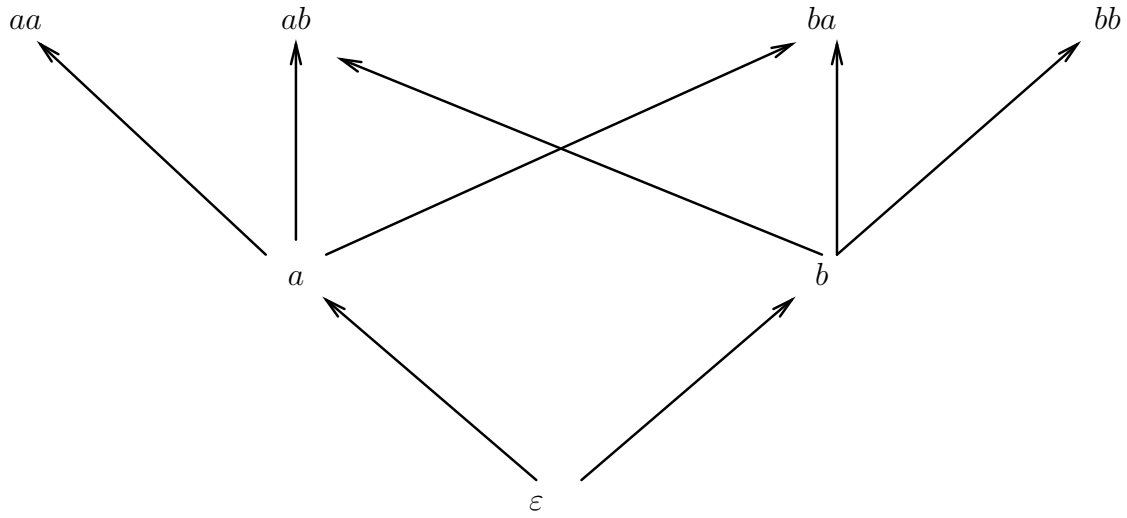
   (c) equiv-closure$(E) =$ $\{(a,a)\ ,(a,d),(a,e),$
   $(b,b),(b,c),(b,f),(c,c),(c,f),(c,b)$
   $(d,d),(d,e),(d,a),(e,e),(e,a),(e,d)$
   $(f,f),(f,b),\ (f,c)\}$

   (In the graph I left out out the self-connections)

   (d) We get two equivalence classes: $[a]_{\approx} = [d]_{\approx} = [e]_{\approx} = \{a,d,e\}$ and $[b]_{\approx} = [c]_{\approx} = [f]_{\approx} = \{b,c,f\}$

2. The relation is reflexive because a string is always a substring of itself. It is anti-symmetric because if a string s is a substring of a string t and also the other way around then they must have the same number of characters and in fact be equal. It is also clearly transitive. The picture:



8 Functions

8.1 The set-theoretic definition of functions

Example. Here is the implementation of a function in Java:

```
public int twice (int x) {return 2*x;}
```

It describes a *procedure* for doubling the given argument. It could have been implemented in many other ways, for example:

```
public int twice_add (int x) {return x+x;}
```

or:

```
public int twice_shift (int x) {return x<<1;}
```

How would we describe this function in Set Theory? For this we remember that Set Theory is about the *What*, not the *How*. To describe *what* the doubling function is doing, we collect the effect of the function *for all possible inputs*:

$$\{\ldots,(-1,-2),(0,0),(1,2),(2,4),\ldots\}$$

A more familiar format for this is the **value table**, which you may have seen before, especially in Statistics:

x	$\cdots$	-1	0	1	2	$\cdots$
y	$\cdots$	-2	0	2	4	$\cdots$

Note that the set theoretic function tells you nothing about how the result is computed, whether by multiplication, by addition, by shifting, or in some other way, but it is certainly precise in recording its *effect* on inputs.

You may also notice that the Java versions are different from the set theoretic function in two ways:

- `int` is not the same as $\mathbb{Z}$, as it has only finitely many elements;
- None of the three implementations actually works for all inputs the way we would like: for example, each of them will return the *negative* number -2147483648 when given the *positive* input 1073741824, which is a side effect of the limited range of the `int` data type which we discussed in Chapter 1.

Set Theory's way of dealing with functions is of course a special case of the more general concept of a relation, which was the topic of the last chapter. In our example, the function is a subset of $\mathbb{Z}\times\mathbb{Z}$.

The precise definition. For a binary relation $R\subseteq A\times B$ to qualify as a function, two conditions need to be satisfied:

Definedness For *every* $a\in A$ there must exist a pair (a,b) in R. In other words, for every argument a there must be a value b. With the help of first-order logic, one writes

$$\forall a\in A\ \ \exists b\in B.\ (a,b)\in R$$

Single valuedness For all $a\in A$, there is *only one* $b\in B$ such that (a,b) belongs to R. In other words, a function has only *one* value for every argument. As a formula:

$$\forall a\in A\ \ \forall b,b'\in B.\ (a,b)\in R\wedge(a,b')\in R\implies b=b'$$

A relation which satisfies these two conditions is called a **function** or (for emphasis) a **functional relation**.

If $R\subseteq A\times B$ is a functional relation then A is called the **domain**, and B the **co-domain** of the function. The set of all possible outcomes from the function, which we can define by the formula

$$\{b\in B\mid \exists a\in A.\ (a,b)\in R\}$$

is called the **range** of the function. This is written as $\mathrm{rng}(f)$ or $\mathrm{im}(f)$ (the **image** of f).

It is customary to denote functional relations with lower-case letters, starting with f. Also, instead of $f \subseteq A \times B$ one writes more suggestively,

$$f\colon A \to B$$

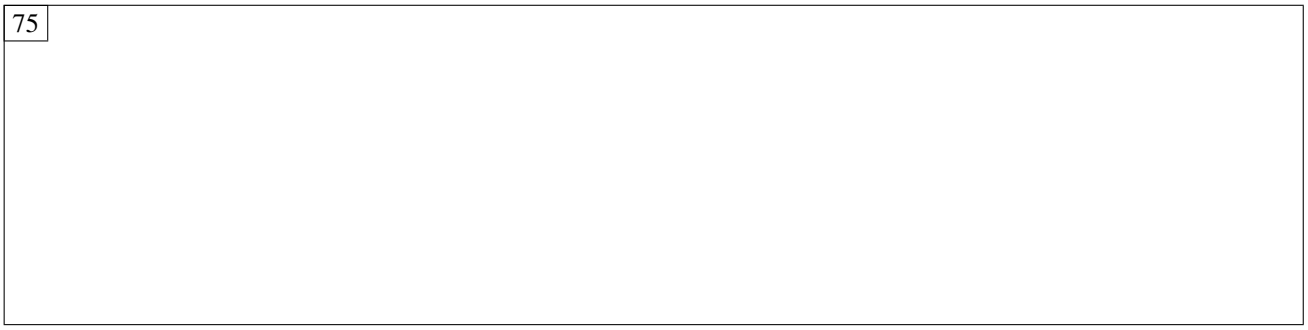
Finally, instead of $(a,b) \in f$ one writes $f(a) = b$.

Often a function has more than one argument, and certainly in computing this is the norm. Mathematically, this does not change anything: instead of a single set A for the input domain one considers a product of sets, for example, for two inputs one would have $A = A_1 \times A_2$. Everything else remains the same.

A relation that is single-valued but not defined for all inputs is called a **partial function**. Most computer implementations of mathematical functions are partial because of size limitations.

The many faces of functions. The idea of a function can be illustrated in a number of ways, and I have reminded you of "value tables" already. I recommend that you familiarise yourself with all of them, as each captures a particular aspect and you may need to go back to them when you learn more advanced concepts. We start with the following schematic drawing:

75



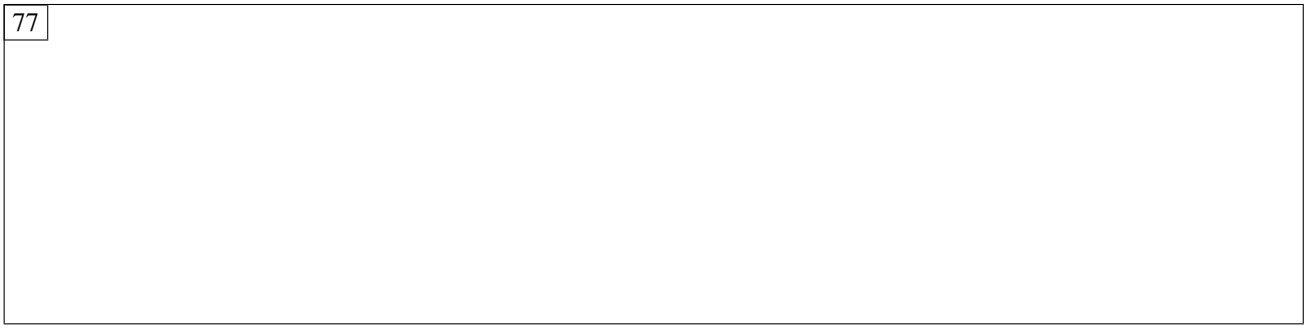
It illustrates well the idea that a function is a certain kind of *binary relation* between domain and co-domain. Definedness of a function means that from every dot on the left there is an arrow emanating. Single-valuedness means that there is never more than one arrow starting from the same dot. This is tantamount to saying that *exactly one* arrow emanates from a dot of the domain.

The next picture expresses the idea that a function is some *transformation* "mechanism" (perhaps implemented by a computer program) about which set theory knows nothing, as it only records the match of input values to output values:

76

Finally, we come to the image that very likely you encountered *first* in your school maths, namely, that of a **graph**.

77



Here the possible input values constitute the x-axis and corresponding output values have to be read off from the y-axis.

Unfortunately, graphs are only really useful if domain and co-domain are $\mathbb{R}$, the set of real numbers, and this is very rarely the case in Computer Science.

8.2 Special properties: injectivity

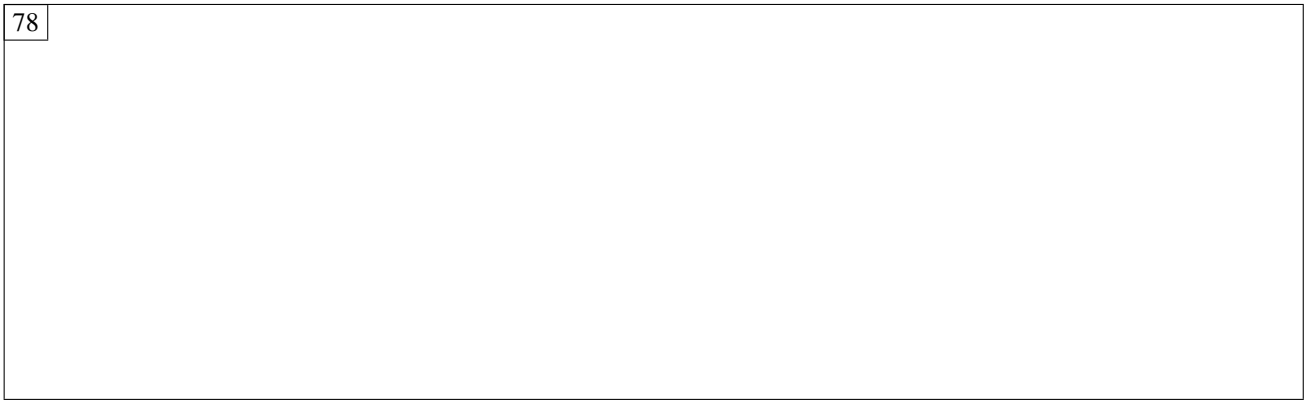
A function that will give different output for different input, is called **injective** or **one-to-one**. As a formula, we can write

$$\forall a, a' \in A.\ a \neq a' \implies f(a) \neq f(a')$$

or, alternatively,

$$\forall a, a' \in A.\ f(a) = f(a') \implies a = a'$$

Injective functions underlie many areas of computer science, for example, *coding theory*, where input "data vectors" (which are just tuples of bits) are mapped to much longer "code words" (which are again tuples of bits). If this is done cleverly, then small errors that change the code word (for example, during transmission over long distances) can be discovered and repaired. An image illustrates this idea of **error correcting codes** well:



Data compression (as in the `zip` utility) is also based on the application of an injective function, or otherwise we could not get the original data back. When the coding is non-injective, it is called **lossy**; examples of lossy compression algorithms are `mp3` (for music) and `jpeg` (for images). Another example that is non-injective *by design* is given by **hash functions**. A **cryptographic hash function** is also non-injective but is designed in such a way that no example that shows non-injectivity can be found without very large computational effort. Widely used cryptographic hash functions are `MD5` and `SHA-2`.

If the domain A of a function has more elements than the co-domain B, then no function from A to B can be injective. This is called the **pigeon hole principle**.

Using the idea of an injective function, we can refine our comparison of cardinalities of sets. In Chapter 5.4 we defined what it means for two sets to have *equal* cardinality, $|A| = |B|$. Now we can define $|A| \leq |B|$ if there is an injective function from A to B. The following can now be shown (but the proofs are anything but trivial):

Theorem 6 *1. If $|A| \leq |B|$ and $|B| \leq |A|$ then $|A| = |B|$.*

2. For any two sets A and B we have $|A| \leq |B|$ or $|B| \leq |A|$.

8.3 Special properties: surjectivity

A function for which the range is all of the co-domain is called **surjective** or **onto**. As a formula:

$$\forall b \in B\ \exists a \in A.\ f(a) = b$$

which says that every b has at least one a that is mapped to it.

To give an example from computer science, a hash function should be surjective onto the set of buckets, or otherwise some buckets remain empty and the hash table will be populated in an unbalanced way.

We have the following complement to the theorem above:

Theorem 7 *Assuming $A \neq \emptyset$, $|A| \leq |B|$ if and only if there is a surjective function from B to A.*

8.4 Special properties: bijectivity

If we combine injective with surjective, we get the concept of a **bijective function**, which can be illustrated well with a picture:

79

A bijective function from a (usually finite) set A back to itself is also called a **permutation**, as it neither loses nor confuses any elements, but merely re-arranges them. Cryptographic encodings are always permutations; they are chosen in such a way that it is computationally very expensive to reverse the permutation without the secret key. Examples are `AES` and `RSA`.

We can now make the definition of two sets having equal cardinality (Chapter 5.4) more precise: $|A| = |B|$ if there is a bijective function between A and B.

8.5 Forward image and pre-image; inverse function

If X is a subset of the domain A of a function f then we can wonder what values f takes on elements from X. One defines the **image** (sometimes called **forward image** for emphasis) as

$$f[X] \stackrel{\text{def}}{=\!=} \{b \in B \mid \exists a \in X.\, f(a) = b\}$$

Clearly, what we have called the "range of f" can now also be written as $f[A]$.

In the reverse direction, for a subset Y of the co-domain B one can ask which elements of A get mapped into Y. This is called the **pre-image**:

$$f^{-1}[Y] \stackrel{\text{def}}{=\!=} \{a \in A \mid f(a) \in Y\}$$

Some pictures may be useful:

80

For the pre-image note that the target set Y does not have to be a subset of the range of the function:

81

Forward image and pre-image of functions make excellent exam questions. For example, you could be asked to show that it is always true that $X \subseteq f^{-1}[f[X]]$ for any subset $X \subseteq A$. Let's make the argument here:

82

If a function is bijective then the pre-image of any element of B is a single point in A, so a function f^{-1} can be defined which goes from the elements of B to the elements of A. It is an **inverse** to f because $f(f^{-1}(b)) = b$ and $f^{-1}(f(a)) = a$ are true for it. You may note that as a relation, f^{-1} is indeed the reverse relation to f, as we defined in the previous chapter (in the paragraph about symmetry).

Asymmetric cryptography is based on the fact that there are bijective functions which are easy to compute but for whom it is difficult to find the algorithm for the inverse function; these are also known as **one-way functions**. The oldest and best known example is *modular exponentiation* which is the basis for the `RSA` crypto system.

8.6 Composing functions

If we have a function f from set A to set B, and a function g from B to set C, then it is very easy and natural to **compose** the two to get a function from A to C. The picture from Box 76 lends itself very nicely to illustrate this process:

83

whereas the usual mathematical picture from Box 77 is hopeless.

Also note that both mathematicians and computer scientists write composition *from right to left*:

$$g \circ f$$

which is really annoying, hard to learn, and easy to get wrong even after many years of practice (I know!).

8.7 Counting functions

If A is a finite set with n elements and B a finite set with m elements, then there are m^n many different functions from A to B. This is true because for every element a of A one can choose one of the m-many elements of B for the value of the function at a. There are n such choices to make and they are all independent of each other.

If A is a countable set (such as the natural numbers), and B has at least two elements, then there are uncountably many functions from A to B. As in Item 5.6 on page 23, it follows that for any programming language there are functions from $\mathbb{N}$ to $\mathbb{N}$, which can not be implemented in that language.[5]

To count *injective* functions we have to take into account that an element of B which has been chosen as the image of some $a \in A$ already may not be chosen again later. This gives the following formula for the number of injective functions from A to B:

$$m \times (m-1) \times (m-2) \times \cdots \times (m-n+1) = \frac{m!}{(m-n)!}$$

Note that this expression evaluates to 0 if $n > m$, as there are no injective functions in this case (pigeon hole principle).

A special case of the previous paragraph is when A and B have the same number of elements; in this case an injective function is automatically also surjective, that is, it is a bijection. Thus we get for the number of bijections from A to B:

$$m \times (m-1) \times (m-2) \times \cdots \times 1 = m!$$

[5] In the next semester you will learn that the choice of programming language is in fact irrelevant for this; if a function can not be implemented in one language then it can not be implemented in any other either.

Exercises

1. Consider the set $A = \{a, b, c, d\}$ and the relation $R = \{(a,a), (b,b), (b,c), (c,d)\} \subseteq A \times A$.
   (a) Draw a diagram of the situation in the style of Box 75.
   (b) Is R a function from A to A?
   (c) Is R^{-1} a function from A to A?
   (d) Make one change to the definition of R so that it becomes a permutation on A.
2. (a) Describe a function f from $\mathbb{N}$ to $\mathbb{N}$ which is injective but not surjective.
   (b) Likewise, describe a function g from $\mathbb{N}$ to $\mathbb{N}$ which is surjective but not injective.
   (c) Do the same for functions from $\mathbb{R}$ to $\mathbb{R}$.

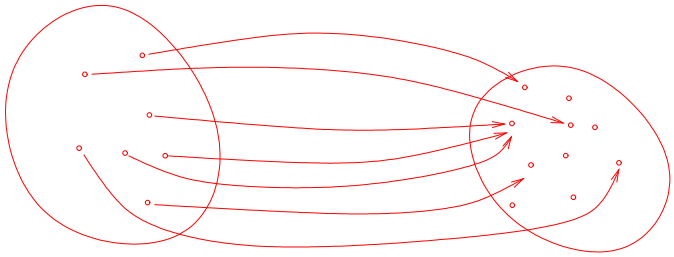
Practical advice

In the exam I expect you to

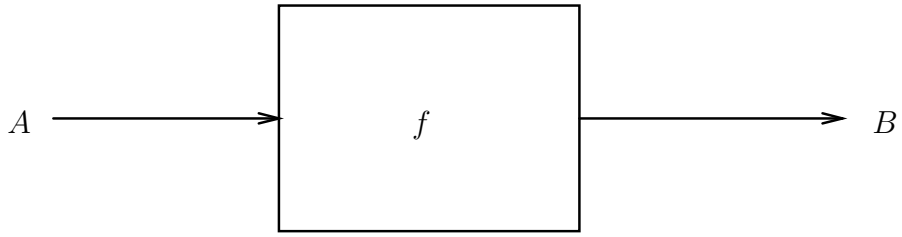
- know what properties a binary relation must have in order to be called a function: definedness and single-valuedness;
- know what injective, surjective, and bijective mean;
- be able to analyse Java methods to see whether they define functions, and if so, whether they are injective, surjective, or bijective;
- know that there are uncountably many functions from $\mathbb{N}$ to $\mathbb{N}$ and so most of these can not be implemented in a programming language.

8 Functions

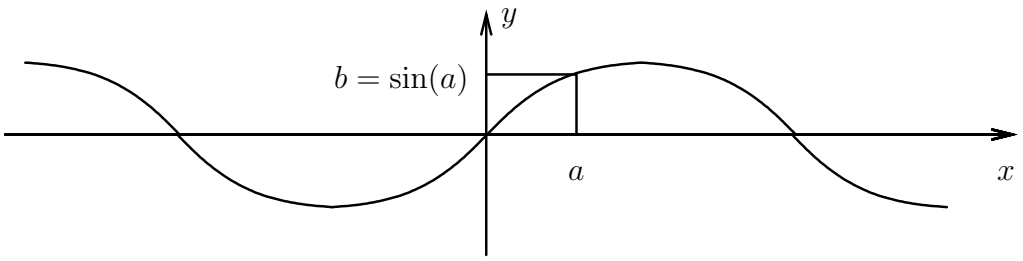
75



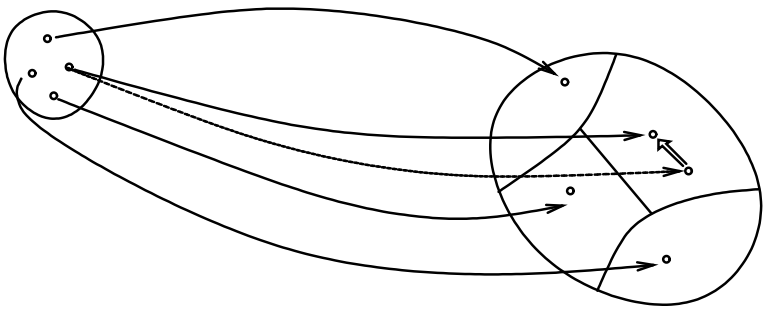
76



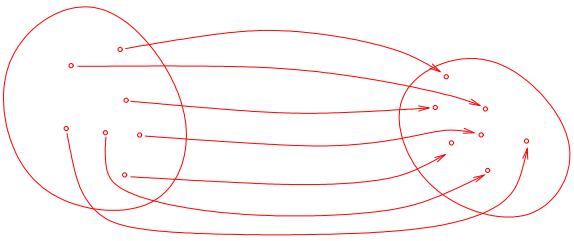
77



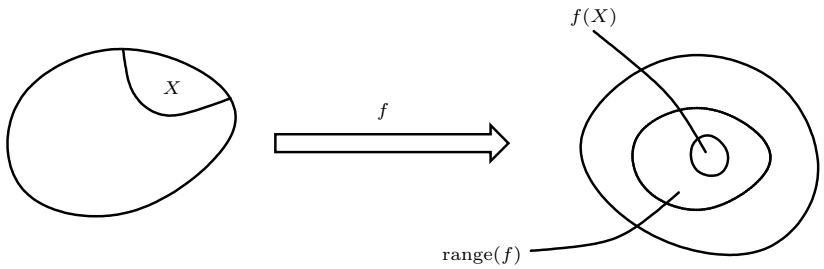
78



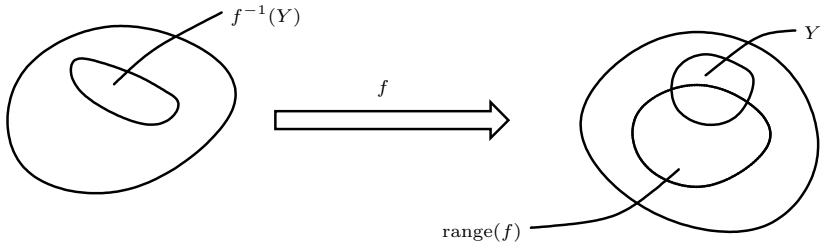
79



80



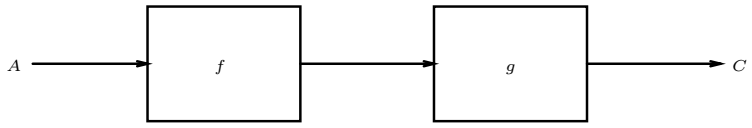
81



82

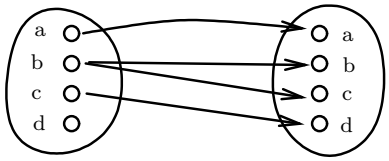
Take $a \in X$. Since functions are everywhere defined, there is $y \in B$ such that $(x,y) \in f$, in other words, $y = f(x)$. So by definition of the forward image we have $y \in f[X]$. Again since $(x,y) \in f$ we can now use the definition of pre-image to conclude $x \in f^{-1}[f[X]]$.

83



Answers to exercises

1. (a)



(b) It is not, because it is not single valued: There are two outputs assigned to input b: b and c.

(c) It is, because for every possible input there is exactly one output.

(d) We change the second entry of R from (b,b) to (d,b). Now for every input there is exactly one output, but furthermore, different inputs are mapped to different outputs (injectivity) and all possible outputs do indeed occur (surjectivity), and these two properties together say that the function is a bijection, or in other words, a permutation.

2. (a) $f(x) = x+1$ does the trick. It is not surjective because zero never appears as an output. (There are many other possibilities, for example, $f(x) = 2x$.)

(b) We can do the reverse of what is going on in (a): We map every $x \geq 1$ to $x-1$ and 0 to 0. The 0 appears twice as the output of both $x = 0$ and $x = 1$.

(c) There are functions in mathematics with the required properties: $\arctan(x)$ and $x^3 - x$, for example, but we can manufacture simpler examples by case distinction:

$$f(x) = \begin{cases} x+1 & \text{if } x \geq 0; \\ x-1 & \text{if } x < 0 \end{cases}$$

is injective but never outputs any of the numbers from the interval $[-1,1)$. Similarly,

$$g(x) = \begin{cases} x-1 & \text{if } x \geq 0; \\ x+1 & \text{if } x < 0 \end{cases}$$

produces every element in $[-1,1)$ twice.

9 Induction

Motivating example: Defining correct syntax. Up to here we have practised the definition of subsets by *selection*, as in

$$\{x \in \mathbb{N} \mid \exists b \in \mathbb{N}.\, x = b^2\}$$

which selects the perfect squares among the natural numbers. This is the prevailing method for defining sets in mathematics but it so happens that it is often not very convenient in computer science. The issue comes up especially when defining **syntax**, for example, the syntax of a programming language. To make this more concrete, consider the set of strings that only use the two square bracket characters "`[`" and "`]`", and which are *properly nested*. How should we define the set of these strings? It is easy to give examples,

"`[]`" "`[[]]`" "`[][]`" "`[[][]]`"

and easy to give *non*-examples,

"`][`" "`[[`" "`[]]`"

but is there a logical expression with which we can single out the correctly nested brackets from the set of arbitrary strings? Is there a systematic (programmable!) way to see that "`[[[][[]]][]]`" is correct but "`[[[][]]][]]`" is not?

The problem appears even more sharply if we look at the syntax of a programming language. How does a compiler check that a program is **syntactically correct**? How does it distinguish the syntactically correct (and semantically meaningless) fragment

```
boolean x = true;
int y = 5;
x=!x;
if ( y!=0 ) ...
```

from the incorrect

```
boolean x = true;
int y = 5;
x!=x;
if ( y=!0 ) ...
```

The technique I am presenting in this chapter is the basis for the answer to this question, although many of the practical details will have to wait until you take the module *Compilers and Languages* in the third year. To whet your appetite, the technique will not only make *syntax checking* possible but also be essential for *code generation*.

9.1 A dynamic way of generating a subset

The new method of specifying which elements belong to the subset is *incremental*; instead of characterising all elements in one go, we add more and more elements to it as we go along, following certain *rules*. For the set B of properly nested brackets, for example, we could have the following three rules:

Rule 1 The string "`[]`" belongs to B.

Rule 2 If s is a string that belongs to B, then "`[s]`" also belongs to B.
(In Java we would write `"[" + s + "]"` for this.)

Rule 3 If s and t are strings that belong to B, then st also belongs to B.
(In Java: `s + t`)

Let us use these rules to justify why the string "`[[[][[]]][]]`" from before belongs to B, in other words, that it is properly nested:

84

1.

2.

3.

4.

5.

6.

This style of defining a set is called an **inductive definition**. The first rule in our example above did not rely on some element in the set B already; this is called a **base case**. The other two rules are contingent on some strings having been generated before; they are called **inductive steps**. Instead of "justification" one usually says **derivation**.

9.2 Grammars

Since inductive definitions are so ubiquitous in computer science, some shorthand notation has been adopted from linguistics; it is called a **grammar**. In computer science, grammars are written down in **Backus-Naur Form** or **BNF**.[6] Our rules for well-balanced brackets ("*wbb*") would be written as a BNF grammar as follows:

wbb ::= `[]` | `[`*wbb*`]` | *wbb wbb*

The symbol "|" is read as "or" and the whole **grammar** as

> *A well-balanced bracket expression is either the string "`[]`," or a well-balanced bracket expression enclosed in brackets, or one well-balanced bracket expression following another one.*

Each rule in the grammar is also called a **production**.

It is important to note that in the last production of the grammar, where it is said that a well-balanced bracket expression can be one well-balanced bracket expression following another one, the two expressions *are independent of each other*. This is in contrast to the usual use of variables in mathematics or computer science, where different occurrences of the same variable refer *to the same entity*. So where the grammar says

wbb ::= *wbb wbb*

a correct instantiation of the right hand side would be

"`[] [[]]`"

[6] In the US this is often called **Backus Normal Form**. See `http://en.wikipedia.org/wiki/Backus-Naur_Form` for the history of this formalism and terminology.

In contrast, in mathematics or computer science, the statement

```
y = x + x;
```

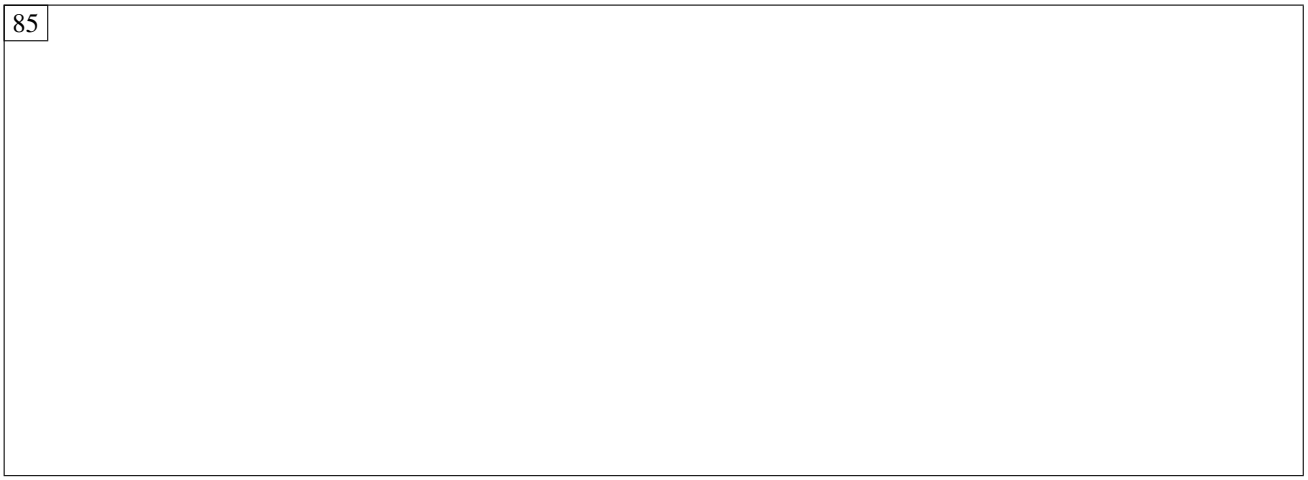
can never mean `y = 3 + 4`.

Summary: testing versus generation. The traditional way to define a subset uses a *testing condition* C with which to *select* those elements of a given set A which are meant to belong to the subset. An example for this was given at the beginning of this chapter. The general format is:

$$X \stackrel{\text{def}}{=\!=} \{x \in A \mid C(x)\}$$

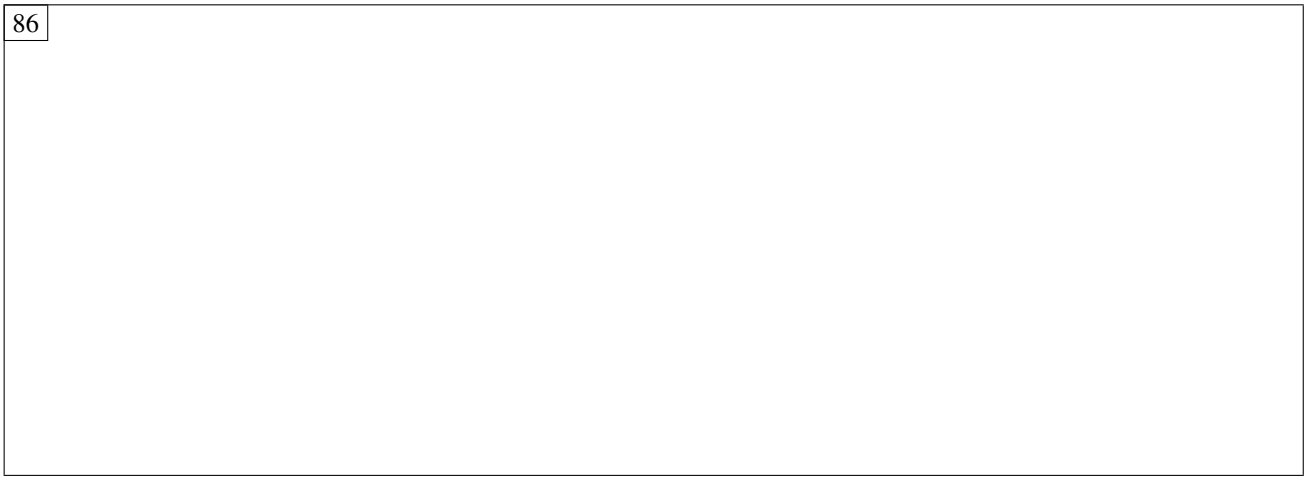
A picture for this in the style of flow charts could be drawn as follows

85



An inductive definition works differently; it allows one to successively *generate* elements of the subset. As a pictorial representation we could use:

86



The two methods have complementary strengths as we can see from the following examples:

$$F = \{x \in \mathbb{N} \mid 2^{2^x} + 1 \text{ is prime}\}$$

This set is defined by selecting those natural numbers x that pass the test "$2^{2^x} + 1$ is prime." It is easy to see that 2 is a member of F because $2^{2^2} + 1 = 17$ is a prime number. It is also (relatively) easy to see that 5 is not a member because $2^{2^5} + 1 = 4294967297 = 641 \times 6700417$, but in fact it is an *unsolved mathematical problem* whether F contains any elements greater than four.[7]

By contrast, consider the following inductively defined subset I of the integers:

[7] See `http://en.wikipedia.org/wiki/Fermat_number` for more information on this.

- 17 belongs to I.
- 20 belongs to I.
- If x and y belong to I then so does $x-y$.

Here it is easy to generate infinitely many members of I but just playing with the rules: $17, 20, 3, -3, 23, 6, -6, 0$, and so on. On the other hand, it is difficult to see whether a given number, say 7, belongs to I or not.

9.3 Proof by (structural) induction

Since the elements of an inductively set are generated one after the other it may seem impossible to state a property that is common to all of them, but in fact this is not difficult at all. The technique is called **proof by (structural) induction** and is best explained with an example: Let's prove that every initial segment of a string of well-balanced brackets contains at least as many opening brackets as closing brackets:

> *Base case:* The string to consider is "`[]`" and we can prove the statement by going through each of the three possible cases, where the initial segment is "ε" (empty string), "`[`", and "`[]`" itself. Indeed, it is true that there are never more closing than opening brackets in any of these.
>
> *Inductive case 1:* Suppose s is a string of brackets that satisfies the claim and consider "`[`s`]`". Now the initial segments are "ε", "`[`", "`[`s'" (where s' is an initial segment of s), and "`[`s`]`". In the first case there are no brackets at all, in the second there is only one opening one, in the third we can assume that s' contains at least as many opening as closing brackets and so this is also true about "`[`s'", and in the last case, the test is also true because both an opening and a closing bracket have been added to s.
>
> *Inductive case 2:* Suppose both s and t satisfy the condition and consider st. The initial segments are s' (where s' is an initial segment of s) and st' (where t' is an initial segment of t). In both cases the condition is again valid.

Note that a proof by induction consists of an argument for each of the rules of the inductive definition. In the base case(s) we must prove the statement outright; in an inductive case we may assume that the statement is true for the elements which are used to construct further elements.

We discussed proofs by induction before, in the context of the natural numbers. Well, it so happens that the natural numbers themselves can be seen as an inductively defined set of strings. The rule set is very simple:

> *base case:* The empty string is a natural number (representing zero).
> *inductive case:* If s is a natural number, then so is "$s|$" (representing $s+1$).

In this representation, natural numbers are strings of the symbol "|", for example, 5 is represented as "|||||".[8] To prove a property of all the natural numbers then amounts to giving an argument for zero, and giving an argument that works for $s+1$ assuming that the property holds for s.

9.4 Defining a function on an inductively defined set

Let us begin with an example for the set B of well-balanced brackets. The aim is to define a function $lon\colon B \to \mathbb{N}$ that gives the maximum *level of nesting* ("*lon*") for a string of well-balanced brackets. (Note that this function makes little sense for strings that are not well-balanced.) We start with a procedural definition in the style of a computer program:

I. Direct (procedural) definition. The idea is to go through the given string character by character from left to right. More formally:

- Create an integer variable `level` initialised to 0.
- Create an integer variable `maxlevel` initialised to 0.
- Begin to read the given string from left to right and for each character do the following:

[8] This is sometimes called the **unary representation** of numbers.

  - If the character is an opening bracket "`[`" increase the `level` variable and check whether it exceeds the value stored in `maxlevel`. If so, increase `maxlevel` by one.
  - If the character is a closing bracket "`]`" then decrease the `level` variable.

- When the end of the string has been reached and no more characters are to be processed, return the contents of `maxlevel`.

Note that while this seems correct, it is a rather unusual way of defining a mathematical function. As an alternative, let us exploit the knowledge that every element of B was constructed in a certain way:

IIa. Inductive definition.

base case: Assign 1 to the string "`[]`".

inductive case 1 If n has been assigned to s already, then assign $n+1$ to "`[`s`]`".

inductive case 2 If n has been assigned to s and m has been assigned to t, then assign $\max\{n,m\}$ to "st".

Note how this definition of *lon* follows directly the intuition that we have about the "level of nesting" in a string of brackets. It needs almost no algorithmic insight to define the function in this way.

IIb. Inductive definition (again, but more formal). To make the analogy with the definition of B even clearer, I write this again but in a more formal set-theoretic notation. For this remember that *lon* is a function from B to $\mathbb{N}$, and as such is really just a certain subset of $B \times \mathbb{N}$. We can define that subset in the style of Item 9.1:

base case The pair $(\texttt{[]}, 1)$ belongs to *lon*.

inductive case 1 If (s,n) belongs to *lon* then so does $(\texttt{[}s\texttt{]}, n+1)$.

inductive case 2 If (s,n) and (t,m) belong to *lon* then so does $(st, \max\{n,m\})$.

While this is very nice, we must check that we are actually defining a function this way. Indeed, when you look at the presentation IIb then it is obvious that we are defining a *relation* from B to $\mathbb{N}$, but for this to be a *function* we must prove that the relation is defined everywhere and that it is single-valued. Well, the first condition is automatically satisfied if the definition of the relation follows exactly the structure of the definition of the subset itself. This is definitely the case in our example *lon*.[9] Singlevaluedness, on the other hand, is *not* that easy and can indeed go wrong, but let's postpone this for the moment.

Terminology: If a function is given in the style of IIa or IIb, then we call it **inductively defined**, or we speak of a **function definition by structural induction**.

Application: Code generation. The technique of defining a function inductively is exactly the one that a compiler employs when generating the machine instructions that correspond to a program written in a high-level programming language. We illustrate this in a small example. Consider the program fragment

```
if ( condition )   block1
else   block2
```

The compiler will recursively translate *condition*, *block1*, and *block2* into sequences S_c, S_{b1}, and S_{b2} of machine instructions and then piece these together as follows:

[9] Haskell checks automatically whether this is the case or not.

87

The second condition for functions revisited. Remember that above we deferred the discussion of when inductively defined relations are single-valued. This is relevant because it is not always the case. To get your attention, consider arithmetic expressions defined by the following simple grammar:

$$\textit{expr} \quad ::= \quad \textit{numeric-literal} \quad | \quad \textit{expr} + \textit{expr} \quad | \quad \textit{expr} * \textit{expr}$$

We could try to define a function from expressions to natural numbers that assigns to an expression its actual value. So to "$3+4$" it should assign 7, and to "$4*5$" it should assign 20. This seems easy to do:

base case If the expression is a "numeric literal" assign that number.

inductive case 1 If the expression has the shape "*expr* + *expr*" then assign the sum of the numbers that you would assign to the two subexpressions.

inductive case 2 If the expression has the shape "*expr* $*$ *expr*" then assign the product of the numbers that you would assign to the two subexpressions.

However, we have a problem, because following the definition we are entitled to assign two different values to the expression "$3+4*5$": Since the grammar knows nothing about BODMAS, the problem is that "$3+4*5$" can be seen as having the shape of a *sum* of the subexpressions "3" and "$4*5$" and alternatively, as a *product* of the subexpressions "$3+4$" and "5". Following our inductive definition above, we are therefore entitled to assign 23 or 35 to "$3+4*5$".

In general it is very difficult (in fact, algorithmically impossible) to decide whether an inductive definition of a relation results in a single-valued relation or not. However, there are some good news as well:

1. If for every element of the inductively defined set we know which rule was last used in its derivation, then an inductively defined function is guaranteed to be single-valued. For example, this is the case with the grammar for the natural numbers (at the end of Item 9.3): The number "$|||||$" can only have been obtained in one way, namely, from "$||||$" by using the inductive rule which adds another stroke at the end of a string.

2. The situation in the previous item is very rare. Even some strings of well-balanced brackets can be derived in more than one way; for example, "`[][[]][]`" could have been constructed from "`[][[]]`" and "`[]`", or from "`[]`" and "`[[]][]`". However, if the function definition is not sensitive to such choices, then again it is single-valued. Our example of "*lon*" was of this type: Since it uses "max" to compute the level of nesting of a string of the shape *st*, it doesn't matter whether we read a succession of three substrings *s*, *t*, and *u* as *st* followed by *u*, or as *s* followed by *tu*. Since this is the only ambiguity in the definition of well-balanced brackets we are OK for single-valuedness of *lon*.

3. Sometimes the grammar can be re-written so that ambiguities are avoided or at least reduced. For example, for arithmetic expressions one might use the following:

$$\begin{array}{lcl} \textit{expr} & ::= & \textit{term} \mid \textit{sum} \\ \textit{term} & ::= & \textit{numeric-literal} \mid \textit{term} * \textit{term} \mid (\textit{sum}) \mid (\textit{term}) \\ \textit{sum} & ::= & \textit{term} + \textit{term} \mid \textit{term} + \textit{sum} \end{array}$$

which makes an explicit distinction between expressions which are "sums of subterms" and those which are not. It no longer allows us to read "$3+4*5$" as a product of "$3+4$" and "5" because "$3+4$" is not a "*term*".

4. If ambiguities are unavoidable then one can adopt a global convention that prefers "left-most" derivations over all others. This is a solution that is often found in computer science; for example both Haskell and Prolog will by default work with the first rule that is applicable.

9.5 Inductive definitions and set theory (not assessed)

The material presented above should be quite familiar to a computer scientist, given the close relationship between inductive definitions, grammars, and recursive datatypes. A mathematician, on the other hand, finds it strange to define sets in this way. She expects a set to be defined in the form

$$A = \{x \in X \mid x \text{ satisfies condition } c\}$$

and not given by a "dynamic process" that goes on forever. It is reassuring, then, to find that inductively given sets can also be defined in the standard way. This is what I will now try to explain. I include it here because it uses almost everything we have been doing in set theory.

We first recall the idea of the **powerset** of a set X, denoted by $\mathscr{P}X$, and already discussed in Chapter 6. We know that the elements of $\mathscr{P}X$ are all the subsets of X. To this we can now add that we have an order relation on $\mathscr{P}X$, given by subset inclusion: $A \subseteq A'$ means that every element of A is also an element of A'. In what follows, our base set X will be the set Σ^* of strings over the alphabet $\Sigma = \{[,]\}$, and we will show how the set B of strings of well-balanced brackets can be defined *without induction*.

The construction makes use of the following function f from $\mathscr{P}\Sigma^*$ to $\mathscr{P}\Sigma^*$:

$$f(A) \stackrel{\text{def}}{=\!=} \{[\,]\} \cup \{[s] \mid s \in A\} \cup \{st \mid s \in A, t \in A\}$$

You should read this carefully: It is defined for *arbitrary* subsets A of Σ^* and returns a set that contains the string "$[\,]$" plus all the strings of the form "$[s]$" and "st" where s and t are in A. (You will note the close similarity between f and the inductive definition of B but there is nothing "inductive" about f because it is defined for all subsets.)

88

$$\begin{aligned} f(\emptyset) &= \{[\,]\} \\ f(\{[\,][\,]\}) &= \{[\,][\,][\,],\ [\,][\,][\,][\,],\ [\,][\,][\,][\,][\,][\,]\} \end{aligned}$$

The function f preserves the order relation; one says that it is **order-preserving** or **monotone**. Here is the formal definition:

$$A \subseteq A' \implies f(A) \subseteq f(A')$$

It is clear that f has this property: The more elements we have in the input set A, the more possibilities we have for constructing elements that belong to $f(A)$. From monotonicity we can show that if we apply f to the empty set, then apply f again to the result, then again to that output, and so on, we get a sequence of bigger and bigger sets. As a (infinitely long) formula:

$$\emptyset \subseteq f(\emptyset) \subseteq f(f(\emptyset)) \subseteq \ldots$$

Here is the proof:

89

$$\begin{array}{rcl} \emptyset & \subseteq & f(\emptyset) \\ f(\emptyset) & \subseteq & f(f(\emptyset)) \\ \emptyset & \subseteq & f(\emptyset) \subseteq f(f(\emptyset)) \end{array}$$

always holds

because f is monotone

putting the last two inequalities together

Now we give the first definition of the set B of well-balanced brackets:

$$B \stackrel{\text{def}}{=\!=} \bigcup_{n=0}^{\infty} f^n(\emptyset)$$

which means that we are combining all those bigger and bigger sets $f^n(\emptyset)$ into one single subset of Σ^*. We could say that this expression "*generates B from below*", very much in the spirit of the inductive definition itself.

9.6 Fixpoints (not assessed)

We could stop here but you could object and say that an expression such as $\bigcup_{n=0}^{\infty} f^n(\emptyset)$ still has a "dynamic feel" to it. Indeed, we can do better. First we observe that B is a **pre-fixpoint** of f; by this I mean that

$$f(B) \subseteq B$$

is true. (For a **fixpoint** we would have equality.) To prove it, we consider each element x of $f(B)$. By the definition of f there are three cases to consider: If $x = []$ then this clearly belongs to B because it already belongs to $f(\emptyset)$. If x is of the form $[s]$ with $s \in B$, then s must be contained in one of the $f^n(\emptyset)$. We can conclude that $x = [s]$ therefore belongs to $f^{n+1}(\emptyset)$ and this is a subset of B by construction. Finally, if $x = st$ with s and t elements of B, then we must have $s \in f^n(\emptyset)$ for some $n \in \mathbb{N}$ and $t \in f^m(\emptyset)$ for some $m \in \mathbb{N}$. Let k be the larger of n and m; then st is contained in $f^{k+1}(\emptyset)$ and this again is contained in B by construction.

We now consider the set of *all pre-fixpoints* of f:

$$\text{Pre} \stackrel{\text{def}}{=\!=} \{A \subseteq \Sigma^* \mid f(A) \subseteq A\}$$

Pre is a set of subsets of Σ^*, or we could say, it is a subset of $\mathscr{P}\Sigma^*$. As we have shown, B is an element of Pre. Another element is our base set Σ^* itself, because it is the largest subset of Σ^*.

We observe that the definition of Pre is a perfectly simple set-theoretic formula, devoid of any "dynamics". Our aim now is to use Pre for a second definition of B, one that we could dub "*generation from above*".

This is what we do: we define B' as the intersection of all the elements of Pre and then show that $B = B'$. Let's do this formally: First the definition of B':

$$B' \stackrel{\text{def}}{=\!=} \bigcap \text{Pre} = \{x \in \Sigma^* \mid \forall A \in \text{Pre}.\, x \in A\}$$

We have already shown that B is a pre-fixpoint of f and since B' is the intersection of all such subsets of Σ^*, we clearly have $B' \subseteq B$.

For the other inclusion, i.e. $B \subseteq B'$, we first show that B' itself is a pre-fixpoint of f, that is, $f(B') \subseteq B'$. For this we need to show that $f(B') \subseteq A$ for every $A \in \text{Pre}$. But this is easy; let A be an arbitrary element of Pre:

90

$$\begin{aligned} B' &\subseteq A && \text{by definition of } B' \\ f(B') &\subseteq f(A) && \text{by monotonicity of } f \\ f(A) &\subseteq A && \text{because } A \in \mathrm{Pre} \\ f(B') &\subseteq A && \text{by transitivity of } \subseteq \end{aligned}$$

So indeed, we have $f(B') \subseteq B'$ and hence $B' \in \mathrm{Pre}$. Now we can show that every term in the long chain of sets with which we defined B, is a subset of B'. The proof is an induction over the natural numbers:

91

$$\begin{aligned} n = 0: \quad f^0(\emptyset) = \emptyset &\subseteq B' && \text{trivially} \\ n \to n+1: \quad f^n(\emptyset) &\subseteq B' && \text{induction hypothesis} \\ f^{n+1}(\emptyset) = f(f^n(\emptyset)) &\subseteq f(B') && \text{monotonicity} \\ f(B') &\subseteq B' && \text{because } B' \in \mathrm{Pre} \\ f^{n+1}(\emptyset) &\subseteq B' && \text{transitivity of } \subseteq \end{aligned}$$

Because every term in the chain belongs to B', so does the union of them all, that is, the set B. In other words, we have $B \subseteq B'$, as we wanted.

The two inclusions together give us $B = B'$, and we see that the expression

$$\bigcap \mathrm{Pre} = \bigcap \{A \subseteq \Sigma^* \mid f(A) \subseteq A\}$$

indeed defines the set of well-balanced brackets without recourse to any "dynamic process".

It remains to observe that this construction can be performed for any inductive definition (or any grammar); all that needs to be changed is the definition of f.

Exercises

1. Let X be the smallest set of natural numbers such that the following three conditions are satisfied:
   - $2 \in X$;
   - $3 \in X$;
   - if $x \in X$ and $y \in X$, then $xy \in X$ (where xy is the product of x and y).

   (a) Apply the rules repeatedly to demonstrate that 36 is an element of X.

   (b) Argue that 35 is not an element of X by giving (and proving) that all elements of X satisfy a certain property.

2. Consider the following inductive definition of a set of strings (written as a grammar for brevity):

   $$S \quad ::= \quad \texttt{b}SS \quad | \quad \texttt{a}S \quad | \quad \texttt{a}$$

   and the string `aabbaaa`.

(a) Find a derivation for this string.

(b) Use structural induction to show that every string that can be derived from the grammar will have strictly more `a`'s than `b`'s.

(c) Give an example of a string with strictly more `a`'s than `b`'s, which can *not* be generated by the grammar.

(d) Extend the grammar so that *all* strings with strictly more `a`'s than `b`'s (and no others) can be generated.

Practical advice

In the exam I expect you to be able to

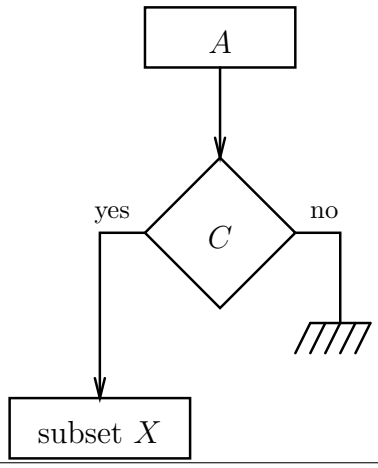
- read an inductive definition and generate some elements from it;
- read an inductive definition that is given in the form of a grammar (Section 9.2);
- prove a simple property that all elements of the inductively defined set satisfy *using structural induction*;
- find such a simple property yourself;
- argue that a certain string/number can not be an element of the inductively defined set because it does not satisfy the property you proved to hold for elements;
- work with relations and functions which are themselves defined by induction;
- prove a statement about natural numbers by induction.

9 Induction

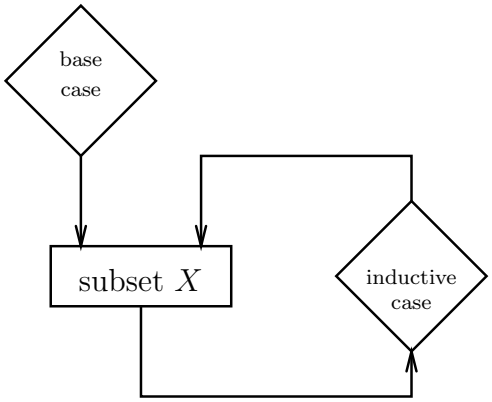
84

1. By Rule 1, "[]" belongs to B.
2. By Rule 2, and using the string generated in Step 1, "[[]]" belongs to B.
3. By Rule 3, and using the strings generated in Steps 1 and 2, "[][[]]" belongs to B.
4. By Rule 2, and using the string generated in Step 3, "[[][[]]]" belongs to B.
5. By Rule 3, and using the strings generated in Steps 4 and 1, "[[][[]]][]" belongs to B.
6. By Rule 2, and using the string generated in Step 5, "[[[][[]]][]]" belongs to B.

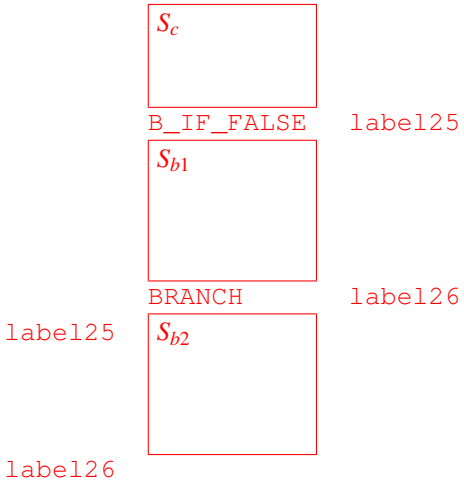
85



86



87



88

$$f(\emptyset) = \{[\,]\}$$

$$f(\{][\}) = \{[\,],\ [\,][\,],\ ][\,][\}$$

89

$\emptyset$	$\subseteq$	$f(\emptyset)$		always holds
$f(\emptyset)$	$\subseteq$	$f(f(\emptyset))$		because f is monotone
$\emptyset$	$\subseteq$	$f(\emptyset)$	$\subseteq$ $f(f(\emptyset))$	putting the last two inequalities together

90

B'	$\subseteq$	A	by definition of B'
$f(B')$	$\subseteq$	$f(A)$	by monotonicity of f
$f(A)$	$\subseteq$	A	because $A \in \mathrm{Pre}$
$f(B')$	$\subseteq$	A	by transitivity of $\subseteq$

91

$$
\begin{array}{rrcll}
n = 0: & f^0(\emptyset) = \emptyset & \subseteq & B' & \text{trivially} \\
n \to n+1: & f^n(\emptyset) & \subseteq & B' & \text{induction hypothesis} \\
f^{n+1}(\emptyset) = f(f^n(\emptyset)) & & \subseteq & f(B') & \text{monotonicity} \\
f(B') & & \subseteq & B' & \text{because } B' \in \text{Pre} \\
f^{n+1}(\emptyset) & & \subseteq & B' & \text{transitivity of } \subseteq
\end{array}
$$

Answers to exercises

1. (a) $2 \in X$ by Rule 1, $2 \times 2 = 4 \in X$ by Rule 3, $3 \in X$ by Rule 2, $3 \times 3 = 9 \in X$ by Rule 3, and $4 \times 9 = 36 \in X$ by Rule 3.

   (b) The property I would like to prove by induction is "All elements of X are divisible by 2 or by 3 (or both)".

   Base case 1: The statement is true for $x = 2$ as this is divisible by 2.

   Base case 2: The statement is true for $x = 3$ as this is divisible by 3.

   Inductive step: If x and y are members of X then by induction hypothesis, x is divisible by either 2 or 3, and then this is also true for any multiple of x, so it is true for $x \times y$.

   The number $35 = 5 \times 7$ is not divisible by either 2 or 3, so can't be a member of X.

2. (a) Re-writing the grammar, we have one base case,

   - "a" is a valid string in S

   and two inductive steps:

   - If s and t are from S, then so is bst.
   - If s is from S then so is as.

   The derivation: a $\in S$ by Rule 1, baa $\in S$ by Rule 2, bbaaa $\in S$ by Rule 2, abbaaa $\in S$ by Rule 3, and aabbaaa $\in S$ by Rule 3.

   (b) This is true for the base case, a. If it is true for s and t then the string st has at least 2 more a's than b's, so bst has at least 1 more a than b's, so the statement remains true. The reasoning for Rule 3 is trivial.

   (c) aba

   (d) I suggest using the grammar:

   $$S \ ::= \ \texttt{b}SS \ \mid \ S\texttt{b}S \ \mid \ SS\texttt{b} \ \mid \ \texttt{a}S \ \mid \ \texttt{a}$$

   (The really interesting question, however, is whether these rules are sufficient to generate *every* string with more a's than b's. I believe they are but my proof is quite long and complicated.)

10 Systems of linear equations and Gaussian elimination

10.1 Linear equations

An equation can have more than one **unknown**, as in

$$x^2 + y^2 = 5$$

An equation with one or more unknowns is called **linear** if the unknowns appear on their own, without exponent (that is to say, with exponent 1). In other words, products of unknowns are not allowed. The example above is not linear (but "quadratic") because it contains the product of x with itself and also the product of y with itself. The following *is* a linear equation (with three unknowns)

$$3x - 2y + z = 7$$

A general linear equation with n unknowns can be written as

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$$

where it is understood that the x's are the unknowns, the a's and b are **placeholders** or **parameters** or **coefficients** which in any concrete instance would be actual numbers.

A linear equation which contains more than one variable will usually have many solutions. For example, the equation

$$2x - y = 0$$

has the solutions $x = 1, y = 2$, and $x = 2, y = 4$, and $x = 3, y = 6$, and so on. In order to pin this down to *exactly one solution* we need *as many equations as there are unknowns*. For example, the system of linear equations

$$\begin{aligned} 2x - y &= 0 \\ x + y &= 6 \end{aligned}$$

has only *one* assignment for x and y which makes *both* equations *simultaneously* true, namely, $x = 2, y = 4$.

10.2 Gaussian elimination

In the language of Computer Science, Gaussian elimination is best described as a *recursive algorithm*. It has a **base case** and a **general case**:

Base Case There is only one equation and one unknown: $ax = b$. This can be solved directly: $x = b/a$.

General Case There are n equations and each contains n unknowns: Then we use the first equation to **eliminate** the first unknown in the remaining $n-1$ equations. Those new $n-1$ equations then only contain $n-1$ unknowns and Gaussian elimination can be applied recursively to those.

Once Gaussian elimination returns the values for those $n-1$ unknowns, they can be substituted into the first equation in which then only one unknown remains. This can be solved with the Base Case.

Time for an example:

92

$$\begin{aligned} x_1 + 5x_2 - 2x_3 &= -11 && \text{(Eq. 1)} \\ 3x_1 - 2x_2 + 7x_3 &= 5 && \text{(Eq. 2)} \\ -2x_1 - x_2 - x_3 &= 0 && \text{(Eq. 3)} \end{aligned}$$

This is the General Case as there is more than one unknown. We use equation 1 to eliminate x_1 from equations 2 and 3. We do this by subtracting 3 times equation 1 from equation 2, and adding twice equation 1 to equation 3:

93

Now we apply Gaussian elimination recursively to the smaller system consisting of equations 2 and 3 which each contain two unknowns, and temporarily forget about equation 1. Once more we have to follow the General Case: we use the second equation to eliminate x_2 in the third. For this we add 9 times equation 2 to 17 times equation 3:

94

We get one equation (the third one) with *one* unknown. It can be solved by the Base Case: $x_3 = -32/32 = -1$.

Now we go back one step in the recursion and substitute $x_3 = -1$ in the second equation and get:

95

The second equation can now be solved by the Base Case and we get $x_2 = (38+13)/(-17) = -3$.

We return to the very first step in the algorithm and replace x_2 and x_3 in the first equation by the values that we have computed:

96

Simplifying the first equation gives $x_1 = -11-2+15 = 2$. All three unknowns have been computed; the algorithm stops.

10.3 A more compact notation

The unknowns themselves are not affected by Gaussian elimination as all computation happens on the numeric coefficients in front of them. This suggest to only write out the coefficients and leave the unknowns implicit. In our example, the picture looks like this

97

$$\left(\begin{array}{rrr|r} 1 & 5 & -2 & -11 \\ 3 & -2 & 7 & 5 \\ -2 & -1 & -1 & 0 \end{array}\right)$$

We include the vertical line to remind us that the last column refers to the right hand sides of the given equations, not to another unknown.

Rather than go through the same example again, we do Gaussian elimination on a second example in this new notation.

$$\left(\begin{array}{rrr|r} 3 & 1 & 5 & 3 \\ -3 & 1 & -2 & -5 \\ 3 & -1 & 7 & 10 \end{array}\right)$$

Eliminating the first variable in equations 2 and 3:

98

Eliminating the second variable in equation 3:

99

Simplifying the last equation:

100

We can now read off that $x_3 = 1$ and as described above, use this value in the first two equations, etc. Alternatively, we can continue working in the compact notation: Using the last line we eliminate the entries in the third column in lines one and two.

101

Now we use the second line and use it to eliminate the second entry in the first row.

102

We finish by dividing the first line by 6 and the second line by 2 so as to get all ones on the diagonal:

103

We can read off the solution by putting back the unknowns:

104

$$\begin{pmatrix} x_1 & & & = & 1/6 \\ & x_2 & & = & -5/2 \\ & & x_3 & = & 1 \end{pmatrix}$$

Summary and useful hints

- Every unknown corresponds to precisely one column of entries in the compact notation. The last column corresponds to the numbers on the right-hand side of the equations.
- Be careful with the signs of the coefficients when translating, and when copying from one step in the algorithm to the next.
- Always work with the compact notation, not the explicit equations containing variables or you will inevitably get confused.
- Neat work will help you avoid common mistakes, such as dropping a minus sign and mis-copying entries.
- Avoid fractions until the very end. For example, in

  $$\left(\begin{array}{cc|c} 2 & -3 & 5 \\ 3 & 7 & -1 \end{array}\right)$$

  don't rewrite to

  $$\left(\begin{array}{cc|c} 2 & -3 & 5 \\ 0 & 7-\frac{3}{2}\times(-3) & -1-\frac{3}{2}\times 5 \end{array}\right) = \left(\begin{array}{cc|c} 2 & -3 & 5 \\ 0 & \frac{23}{2} & -\frac{17}{2} \end{array}\right)$$

  by subtracting $\frac{3}{2}$ of the first line, but first multiply suitably as in

  $$\left(\begin{array}{cc|c} 2 & -3 & 5 \\ 6 & 14 & -2 \end{array}\right)$$

  and then subtract an *integer multiple* (here 3 times the first line):

  $$\left(\begin{array}{cc|c} 2 & -3 & 5 \\ 0 & 1-3\times(-3) & -2-3\times 5 \end{array}\right) = \left(\begin{array}{cc|c} 2 & -3 & 5 \\ 0 & 23 & -17 \end{array}\right)$$

- Know your sign rules:
  - subtracting a negative number from a negative number:

    $$-x-(-y) \text{ is the same as } -x+y \text{ which is the same as } y-x$$

  - multiplying negative numbers:

    $$(-x)\times(-y) \text{ is the same as } x\times y$$

- Don't even think about using the highschool "substitution method" if there are more than two equations!

- Despite all the good advice I am trying to give, it's still very easy to make mistakes in Gaussian elimination, so when you have worked out a solution *check* it by substituting the values back into the equations. It takes very little time.
- Abstracting away from the actual values, the first phase of Gaussian elimination leads to a regular even-spaced "staircase" that separates the zeros created by the algorithm from the rest:

$$\left(\begin{array}{cccc|c} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{array}\right) \longrightarrow \cdots \longrightarrow \left(\begin{array}{cccc|c} \bullet & * & * & * & * \\ 0 & \bullet & * & * & * \\ 0 & 0 & \bullet & * & * \\ 0 & 0 & 0 & \bullet & * \end{array}\right)$$

The term often used in books for this pattern is **echelon form**. In the second phase, one creates zeros above the staircase. This can be pictured as follows:

$$\left(\begin{array}{cccc|c} \bullet & * & * & * & * \\ 0 & \bullet & * & * & * \\ 0 & 0 & \bullet & * & * \\ 0 & 0 & 0 & \bullet & * \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} \bullet & * & * & 0 & * \\ 0 & \bullet & * & 0 & * \\ 0 & 0 & \bullet & 0 & * \\ 0 & 0 & 0 & 1 & * \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} \bullet & * & 0 & 0 & * \\ 0 & \bullet & 0 & 0 & * \\ 0 & 0 & 1 & 0 & * \\ 0 & 0 & 0 & 1 & * \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & 0 & 0 & 0 & * \\ 0 & 1 & 0 & 0 & * \\ 0 & 0 & 1 & 0 & * \\ 0 & 0 & 0 & 1 & * \end{array}\right)$$

10.4 The special cases

The presentation of Gaussian elimination as a recursive algorithm makes it very easy to analyse the special cases. All special cases are to do with zeros appearing in the "wrong place". We discuss each in turn and give an example. The last special case will force us to generalise the Gaussian elimination algorithm.

Special Base Case 1: contradictory equation. The base case concerns one equation with one unknown: $ax = b$. It can happen that $a = 0$, so the equation really reads $0 = b$. If, furthermore, it is the case that $b \neq 0$, then the equation is *contradictory* and can not be solved. In this case, the algorithm should exit and indicate that a solution cannot be found, for example, by raising an exception. If the original system had several equations, and the base case was reached through a series of recursive calls, then the overall result is still that there is **no solution**. Example:

105

$$\begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ 2x_1 & - & 4x_2 & = & 7 \end{array} \longrightarrow$$

Special Base Case 2: irrelevant equation. This is like the previous case but in fact both sides are zero, that is, in $ax = b$ both a and b are zero and the equation reduces to $0 = 0$. This means that *the equation does not constrain the value of x in any way*. The answer of the algorithm should therefore be that this unknown **can be chosen freely**. Example:

106

$$\begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ 2x_1 & - & 4x_2 & = & 6 \end{array} \longrightarrow$$

In this example the second equation is redundant (it is exactly twice the first equation), so in actual fact we are left with only one equation for two unknowns. We can choose x_2 freely to be any number and once we have done this, we have to set x_1 to $3+2x_2$. We write this as

$$\begin{array}{rcl} x_1 & = & 3+2x_2 \\ x_2 & : & \text{chosen freely} \end{array}$$

Special Recursive Case 1: can't use the first equation for eliminating the first unknown. If the coefficient of the first unknown in the first equation is zero, then no matter how often the first equation is added to another one, the first variable of that other one is not affected.

In this case, we simply exchange the first equation with another one for which the first coefficient is *not* zero, and run the algorithm on this rearranged system. (In fact, it is useful to "pick" the most suitable equation for elimination anyway. If we do computation on paper, then we would like the relevant coefficient to be close to 1; if it is done on a computer, then one picks the one with the *largest* lead coefficient because this keeps the round-off errors down.) Example:

107

$$\left(\begin{array}{ccc|c} 0 & -2 & 3 & 3 \\ 2 & -1 & -2 & 6 \\ -5 & -2 & -1 & -3 \end{array}\right) \quad \longrightarrow$$

Special Recursive Case 2: Can't use any of the given equations to eliminate the first unknown. This is like "Special Base Case 2", in that it means that the first variable is not constrained at all by the given system. The algorithm should report back that the first variable can be chosen freely. Example:

108

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 3 \\ 2 & -4 & -2 & 7 \\ -5 & 10 & -1 & -9 \end{array}\right) \quad \longrightarrow$$

However, the example also shows that this brings us to a situation where we have *two* equations for *one* unknown — a case we have not had to consider before. So this special case forces us to consider *more general* systems of linear equations, where the number of unknowns does not necessarily match the number of equations.

General Gaussian elimination Luckily, the algorithm does not need much adjustment. What we have called the "Recursive Case" or "General Case" stays the same and we only have to reconsider the Base Case. This now splits into two:

Base Case 1 Only one unknown is left over but there may be more than one equation. In this case we solve each equation independently and compare the answers; if they agree, then that is what we return. If they disagree, then the system has no solution (which we can communicate back by raising an exception).

Base Case 2 Only one equation is left over but more than one unknown appears in it (let's say m many). In this case $m-1$ unknowns can be chosen freely and the other one is computed from those choices and the equation. (It does not matter which $m-1$ unknowns are chosen and which is computed.)

The second Base Case should be illustrated with an example. Suppose we end up in a situation where only this equation remains

$$x-2y+z=3$$

The answer to return to the previous level of the recursion should be:

109

Generalising Gaussian elimination in this way does not affect the strategies we follow in the special cases discussed in this chapter. We can, however, be more precise about what to do in "Special Recursive Case 2" above. In this case we remember that the first variable is not constrained and run the algorithm recursively on the n equations for only $n-1$ unknowns. When we get an answer from the recursive call we report this back up and also report that the first variable can be chosen freely.

Special cases in the compact notation. The above analysis is attractive as it explains exactly what needs to be done in each special case; in fact, we could use it to implement Gaussian elimination on a computer. For working on paper, however, a global geometric description is more helpful.

Remember that without the special cases, Gaussian elimination leads to a regular even-spaced "staircase":

$$\left(\begin{array}{cccc|c} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{array}\right) \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \left(\begin{array}{cccc|c} \bullet & * & * & * & * \\ 0 & \bullet & * & * & * \\ 0 & 0 & \bullet & * & * \\ 0 & 0 & 0 & \bullet & * \end{array}\right)$$

To have no special case occur during the elimination is the same as saying that the entries represented as bullets are different from zero. The special cases have the effect that some steps are wider than others (but the height of the steps is still at most one); we get the *echelon form* for general systems:

$$\left(\begin{array}{ccccc|c} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{array}\right) \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \left(\begin{array}{ccccc|c} 0 & \bullet & * & * & * & * \\ 0 & 0 & 0 & \bullet & * & * \\ 0 & 0 & 0 & 0 & \bullet & * \\ 0 & 0 & 0 & 0 & 0 & ? \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

As before, the bullets indicate entries different from zero, and the asterisks arbitrary entries. If the question mark is nonzero, then there is no solution. If it is zero, then we have infinitely many solutions because one or several variables can be chosen freely. The ones that can be chosen freely are those where the staircase does not drop down one level (x_1 and x_3 in the example). The other ones are computed but their value depends on what was chosen for the free ones.

To give a numeric example, the above final echelon form could have the entries

$$\left(\begin{array}{ccccc|c} 0 & 2 & -3 & 7 & 0 & 4 \\ 0 & 0 & 0 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & -3 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

We record the result of the computation as

$$\begin{array}{lcl} x_1 & : & \text{chosen freely} \\ x_2 & = & 2+\frac{3}{2}x_3 \\ x_3 & : & \text{chosen freely} \\ x_4 & = & 0 \\ x_5 & = & -2 \end{array}$$

Extended Gaussian elimination. Once we have reached echelon form, and once we have checked that the system is not contradictory, we can continue to eliminate entries *above* the pivots (i.e., the bullets in our diagrams). In the examples above, for example, we could carry on in the following way (assuming the question mark is a zero):

$$\left(\begin{array}{ccccc|c} 0 & \bullet & * & * & * & * \\ 0 & 0 & 0 & \bullet & * & * \\ 0 & 0 & 0 & 0 & \bullet & * \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \left(\begin{array}{ccccc|c} 0 & 1 & * & 0 & 0 & * \\ 0 & 0 & 0 & 1 & 0 & * \\ 0 & 0 & 0 & 0 & 1 & * \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

and this is of help for writing down the general solution. In the numerical example we get:

$$\left(\begin{array}{ccccc|c} 0 & 2 & -3 & 7 & 0 & 4 \\ 0 & 0 & 0 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & -3 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \quad \longrightarrow \quad \left(\begin{array}{ccccc|c} 0 & 1 & -\frac{3}{2} & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Other methods. In school you may have learned that besides the elimination method, there is also the *substitution method* for solving systems of linear equations. In fact, it is not really different, but actually just another way of writing down the steps of the algorithm. However, the chance of making an arithmetic error with the substitution method is *much greater* than with Gaussian elimination, so I do not recommend it!

You may also have learned about determinants and *Cramer's rule* which gives the solutions as the quotient of two determinants. Computationally, this is a *bad idea* for three reasons: Cramer's rule cannot cope with special cases, and it requires many more arithmetic operations than Gaussian elimination, and those operations are numerically unstable.

Finally, I must admit that Gaussian elimination is also not ideal from a computational point of view. When we have a system with many variables (say, more than 2,000) then round-off errors tend to accumulate leading to completely wrong "solutions". Instead, people use *iterative* methods which approximate the solution vector step by step. A discussion of these is beyond the scope of this module.

Systems of linear equations over other fields. If we reflect on the work we have done in this chapter then we may notice that we only used the *four basic arithmetic operations*, addition, subtraction, multiplication and (in the very last step of Gaussian elimination) division. From this we can conclude that Gaussian elimination can be employed for systems of equations *over an arbitrary field*, for example, the finite field $\mathrm{GF}(2)$ discussed previously.

Exercises

For each of the following systems of linear equations, transcribe into the compact notation and solve by Gaussian elimination:

1. $$\begin{array}{rcrcrcr} x_1 & + & 2x_2 & + & x_3 & = & -1 \\ 2x_1 & - & x_2 & + & x_3 & = & 1 \\ -2x_1 & + & x_2 & - & 2x_3 & = & 2 \end{array}$$

2. $$\begin{array}{rcrcrcr} & & x_1 & - & 2x_2 & = & 0 \\ 2x_1 & - & 2x_2 & - & 3x_3 & = & 0 \\ 4x_2 & - & 3x_3 & - & 4x_4 & = & 0 \\ 6x_3 & - & 4x_4 & - & 5x_5 & = & 0 \\ & & 8x_4 & - & 5x_5 & = & 1 \end{array}$$

3. $$\begin{array}{rcrcrcr} 2x_1 & - & x_2 & - & x_3 & = & 3 \\ x_1 & + & 2x_2 & + & 2x_3 & = & -1 \\ 3x_1 & + & x_2 & + & x_3 & = & 3 \end{array}$$

4. $$\begin{array}{rcrcrcr} -2x_1 & + & 2x_2 & - & x_3 & = & 4 \\ 3x_1 & + & 2x_2 & + & 2x_3 & = & -1 \\ -x_1 & - & 4x_2 & - & x_3 & = & -3 \end{array}$$

5. The following system of equations is to be read as being over $\mathrm{GF}(2)$. Compute *all* solutions.

$$\begin{array}{rcl} x_1+x_2+x_3+x_4+x_5 & = & 1 \\ x_1+x_3+x_5 & = & 1 \\ x_1+x_4 & = & 1 \end{array}$$

Practical advice

In the exam, I expect you to be able to solve a general system of linear equations, whether it is over the ordinary numbers or over the finite field $\mathrm{GF}(2)$. In particular, you must be able to

- translate from the explicit notation with unknowns (as in Box 92) to the compact notation introduced in Section 10.3, Box 97, and back;
- recognise when a system is contradictory and has no solution;
- recognise when there are infinitely many solutions and clearly write down which variables can be freely chosen and which have to be computed and how. If the field is $\mathrm{GF}(2)$, then there are only *two choices* for the freely chosen variables, so we can actually list them all.

Please remember...

- Always work with the compact notation and transform the number scheme until the general staircase pattern (the echelon form) has been achieved.
- Always report the result in the form outlined above at the end of Section 10.4.

10 Systems of linear equations and Gaussian elimination

92

$$\begin{aligned} x_1 + 5x_2 - 2x_3 &= -11 && \text{(Eq. 1)} \\ 3x_1 - 2x_2 + 7x_3 &= 5 && \text{(Eq. 2)} \\ -2x_1 - x_2 - x_3 &= 0 && \text{(Eq. 3)} \end{aligned}$$

93

$$\begin{aligned} -17x_2 + 13x_3 &= 38 && \text{(Eq. 2)} \\ 9x_2 - 5x_3 &= -22 && \text{(Eq. 3)} \end{aligned}$$

94

$$32x_3 = -32 \qquad \text{(Eq. 3)}$$

95

$$-17x_2 - 13 = 38 \qquad \text{(Eq. 2)}$$

96

$$x_1 - 15 + 2 = -11 \qquad \text{(Eq. 1)}$$

97

$$\left(\begin{array}{ccc|c} 1 & 5 & -2 & -11 \\ 3 & -2 & 7 & 5 \\ -2 & -1 & -1 & 0 \end{array}\right)$$

98

$$\left(\begin{array}{ccc|c} 3 & 1 & 5 & 3 \\ 0 & 2 & 3 & -2 \\ 0 & -2 & 2 & 7 \end{array}\right)$$

99

$$\left(\begin{array}{ccc|c} 3 & 1 & 5 & 3 \\ 0 & 2 & 3 & -2 \\ 0 & 0 & 5 & 5 \end{array}\right)$$

100

$$\left(\begin{array}{ccc|c} 3 & 1 & 5 & 3 \\ 0 & 2 & 3 & -2 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

101

$$\left(\begin{array}{ccc|c} 3 & 1 & 5 & 3 \\ 0 & 2 & 0 & -5 \\ 0 & 0 & 1 & 1 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 3 & 1 & 0 & -2 \\ 0 & 2 & 0 & -5 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

102

$$\left(\begin{array}{ccc|c} 6 & 2 & 0 & -4 \\ 0 & 2 & 0 & -5 \\ 0 & 0 & 1 & 1 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 6 & 0 & 0 & 1 \\ 0 & 2 & 0 & -5 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

103

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & 1/6 \\ 0 & 1 & 0 & -5/2 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

104

$$\left(\begin{array}{rcrcr} x_1 & & & = & 1/6 \\ & x_2 & & = & -5/2 \\ & & x_3 & = & 1 \end{array}\right)$$

105

$$\begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ 2x_1 & - & 4x_2 & = & 7 \end{array} \longrightarrow \begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ & & 0 & = & 1 \end{array}$$

$$\left(\begin{array}{cc|c} 1 & -2 & 3 \\ 2 & -4 & 7 \end{array}\right) \longrightarrow \left(\begin{array}{cc|c} 1 & -2 & 3 \\ 0 & 0 & 1 \end{array}\right)$$

106

$$\begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ 2x_1 & - & 4x_2 & = & 6 \end{array} \longrightarrow \begin{array}{rcrcl} x_1 & - & 2x_2 & = & 3 \\ & & 0 & = & 0 \end{array}$$

$$\left(\begin{array}{cc|c} 1 & -2 & 3 \\ 2 & -4 & 6 \end{array}\right) \longrightarrow \left(\begin{array}{cc|c} 1 & -2 & 3 \\ 0 & 0 & 0 \end{array}\right)$$

107

$$\left(\begin{array}{ccc|c} 0 & -2 & 3 & 3 \\ 2 & -1 & -2 & 6 \\ -5 & -2 & -1 & -3 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 2 & -1 & -2 & 6 \\ 0 & -2 & 3 & 3 \\ -5 & -2 & -1 & -3 \end{array}\right)$$

108

$$\left(\begin{array}{ccc|c} 1 & -2 & 1 & 3 \\ 2 & -4 & -2 & 7 \\ -5 & 10 & -1 & -9 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & -2 & 1 & 3 \\ 0 & 0 & -4 & 1 \\ 0 & 0 & 4 & 6 \end{array}\right)$$

109

$$\begin{array}{rcl} x & = & 3+2y-z \\ y & : & \text{chosen freely} \\ z & : & \text{chosen freely} \end{array}$$

Answers to exercises

1. $x_1 = 2, x_2 = 0, x_3 = -3$
2. $x_1 = 1, x_2 = 1/2, x_3 = 1/3, x_4 = 1/4, x_5 = 1/5$
3. No solution
4. x_3 can be chosen freely; following this choice, we get $x_2 = 1 - \frac{1}{10}x_3$ and $x_1 = -1 - \frac{3}{5}x_3$.
5. x_4 and x_5 can be chosen freely and the other three then compute as $x_1 = x_4 + 1$, $x_2 = x_4$, $x_3 = x_4 + x_5$. There are only two values that can be chosen for x_4 and x_5, respectively, since we are working over GF(2). So we can list all *four*

solutions:

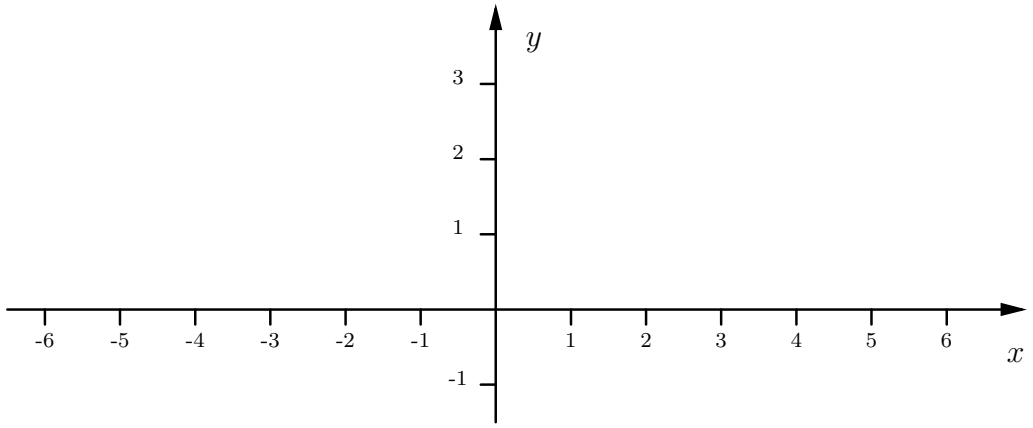
x_1	x_2	x_3	x_4	x_5
1	0	0	0	0
1	0	1	0	1
0	1	1	1	0
0	1	0	1	1

11 Analytic Geometry

11.1 Cartesian coordinate systems and points in the plane

Recall the idea of a **coordinate system**:

110



It consists of two **axes**, usually referred to as the *x*-**axis** (in horizontal direction) and the *y*-**axis** (in vertical direction). The axes are at right angles to each other. Their intersection point is called the **origin**. Furthermore, a **unit of length** is chosen on the axes.

A coordinate system allows us to identify a point P in the plane with its **coordinates** $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$: From the origin, go p_1 units along the x-axis (to the right if p_1 is positive, and to the left if it is negative), then go p_2 units parallel to the y-axis (up if p_2 is positive, down if it is negative).[10]

The coordinates of a point are unique once the axes and the unit of length have been chosen.

On a computer screen the origin is usually placed at the top left corner and the y-axis points *downward*.

Distance between points. Given points P and Q with coordinates $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$ and $\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$, their **distance** d is computed with the help of the Pythagorean Theorem:

$$d = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

111

This works for negative coordinates just as well as for positive ones.

[10] Most textbooks write the two coordinates of a point in a *row* rather than a *column*, that is, in the form (p_1, p_2). This is purely to save space in printing; I prefer the column form because it is visually more appealing and suggestive.

11.2 Vectors and straight line movements

We can interpret a pair of numbers $\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ also as a **movement** of the plane: every point $P = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$ is shifted to $P' = \begin{pmatrix} p_1 + v_1 \\ p_2 + v_2 \end{pmatrix}$.

112

Instead of a single movement, we can also see this as moving the points in the plane by v_1 units parallel to the x-axis, and v_2 units parallel to the y-axis.

We use uppercase letters for points: P, Q, R,..., and lowercase letters with arrows for movements: $\vec{v}$, $\vec{w}$, $\vec{u}$,... Instead of "(straight line) movement" we also say **vector**.

We allow vectors to be added to points:

$$P' = P + \vec{v}$$

because the coordinates of P after moving it along $\vec{v}$ are computed as $\begin{pmatrix} p_1 + v_1 \\ p_2 + v_2 \end{pmatrix}$.

A vector has a **length**, again computed by the Pythagorean Theorem:

$$|\vec{v}| = \sqrt{v_1^2 + v_2^2}$$

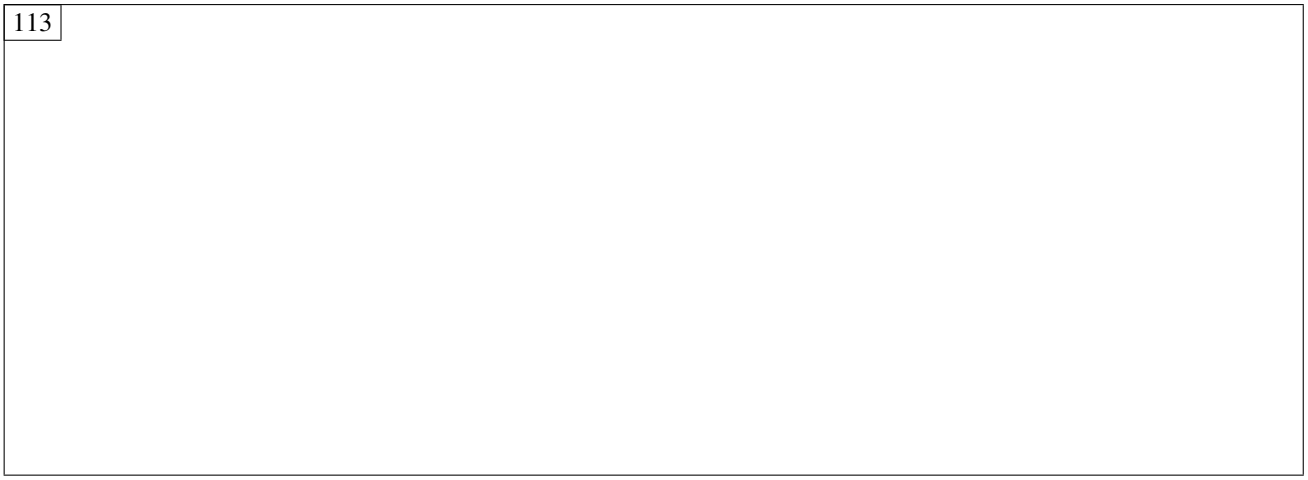
which is the distance each point travels under the movement described by $\vec{v}$. A vector of length 1 is called a **unit vector**; $\vec{0} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the **null vector**. For every vector $\vec{v} \neq \vec{0}$ there is a unit vector pointing in the same direction; it is computed as

$$\frac{\vec{v}}{|\vec{v}|} = \begin{pmatrix} v_1/|\vec{v}| \\ v_2/|\vec{v}| \end{pmatrix}$$

Given two points P and Q we have the movement $\overrightarrow{PQ}$ that moves P into Q; it has the coordinates $\begin{pmatrix} q_1 - p_1 \\ q_2 - p_2 \end{pmatrix}$.

Movements as an algebra. Movements can be composed: write $\vec{v} + \vec{w}$ for the movement of first following $\vec{v}$, then $\vec{w}$. The result is another straight line movement with coordinates $\begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \end{pmatrix}$.

113



Movements can also be extended or shrunk by a **factor** or **scalar**. For example,

$$\frac{1}{2}\cdot\vec{v}=\begin{pmatrix} v_1/2 \\ v_2/2 \end{pmatrix}$$

is the movement that shifts points in the same direction as $\vec{v}$ but only half as far. If the scalar is negative then the new vector points in the *opposite direction*. This is called **scalar multiplication**.

The laws of vector algebra.

114: Laws of vector addition

$$\begin{aligned} \vec{u}+\vec{v} &= \vec{v}+\vec{u} \\ \vec{u}+(\vec{v}+\vec{w}) &= (\vec{u}+\vec{v})+\vec{w} \end{aligned}$$

115: Law for the null vector

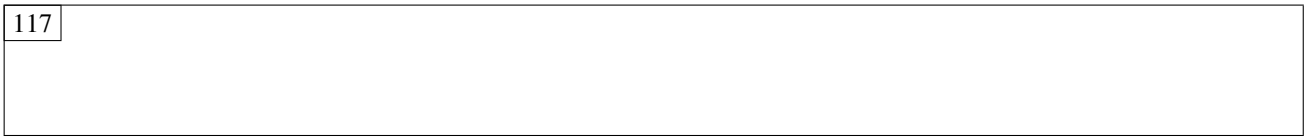
$$\vec{v}+\vec{0} = \vec{v}$$

116: Laws of scalar multiplication

$$\begin{aligned} 1\cdot\vec{v} &= \vec{v} \\ 0\cdot\vec{v} &= \vec{0} \\ s\cdot\vec{0} &= \vec{0} \\ (s+t)\cdot\vec{v} &= s\cdot\vec{v}+t\cdot\vec{v} \\ s\cdot(\vec{v}+\vec{w}) &= s\cdot\vec{v}+s\cdot\vec{w} \\ (st)\cdot\vec{v} &= s\cdot(t\cdot\vec{v}) \end{aligned}$$

Note how the laws of scalar multiplication are similar to those in a ring; the only law that is missing, is $a\times b=b\times a$; this is because it makes no sense to exchange a scalar (a number) for a vector and vice versa. We also have additive inverses:

117



On the other hand, the idea of a multiplicative inverse makes no sense.

11.3 Analytic geometry in the plane

Representing straight lines. Given a point P and a vector $\vec{v}\neq\vec{0}$, we can consider all points X that one can reach from P by following some distance along the direction of $\vec{v}$:

118

All such points lie on a straight line. We write

$$X = P + s \cdot \vec{v}$$

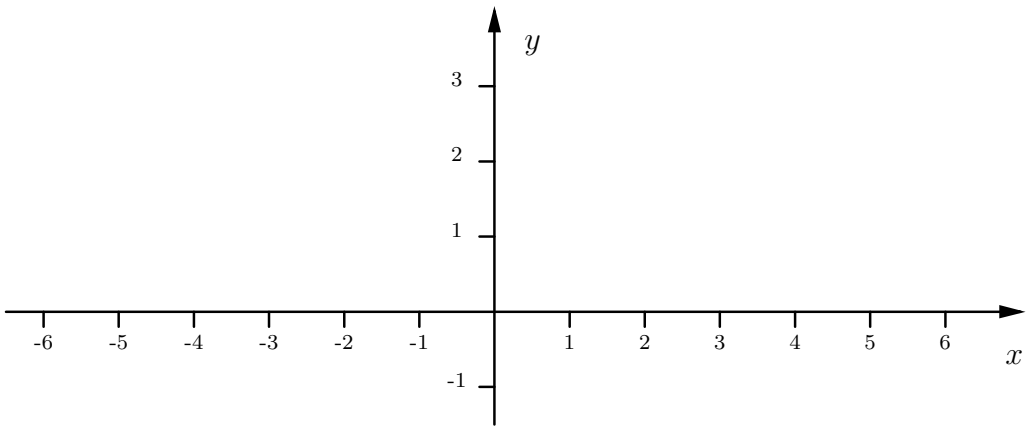
for this. Note that s is allowed to be any positive or negative number.

The expression $X = P + s \cdot \vec{v}$ is called a **parametric representation** of a line where the **parameter** is s. It can also be said to be a **generating expression** because in any application we would have P and $\vec{v}$ given as pairs of numbers, and the presentation would allow us to *generate* points on the line by simply choosing some value for the scalar s and computing the resulting coordinates for X. Example:

119

$$X = \begin{pmatrix} -2 \\ -1 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

120



If we are given two different points P and Q in the plane then this defines a straight line whose generating expression is $X = P + s \cdot \overrightarrow{PQ}$.

Intersection of two lines. Given two straight lines described by $X = P + s \cdot \vec{v}$ and $Y = Q + t \cdot \vec{w}$ we can find their **point of intersection** as follows:

The intersection point satisfies

$$X = Y$$

so

$$\begin{pmatrix} p_1 + sv_1 \\ p_2 + sv_2 \end{pmatrix} = P + s \cdot \vec{v} = Q + t \cdot \vec{w} = \begin{pmatrix} q_1 + tw_1 \\ q_2 + tw_2 \end{pmatrix}$$

This gives us two ordinary equations, one for each coordinate:

$$\begin{array}{rcl} p_1 + sv_1 & = & q_1 + tw_1 \\ p_2 + sv_2 & = & q_2 + tw_2 \end{array}$$

The unknowns are s and t; s will tell us how far in direction $\vec{v}$ we have to move from P in order to reach the point of intersection. Likewise, t will tell us how far in direction $\vec{w}$ we have to move from Q in order to reach the intersection point. Obviously, it will be enough to compute **either** s or t to find the coordinates of the intersection point. We do so through the system of linear equations which we obtain by considering each coordinate separately:

$$\begin{array}{rcl} v_1 s - w_1 t & = & q_1 - p_1 \\ v_2 s - w_2 t & = & q_2 - p_2 \end{array}$$

We can attack this with Gaussian elimination. Example:

121

The two lines intersect at point $A = Q + 1 \cdot \vec{w} = \begin{pmatrix} -1 \\ 3 \end{pmatrix} + \begin{pmatrix} 3 \\ -2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$.

11.4 Analytic geometry in three dimensions

Cartesian coordinate systems and points in space. For a coordinate system in three dimensions we need a third axis, usually called the z**-axis**. To make it "cartesian" the z-axis must be orthogonal (at right angles) to the other two axes, and the unit of length must be chosen the same on all three.

An example would be to let the x-axis be horizontal pointing to the East, the y-axis horizontal pointing North, and the z-axis pointing vertically upwards.

Once a coordinate system has been chosen, points in space are determined by *three coordinates*: $P = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$. The distance between two points $P = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$ and $Q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$ is computed as

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2}$$

Vectors and lines. A movement along a straight line for a certain distance is again called a **vector**; it is determined by its three coordinates. The length of the vector $\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$ is

$$\sqrt{v_1^2 + v_2^2 + v_3^2}$$

Any two points P and Q determine a vector $\overrightarrow{PQ}$ whose coordinates are $\begin{pmatrix} q_1 - p_1 \\ q_2 - p_2 \\ q_3 - p_3 \end{pmatrix}$.

The generating expression for a line remains the same

$$X = P + s \cdot \vec{v}$$

but we must remember that P and $\vec{v}$ now have three, rather than two coordinates.

Planes. The parametric representation of a plane has the form

$$X = P + s \cdot \vec{v} + t \cdot \vec{w}$$

where P is a point in space, and $\vec{v}$ and $\vec{w}$ are vectors (neither of which is the null vector). Furthermore, $\vec{w}$ must not point in the same (or in the opposite) direction as $\vec{v}$, or otherwise only a line is generated.

Three points P, Q, and R which are not all on the same line determine a plane:

$$X = P + s \cdot \overrightarrow{PQ} + t \cdot \overrightarrow{PR}$$

122

Intersection tasks. Consider the following tasks:

1. test whether a given point lies on a given line or plane;
2. find the intersection point between two lines;
3. find the intersection point between a line and a plane;
4. find the intersection line between two planes.

All of these can be solved in the same way as described in Item 11.3 on page 68, that is, by expressing the task as a system of linear equations, one for each coordinate.

We do an example for task 4. Consider the following two planes

$$X = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} + s \cdot \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix} + t \cdot \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$$
$$Y = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + r \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + q \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

We equate them and transform them into *three* linear equations for the *four* unknowns s, t, r, and q:

123

We solve the system with Gaussian elimination:

124

We find that q can be chosen freely and r computes to $-2+q/2$. Substituting this into the equation of the second plane, we obtain

125

which simplifies to

$$Y = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} + q \cdot \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

(where we have doubled the length of the vector component to avoid fractions). We could likewise have computed the relationship between s and t and would have found a representation of the line of intersection from the parametric representation of the first plane.

11.5 Lines and planes given by points

Two-point description of a line. Two (separate) points P and Q determine a line. We can easily translate this into the parametric representation: The vector that moves P into Q is denoted by $\overrightarrow{PQ}$ and has the coordinates $\begin{pmatrix} q_1 - p_1 \\ q_2 - p_2 \end{pmatrix}$ in 2D

and $\begin{pmatrix} q_1 - p_1 \\ q_2 - p_2 \\ q_3 - p_3 \end{pmatrix}$ in 3D. The parametric representation for the line through P and Q can therefore be written as

$$X = P + s \cdot \overrightarrow{PQ}$$

Using the laws of vector algebra we can write this as follows:

126

Two-point description of a line

$$X = (1 - s) \cdot P + s \cdot Q$$

(Sometimes this is written as $X = s \cdot P + t \cdot Q$ with the side condition $s + t = 1$.)

In this calculation we have applied scalar multiplication to points, something that we said before we would never do. Our excuse is that the operation we are defining here involves *two points and a scalar*; indeed, this operation is called a **convex combination of points**[11], especially when $0 \leq s \leq 1$. A picture is also helpful:

127

We see that the **line segment** between P and Q is characterised by the property that s is between 0 and 1. Convex combination is the basis for defining **Bezier curves**.

Three-point description of a plane. We do exactly the same as for lines but start with three points P, Q and R. A parametric representation of the plane through these three points is given by

$$X = P + s \cdot \overrightarrow{PQ} + t \cdot \overrightarrow{PR}$$

Applying vector algebra just as in Box 126 we get

Three-point description of a plane

$$X = (1 - s - t) \cdot P + s \cdot Q + t \cdot R$$

The part of the plane where all three parameters s, t and $1 - s - t$ are between 0 and 1 is exactly the triangle with corners P, Q and R.

[11] Another term for this is **weighted average**.

11.6 Algebras of vectors (aka, vector spaces)

We have discussed the operations and laws of vectors in subsection 11.2 above. To repeat: We have a null vector, can add two vectors, and we can multiply vectors with a scalar. We can also solve the equation $\vec{v}+\vec{x}=\vec{0}$ for $\vec{x}$: Simply set $\vec{x}=(-1)\cdot\vec{v}$. (So here scalar multiplication immediately provides us with negation and, more generally, subtraction.) Any structure that carries these two operations and satisfies these laws is called an **algebra of vectors** or a **vector space**.

Although we humans find it difficult to imagine spaces of more than 3 dimensions, we can readily believe that the movements of n-dimensional space form an algebra of vectors: This would work exactly as in the 2- and 3-dimensional case. However, the idea of an algebra of vectors is more general and more abstract than "straight-line movements of a space". Indeed, we can go one step further and realise that the scalars could come from any field $\mathbb{K}$, for example $\mathrm{GF}(2)$. One then speaks of an "algebra of vectors over $\mathbb{K}$."

Subalgebras. If $\vec{v}$ is an element of some vector algebra then we can generate a **subalgebra** (or more precisely, a **subalgebra of vectors**) by considering all vectors of the form $s\cdot\vec{v}$. This contains the null vector (because $\vec{0}=0\cdot\vec{v}$), is closed under addition, because

128

and is closed under scalar multiplication because

129

where we are using two of the laws of scalar multiplication. The same would be true if we took two vectors $\vec{v}$ and $\vec{w}$ and formed all expressions of the form $s\cdot\vec{v}+t\cdot\vec{w}$.

Another way of looking at our parametric representations, then, is to say that we pick a point and allow all movements from a *subalgebra* to act on this point. This construction leads to what in general is called an **affine subspace**. In other words, lines and planes are affine subspaces of 2D/3D.

Exercises

1. Compute the distance between $P=\begin{pmatrix}-1\\2\end{pmatrix}$ and $Q=\begin{pmatrix}2\\6\end{pmatrix}$.
2. Compute the point in which the line
$$X=\begin{pmatrix}-1\\1\\-2\end{pmatrix}+s\cdot\begin{pmatrix}2\\-2\\3\end{pmatrix}$$
and the plane
$$Y=\begin{pmatrix}4\\0\\-1\end{pmatrix}+t\cdot\begin{pmatrix}1\\1\\0\end{pmatrix}+u\cdot\begin{pmatrix}0\\1\\1\end{pmatrix}$$
meet.
3. Compute the line of intersection between the two planes
$$X=\begin{pmatrix}3\\4\\0\end{pmatrix}+s\cdot\begin{pmatrix}2\\2\\3\end{pmatrix}+t\cdot\begin{pmatrix}2\\0\\1\end{pmatrix}\qquad Y=\begin{pmatrix}-1\\3\\-2\end{pmatrix}+u\cdot\begin{pmatrix}1\\-1\\-1\end{pmatrix}+r\cdot\begin{pmatrix}1\\0\\0\end{pmatrix}$$

Practical advice

In the exam I expect you to be able to

- set up the generating expression for a line (or a plane) when given two (respectively, three) points;

- test whether a given point lies on a given line (or plane);
- find the intersection point between two lines or a line and a plane;
- find the intersection line between two planes;
- know the laws of vector algebra;
- do all of these things also over GF(2).

Notes

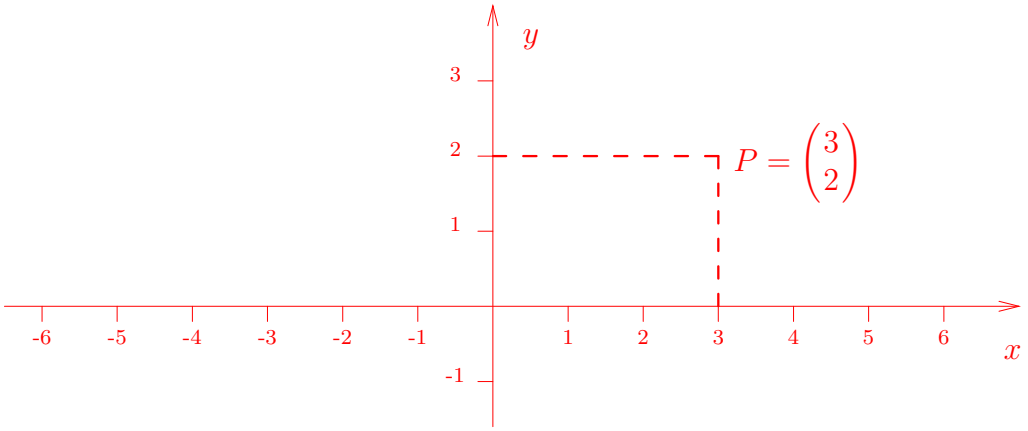
- Always write the coordinates of points and vectors in the column form. This will help you to avoid mistakes such as confusing coordinates (particularly in 3D).
- A line has infinitely many generating expressions. Any point P on the line and any vector $\vec{v}$ pointing in the direction of the line will do. This has two consequences:
  - Your solution and my model solution could look different although they both describe the same line.
  - Sometimes your computation will produce a direction vector with fractions as coordinates. You can avoid these by simply stretching the direction vector. For example, the two generating expressions

$$P + s \cdot \begin{pmatrix} 2/3 \\ 1/2 \end{pmatrix} \qquad \text{and} \qquad P + s \cdot \begin{pmatrix} 4 \\ 3 \end{pmatrix}$$

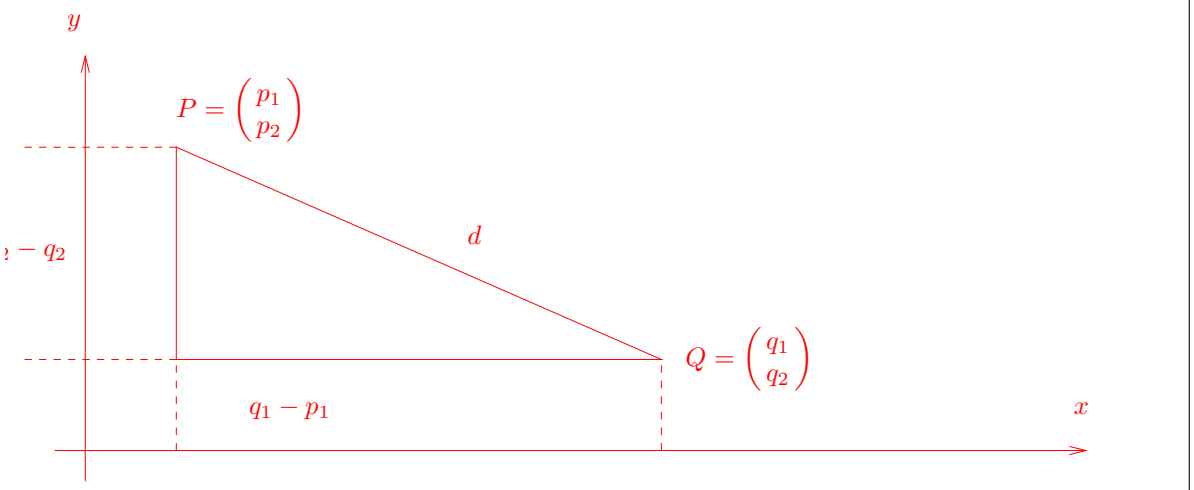
    describe the same line.
- The same is true for planes in 3D, only more so!
- When computing the intersection point between two lines (or a line and a plane, or two planes) with Gaussian elimination, make sure you are not using the same parameter name twice! (Note how I used r and q in Section 11.4 for the second plane.)

11 Analytic Geometry

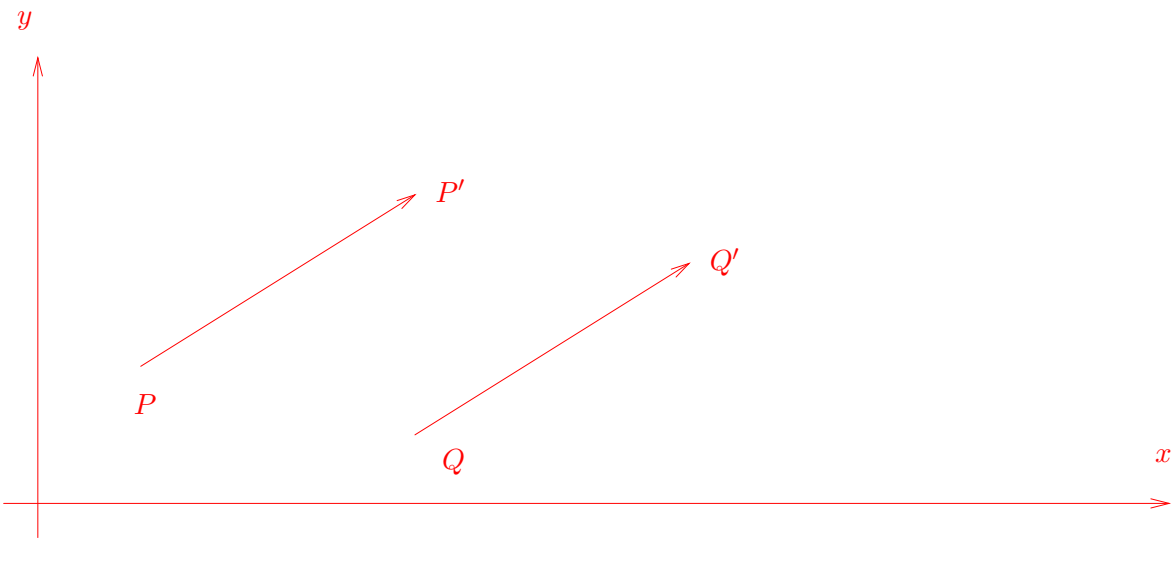
110



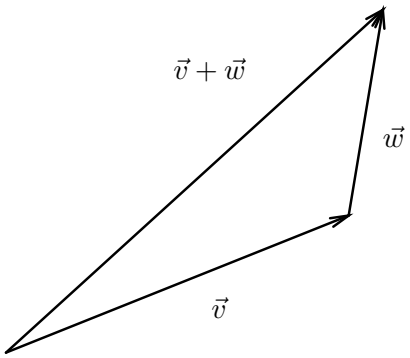
111



112



113



114: Laws of vector addition

$$\begin{aligned} \vec{u}+\vec{v} &= \vec{v}+\vec{u} \\ \vec{u}+(\vec{v}+\vec{w}) &= (\vec{u}+\vec{v})+\vec{w} \end{aligned}$$

115: Law for the null vector

$$\vec{v}+\vec{0} = \vec{v}$$

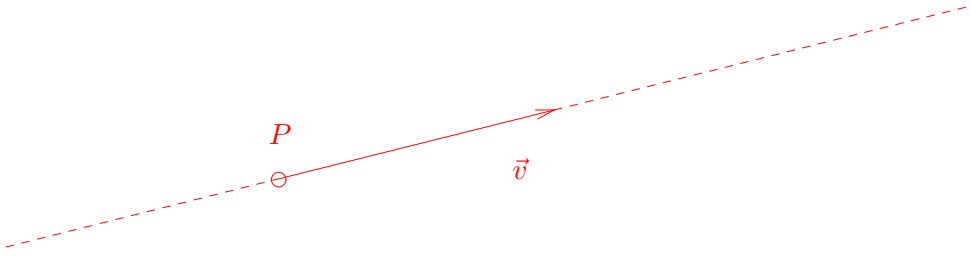
116: Laws of scalar multiplication

$$\begin{aligned} 1\cdot\vec{v} &= \vec{v} \\ 0\cdot\vec{v} &= \vec{0} \\ s\cdot\vec{0} &= \vec{0} \\ (s+t)\cdot\vec{v} &= s\cdot\vec{v}+t\cdot\vec{v} \\ s\cdot(\vec{v}+\vec{w}) &= s\cdot\vec{v}+s\cdot\vec{w} \\ (st)\cdot\vec{v} &= s\cdot(t\cdot\vec{v}) \end{aligned}$$

117

$$\vec{v}+(-1)\cdot\vec{v} = 1\cdot\vec{v}+(-1)\cdot\vec{v} = (1+(-1))\cdot\vec{v} = 0\cdot\vec{v} = \vec{0}$$

118



119

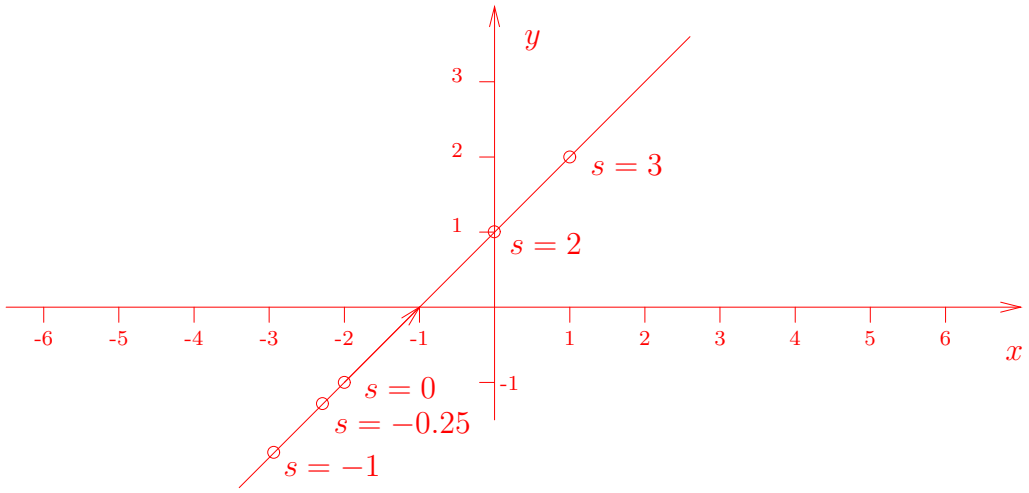
$$X = \begin{pmatrix} -2 \\ -1 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$s = 0 \quad \Rightarrow \quad X = \begin{pmatrix} -2 \\ -1 \end{pmatrix}$ $\qquad$ $s = -1 \quad \Rightarrow \quad X = \begin{pmatrix} -3 \\ -2 \end{pmatrix}$

$s = 1 \quad \Rightarrow \quad X = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$ $\qquad$ $s = -0.25 \quad \Rightarrow \quad X = \begin{pmatrix} -2.25 \\ -1.25 \end{pmatrix}$

$s = 2 \quad \Rightarrow \quad X = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

120



121

$$\begin{aligned} X &= \begin{pmatrix} 1 \\ -1 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ Y &= \begin{pmatrix} -1 \\ 3 \end{pmatrix} + t \cdot \begin{pmatrix} 3 \\ -2 \end{pmatrix} \end{aligned}$$

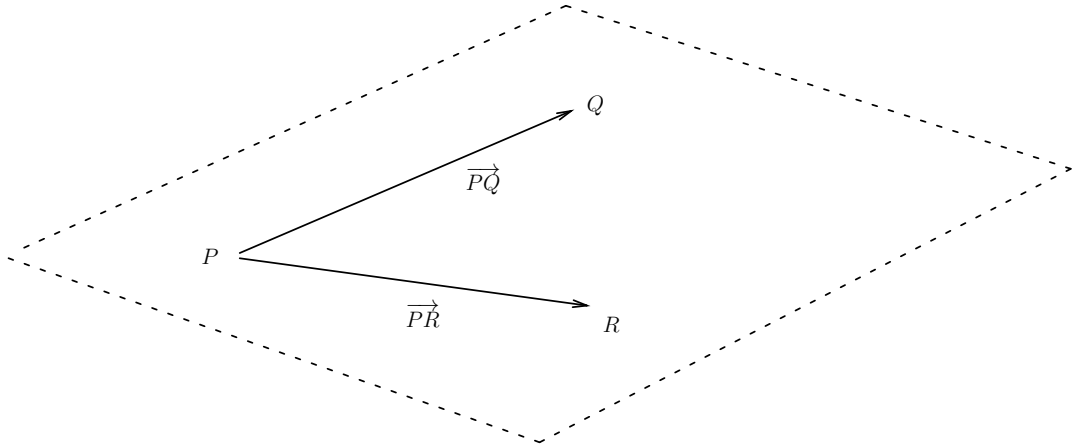
$$\begin{aligned} 1 + s &= -1 + 3t \\ -1 + 2s &= 3 - 2t \end{aligned}$$

$$\begin{aligned} s - 3t &= -2 \\ 2s + 2t &= 4 \end{aligned}$$

$$\begin{aligned} s - 3t &= -2 \\ 8t &= 8 \end{aligned}$$

$$t = 1$$

122



123

$$\begin{array}{rcl} 1-t & = & 2+r \\ -2+2s+t & = & 1+r \\ 1+s-t & = & q \end{array} \qquad \begin{array}{rcl} -t-r & = & 1 \\ 2s+t-r & = & 3 \\ s-t-q & = & -1 \end{array}$$

124

$$\left(\begin{array}{cccc|c} 0 & -1 & -1 & 0 & 1 \\ 2 & 1 & -1 & 0 & 3 \\ 1 & -1 & 0 & -1 & -1 \end{array}\right) \Longrightarrow \left(\begin{array}{cccc|c} 1 & -1 & 0 & -1 & -1 \\ 0 & -1 & -1 & 0 & 1 \\ 2 & 1 & -1 & 0 & 3 \end{array}\right) \Longrightarrow$$

$$\left(\begin{array}{cccc|c} 1 & -1 & 0 & -1 & -1 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 3 & -1 & 2 & 5 \end{array}\right) \Longrightarrow \left(\begin{array}{cccc|c} 1 & -1 & 0 & -1 & -1 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & -4 & 2 & 8 \end{array}\right)$$

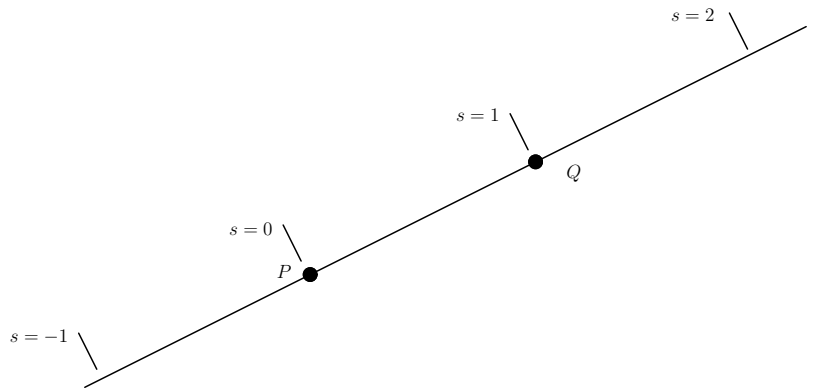
125

$$Y = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + \left(\begin{pmatrix} -2 \\ -2 \\ 0 \end{pmatrix} + q \cdot \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix} \right) + q \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

126

$$\begin{array}{rcl} X & = & P+s\cdot(Q-P) \\ & = & P+s\cdot Q-s\cdot P \\ & = & (1-s)\cdot P+s\cdot Q \end{array}$$

127



128

$$s\cdot\vec{v}+t\cdot\vec{v}=(s+t)\cdot\vec{v}$$

129

$$s\cdot(t\cdot\vec{v})=(st)\cdot\vec{v}$$

Answers to exercises

1. The distance is $\sqrt{((-1)-2)^2+(2-6)^2}=\sqrt{25}=5$.

2. We get the system of linear equations (one for each of the three coordinates):

$$\begin{array}{rcl} -1+2s &=& 4+t \\ 1-2s &=& t+u \\ -2+3s &=& -1+u \end{array}$$

   Collecting all unknowns on the left hand side and all constants on the right hand side we get

$$\begin{array}{rrrcr} 2s & -t & & = & 5 \\ -2s & -t & -u & = & -1 \\ 3s & & -u & = & 1 \end{array} \quad\rightarrow\quad \begin{array}{rrrcr} 2s & -t & & = & 5 \\ & -2t & -u & = & 4 \\ & 3t & -2u & = & -13 \end{array} \quad\rightarrow\quad \begin{array}{rrrcr} 2s & -t & & = & 5 \\ & -2t & -u & = & 4 \\ & & -7u & = & -14 \end{array}$$

   From this we get $u=2$, $t=-3$, and $s=1$, so the intersection point has coordinates $\begin{pmatrix}-1\\1\\-2\end{pmatrix}+\begin{pmatrix}2\\-2\\3\end{pmatrix}=\begin{pmatrix}1\\-1\\1\end{pmatrix}$.

3. We set up a system of equations:

$$\begin{array}{rcl} 3+2s+2t &=& -1+u+r \\ 4+2s &=& 3-u \\ 3s+t &=& -2-u \end{array}$$

   which we solve by Gaussian elimination (but from right to left)

$$\begin{array}{rrrrcr} 2s & +2t & -u & -r & = & -4 \\ 2s & & +u & & = & -1 \\ 3s & +t & +u & & = & -2 \end{array} \quad\rightarrow\quad \begin{array}{rrrrcr} 2s & +2t & -u & -r & = & -4 \\ 2s & & +u & & = & -1 \\ s & +t & & & = & -1 \end{array}$$

   From this we see that s can be chosen freely and t has to be set to $-1-s$, so we get for the generating expression for the line of intersection: $X=\begin{pmatrix}3\\4\\0\end{pmatrix}+s\cdot\begin{pmatrix}2\\2\\3\end{pmatrix}+(-1-s)\cdot\begin{pmatrix}2\\0\\1\end{pmatrix}=\begin{pmatrix}1\\4\\-1\end{pmatrix}+s\cdot\begin{pmatrix}0\\2\\2\end{pmatrix}$.

12 The inner product

12.1 Equations in analytic geometry

A line from a linear equation. We start with an example. Consider the linear equation

$$x_1 - 2x_2 = 3$$

According to Chapter 10, the solutions are given by

$$\begin{array}{lcl} x_2 & : & \text{choose freely} \\ x_1 & = & 3 + 2x_2 \end{array}$$

We write the possible solutions in vector form:

130

and we see that the solutions all lie on a line. This is true in general and worth highlighting:

Theorem

The solution space of a linear equation in two unknowns is a line in 2D.

A plane from a linear equation. Again, we start with an example. Consider the linear equation

$$x_1 - 2x_2 + 3x_3 = 5$$

The solutions are given by

$$\begin{array}{lcl} x_3 & : & \text{choose freely} \\ x_2 & : & \text{choose freely} \\ x_1 & = & 5 + 2x_2 - 3x_3 \end{array}$$

and in vector form we get:

131

Theorem

The solution space of a linear equation in three unknowns is a plane in 3D.

Intersection tasks revisited. Intersection tasks become very easy when one of the geometric entities is given in the parametric representation and the other as an equation. We illustrate with an example, intersecting a line with a plane in 3D:

$$\text{line:} \quad X = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}$$

$$\text{plane:} \quad x_1 - 2x_2 + 3x_3 = 5$$

We substitute the line into the equation, using the fact that the coordinates of points on the line are

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2+s \\ 1 \\ 1-3s \end{pmatrix}$$

132

The intersection point A has coordinates

We could ask why this calculation is so much simpler than what we did in Chapter 11, where we had to use Gaussian elimination. One way to understand this is to remember that a parametric representation *generates points*. As a picture:

133

In contrast, an equation *tests* a given point:

134

If we intersect a geometric object given in a parametric representation with one given by an equation, then these two fit together perfectly to give us a condition on the parameter s:

135

12.2 Translating between the parametric and equational representations.

In the last item we have seen that the equational representation is computationally very helpful, but how does one get it when the geometric object is given in parametric form? This can be answered in complete generality.

Lines. If we are given the parametric representation of a line in 2D

$$X = P + s \cdot \vec{v} = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + s \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

and we are looking for an equation

$$ax_1 + bx_2 = d$$

which describes the same line, then all we need to do is set

$$\begin{array}{rcl} a & = & -v_2 \\ b & = & v_1 \\ d & = & ap_1 + bp_2 \end{array}$$

Let's test that this formula works:

136

Planes. If we are given the parametric representation of a plane in 3D

$$X = P + s \cdot \vec{v} + t \cdot \vec{w} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} + s \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + t \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

and we are looking for an equation

$$ax_1 + bx_2 + cx_3 = d$$

which describes the same plane, then all we need to do is set

$$\begin{array}{rcl} a & = & v_2w_3 - v_3w_2 \\ b & = & v_3w_1 - v_1w_3 \\ c & = & v_1w_2 - v_2w_1 \\ d & = & ap_1 + bp_2 + cp_3 \end{array}$$

For the translation in the other direction I gave examples in items 12.1 and 12.1 already; it is the same as solving the equation.

Practical advice

In the exam I expect you to be able to

- translate between the parametric and the equational representation of a line or a plane;
- solve intersection tasks that involve an equational representation of a line or a plane.

Notes

The formulas in Section 12.2 may seem "magic" and hard to remember. The following may help:

- For the coefficients of the equation of a line, just exchange the two coordinates of the direction vector and change the sign of one of them.
- For the coefficients of the equation of a plane, remember the **determinant**:

  $$\begin{vmatrix} a & c \\ b & d \end{vmatrix} = ad - bc$$

  which can be read out as "multiply the two entries on the main diagonal and subtract from this the product of the two entries on the other diagonal."

  The three coefficients for the equation of the plane can be written as determinants:

  $$a = \begin{vmatrix} v_2 & w_2 \\ v_3 & w_3 \end{vmatrix} \qquad b = -\begin{vmatrix} v_1 & w_1 \\ v_3 & w_3 \end{vmatrix} \qquad c = \begin{vmatrix} v_1 & w_1 \\ v_2 & w_2 \end{vmatrix}$$

In both cases we get the right-hand side d of the equation by plugging the coordinates of the point P into the equation.

12.3 The geometric interpretation of equational presentations.

Given a linear equation

$$ax_1 + bx_2 + cx_3 = d$$

we assemble the coefficients in a vector:

$$\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Geometrically, this vector is **orthogonal**[12] ("at right angles") to the plane — unlike $\vec{v}$ and $\vec{w}$ from the parametric presentation which lie *inside* the plane. A 2D picture:

137

This is called the **normal** or a **normal vector** to the plane.[13] Note that a normal must not be the null vector but otherwise it can be arbitrarily long, and also, it doesn't matter whether it points up or down.

The right-hand side d of the equation also has a geometric interpretation. It is negative if the origin is on the same side of the plane as the normal and positive otherwise. Its value tells us something about the **distance** of the origin from the plane, because

Distance of origin from plane or line

$$\text{distance} = \frac{d}{|\vec{n}|}$$

(Remember that $|\vec{n}|$ is the length of $\vec{n}$ computed as $\sqrt{a^2+b^2+c^2}$.)

From this we see that if we make $\vec{n}$ have length 1 then d gives us exactly the distance; that's why some books insist that a normal should always have length 1.

Note that the expression $d/|\vec{n}|$ could be positive or negative contradicting the everyday use of the word "distance." That is because it is giving us additional information about which side of the plane the origin is located.

In any case, because of this geometric interpretation, the equational presentation is also called the **normal form** of the line (in 2D) or plane (in 3D).

A new notation. The expression $ax_1 + bx_2 + cx_3$ on the left-hand side of an equation (describing a plane) can be viewed as obtaining a *number* from the two *tuples* $\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ and $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$. It is called the **inner product**[14] of $\vec{n}$ and X, and is denoted by $\langle \vec{n}, X \rangle$. Let's highlight the definition:

[12] Another commonly used word for this is **perpendicular**.

[13] The name "normal" is historical and of no geometric significance in itself.

[14] Also called **scalar product** (because numbers are also called scalars) or **dot product** (because it is sometimes written as $\vec{n} \cdot X$.

The inner product of two tuples

$$\text{If } \vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \text{ and } \vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \text{ then } \langle \vec{v}, \vec{w} \rangle = v_1 w_1 + v_2 w_2 + v_3 w_3$$

Note that this is a purely formal definition for us and we don't need to know whether the tuples in question are to be interpreted as vectors or as the coordinates of some point. Let's compute an example:

138

$$\vec{v} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} -2 \\ 3 \\ -1 \end{pmatrix}$$

$$\langle \vec{v}, \vec{w} \rangle =$$

Given this simple definition it is easy to check the following laws (which are analogous to the laws of multiplication of numbers, but note that there can not be an associativity law):

The laws of the inner product

$$\begin{aligned} \langle \vec{0}, \vec{v} \rangle &= 0 \\ \langle \vec{v}, \vec{w} \rangle &= \langle \vec{w}, \vec{v} \rangle \\ \langle \vec{v} + \vec{w}, \vec{x} \rangle &= \langle \vec{v}, \vec{x} \rangle + \langle \vec{w}, \vec{x} \rangle \\ \langle s \cdot \vec{v}, \vec{w} \rangle &= s \times \langle \vec{v}, \vec{w} \rangle \end{aligned}$$

If you know the normal to a plane and a point P on it, then the normal form can be written as

The normal form of a line/plane

$$\langle \vec{n}, X \rangle = \langle \vec{n}, P \rangle$$

where we have used the fact that the three-tuple $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ can be read either as a vector $\vec{x}$ or as the coordinates of a point X.

12.4 Geometric properties of the inner product

Orthogonality test. Two vectors $\vec{v}$ and $\vec{w}$ are orthogonal ("at right angles") to each other exactly if $\langle \vec{v}, \vec{w} \rangle = 0$.

Angles. More generally, we have $\langle \vec{v}, \vec{w} \rangle = |\vec{v}| \times |\vec{w}| \times \cos(\alpha)$ where α is the angle between the vectors $\vec{v}$ and $\vec{w}$. From the formula one can see that $\langle \vec{v}, \vec{w} \rangle$ is positive exactly if the two vectors form an *acute* angle, and negative if that angle is *obtuse*.

Inner product and length. For any vector $\vec{v}$ we have

$$|\vec{v}| = \sqrt{\langle \vec{v}, \vec{v} \rangle}$$

which we can also write as

$$|\vec{v}|^2 = \langle \vec{v}, \vec{v} \rangle$$

Projection property. Given $\vec{n} \neq \vec{0}$ and $\vec{v}$ we can compute the number

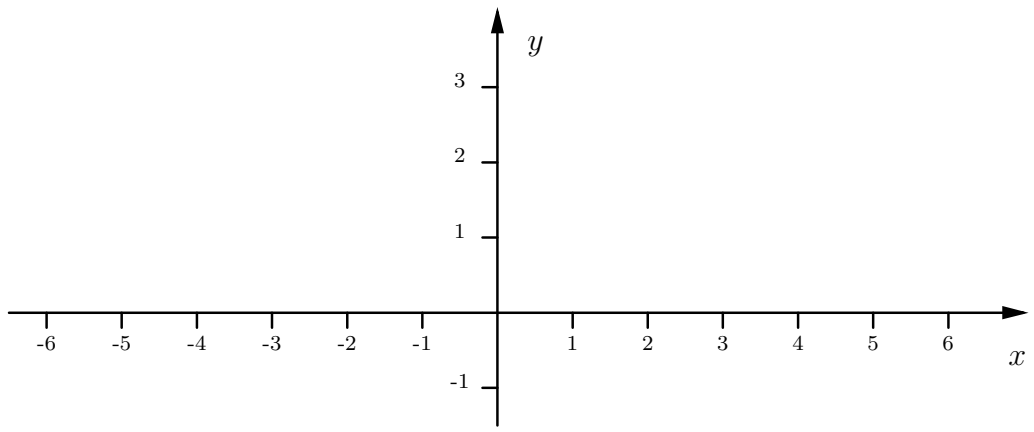
$$\frac{\langle \vec{n}, \vec{v} \rangle}{\langle \vec{n}, \vec{n} \rangle}$$

If we stretch $\vec{n}$ by this value, that is, if we compute

$$\frac{\langle \vec{n}, \vec{v} \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n}$$

then we get the **orthogonal projection** of $\vec{v}$ onto the line defined by $\vec{n}$. A picture:

139



12.5 Graphics-related tasks

We will now use these geometric properties to solve some common graphics-related problems:

Task 1: Computing the distance from a plane. If $\langle \vec{n}, X \rangle = \langle \vec{n}, P \rangle = d$ is the normal form of a plane E and Q a point, then the distance of Q to E (measured in the direction of the normal, that is, orthogonal to the plane) is given by

Distance from plane given by $\langle \vec{n}, X \rangle = d$

$$\frac{d - \langle \vec{n}, Q \rangle}{|\vec{n}|} = \frac{\langle \vec{n}, P \rangle - \langle \vec{n}, Q \rangle}{|\vec{n}|} = \frac{\left\langle \vec{n}, \overrightarrow{QP} \right\rangle}{|\vec{n}|}$$

If this expression returns 0 then Q is a point *in the plane*.

The result of the computation could be negative; if so, then the normal $\vec{n}$ is pointing *towards the side on which Q is located*. This property is used in graphics engines to determine if a point is before a plane (and therefore visible) or behind it (and therefore hidden).

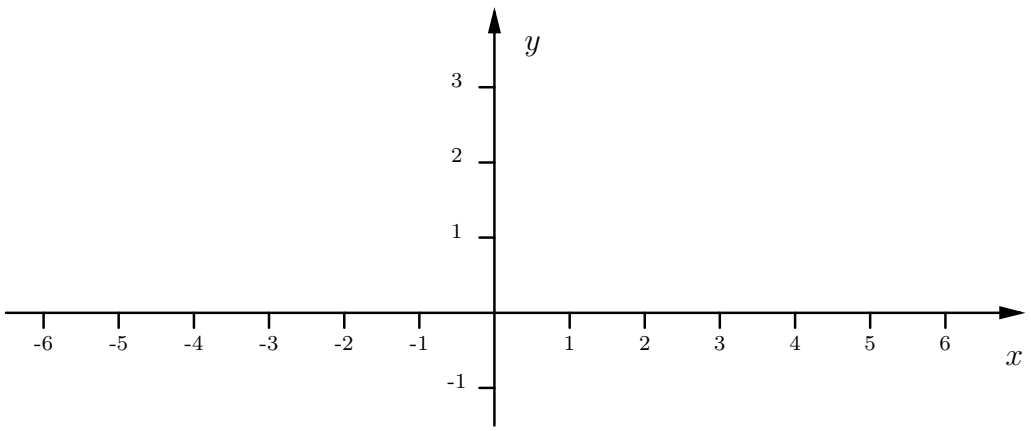
Task 2: Nearest neighbour. If $\langle \vec{n}, X \rangle = \langle \vec{n}, P \rangle = d$ is the normal form of a plane E and Q a point, then we ask which point on E is closest to Q. If you have sufficient spatial awareness then you know that we find this point if we move from Q to E *in the direction of the normal* $\vec{n}$. The formula for it is

Nearest neighbour to Q on plane given by $\langle \vec{n}, X \rangle = d$

$$Q' \quad = \quad Q + \frac{d - \langle \vec{n}, Q \rangle}{|\vec{n}|} \cdot \frac{\vec{n}}{|\vec{n}|} \quad = \quad Q + \frac{d - \langle \vec{n}, Q \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n}$$

If the plane goes through the origin, the right-hand side d of the normal form is 0 and the formula simplifies to $Q' = Q - \frac{\langle \vec{n}, Q \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n}$, and you will see the connection to the projection formula in the previous item: To reach Q' we travel back in direction of $\vec{n}$ towards the plane. A 2D picture:

140



Task 3: Reflecting a point at a plane. With the same notation as before, the formula is very similar to that for the nearest neighbour, except that we go *twice* the distance in direction of the normal:

Point Q reflected on plane given by $\langle \vec{n}, X \rangle = d$

$$Q'' \quad = \quad Q + 2 \times \frac{d - \langle \vec{n}, Q \rangle}{|\vec{n}|} \cdot \frac{\vec{n}}{|\vec{n}|} \quad = \quad Q + 2 \times \frac{d - \langle \vec{n}, Q \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n}$$

Task 4: Reflecting a line at a plane. This is now very easy: Just pick two (different) points on the line and reflect those according to the previous item. Then connect the two mirror images, and you are done!

Example. Let's do the example in 2D so that you can draw a picture and check my computations; so instead of using a plane for reflection we use a line E, given by

$$x_1 - 2x_2 = 1$$

and the line L that we want to reflect as

$$X = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

141

Exercises

1. Compute the normal form of the line $X = P + s \cdot \vec{v} = \begin{pmatrix} -1 \\ -1 \end{pmatrix} + s \cdot \begin{pmatrix} -4 \\ 3 \end{pmatrix}$.
   Draw a picture in a coordinate system that shows P, $\vec{v}$, and the normal $\vec{n}$.
2. Given the following three points

   $$P = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad Q = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \qquad R = \begin{pmatrix} 2 \\ 2 \\ 0 \end{pmatrix}$$

   find the parametric representation of the plane determined by them, and convert this into the normal form.

Practical advice

In the exam I expect you to be able to use the technology of this chapter to

- sketch the position of a line given in normal form in a coordinate system;
- determine whether two vectors are at right angles to each other;

- determine whether two points are on the same side of a plane or on different sides;
- compute the distance of a point to a plane;
- compute the nearest neighbour of a point on a plane;
- reflect a point at a plane;
- reflect a line at a plane;
- solve a *new* geometric task similar to the ones above.

Notes

- For the exam you need to remember the formula for the inner product of two vectors (given in Section 12.3 above).
- You also need to remember the orthogonality test that the inner product provides.
- You *don't* need to memorise the formulas for distance, nearest neighbour, or reflection; they will be given on the exam paper.

12 The inner product

130

$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3+2x_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} + x_2 \cdot \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

131

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5+2x_2-3x_3 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \\ 0 \end{pmatrix} + x_2 \cdot \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix} + x_3 \cdot \begin{pmatrix} -3 \\ 0 \\ 1 \end{pmatrix}$$

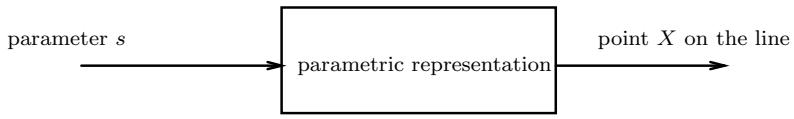
132

$$\begin{aligned} 2+s-2+3-9s &= 5 \\ -8s &= 2 \\ s &= -\tfrac{1}{4} \end{aligned}$$

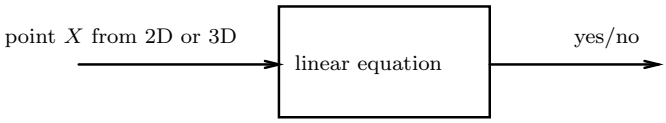
The intersection point A has coordinates

$$A = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} - \frac{1}{4} \cdot \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix} = \begin{pmatrix} 7/4 \\ 1 \\ 7/4 \end{pmatrix}$$

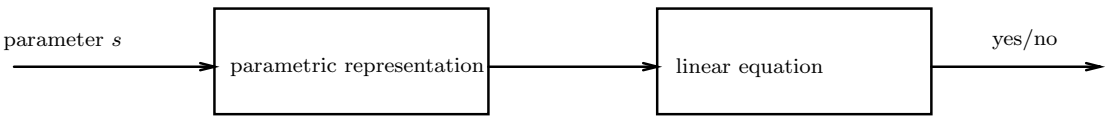
133



134



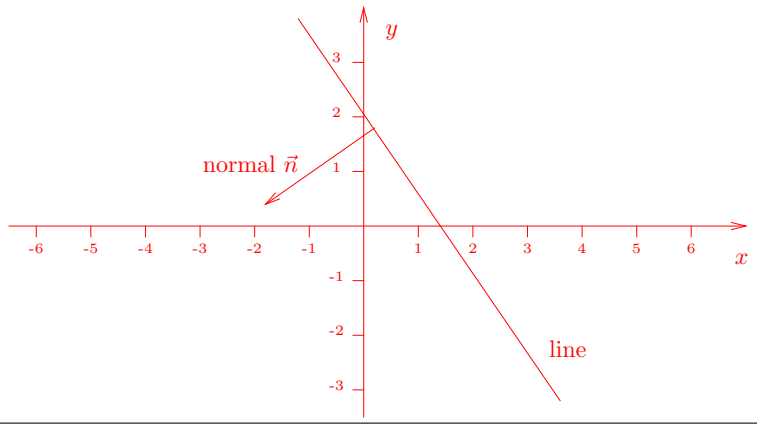
135



136

$X = \begin{pmatrix} p_1 + sv_1 \\ p_2 + sv_2 \end{pmatrix}$ substituted into the equation: $(-v_2)(p_1 + sv_1) + v_1(p_2 + sv_2) = -v_2 p_1 + v_1 p_2 - v_2 sv_1 + v_1 sv_2 = d$

137

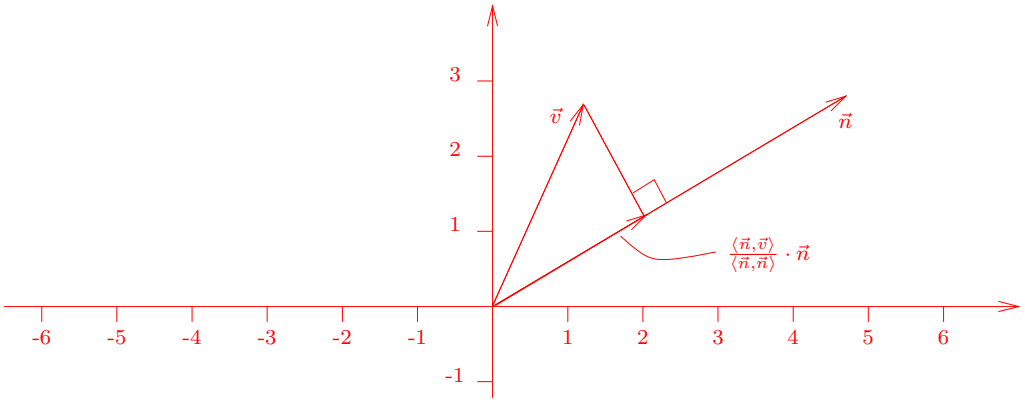


138

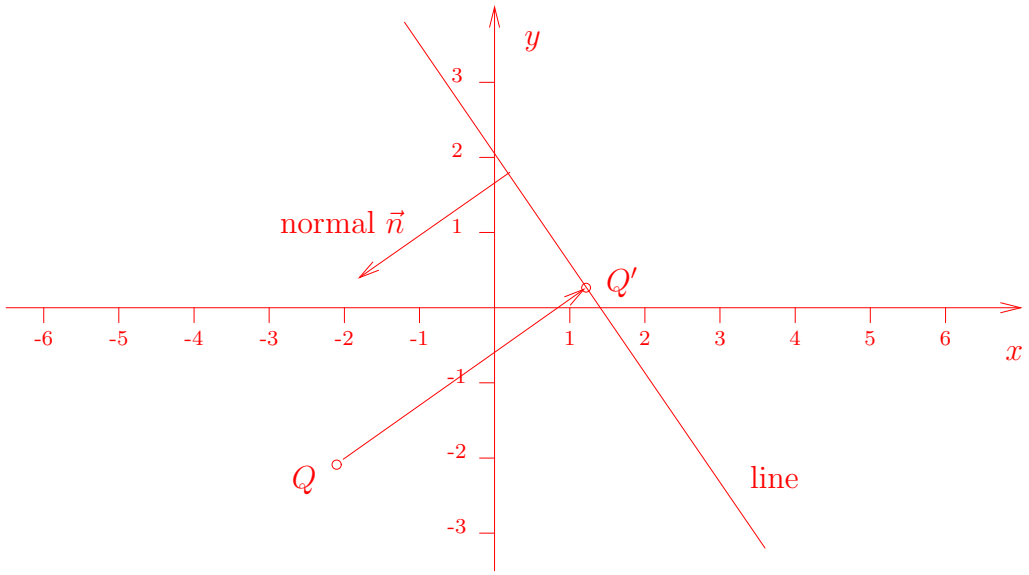
$$\vec{v} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix} \qquad \vec{w} = \begin{pmatrix} -2 \\ 3 \\ -1 \end{pmatrix}$$

$$\langle \vec{v}, \vec{w} \rangle = 1 \times (-2) + 0 \times 3 + 2 \times (-1) = -2 - 2 = -4$$

139



140



141

For the two points we can choose

$$A = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 0 \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \text{and} \qquad B = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 1 \cdot \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Reflecting A gives us

$$A'' = A + 2 \times \frac{d - \langle \vec{n}, A \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 2 \times \frac{1+2}{1+4} \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \frac{6}{5} \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \frac{1}{5} \cdot \begin{pmatrix} 6 \\ -7 \end{pmatrix} = \begin{pmatrix} 1.2 \\ -1.4 \end{pmatrix}$$

The same computation for B yields:

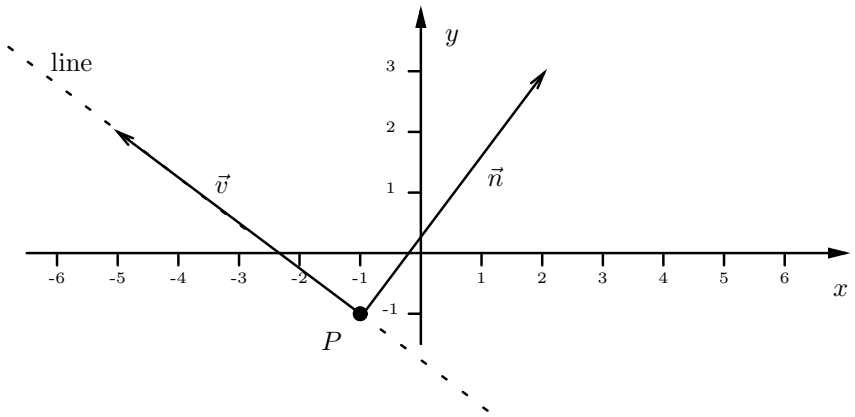
$$B'' = B + 2 \times \frac{d - \langle \vec{n}, B \rangle}{\langle \vec{n}, \vec{n} \rangle} \cdot \vec{n} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 2 \times \frac{1-1}{1+4} \cdot \begin{pmatrix} 1 \\ -2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and we see that B lies on E and does not change under reflection. Connecting A'' and B'' gives the parametric representation:

$$X = B'' + s \cdot \overrightarrow{B''A''} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + s \cdot \left(\begin{pmatrix} 1.2 \\ -1.4 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} + s \cdot \begin{pmatrix} 0.2 \\ -1.4 \end{pmatrix}$$

Answers to exercises

1. Change the direction vector $\vec{v} = \begin{pmatrix} -4 \\ 3 \end{pmatrix}$ to the normal vector: $\vec{n} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$. Set up the normal form for the line as $\langle \vec{n}, \vec{x} \rangle = \langle \vec{n}, P \rangle$ with $P = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ gives us $3x_1 + 4x_2 = -7$. Picture:



2. The parametric presentation derived from three points is computed as $X = P + s \cdot \overrightarrow{PQ} + t \cdot \overrightarrow{PR}$, so we get $X = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + s \cdot \begin{pmatrix} -1-1 \\ 0-2 \\ 1-3 \end{pmatrix} + t \cdot \begin{pmatrix} 2-1 \\ 2-2 \\ 0-3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + s \cdot \begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix} + t \cdot \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}$, which can be simplified to (because the length of a direction vector does not matter): $X = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + s \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + t \cdot \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}$. The coordinates of the normal vector $\vec{n}$ compute as

$$\begin{array}{rcrcr} a & = & -3-0 & = & -3 \\ b & = & 1-(-3) & = & 4 \\ c & = & 0-1 & = & -1 \end{array}$$

and the normal form is obtained as $\langle \vec{n}, \vec{x} \rangle = \langle \vec{n}, P \rangle$, so $-3x_1 + 4x_2 - x_3 = -3 + 8 - 3 = 2$.

13 Bases

Motivation. **Digital images** are composed of discrete pixels. For a grey-scale picture, every pixel is represented by a one-byte number, from 0 (for black) to 255 (for white). In a colour picture, every pixel is represented by three one-byte numbers, one each for red, green, and blue. Here is a 20×20 pixel colour image, greatly magnified:

This snippet was taken from a 1368×855 pixel colour image. To store all its pixels we require $1368 \times 855 \times 3 = 3,508,920$ bytes, or approximately 3.5MB. Storing the image in the `bmp` ("bitmap") format takes this amount of storage space.

If this image is transformed into the `jpeg` format, then only 600KB are required, a nearly six-fold saving. The translation between the `bmp` and the `jpeg` representation is pure linear algebra and in this chapter we will find out how it works.

To start with, a digital image (from now on let's always think of a grey-scale one) is just a very long tuple of natural numbers, in our example it is an element of $1368 \times 855 = 1,169,640$-dimensional "space". So where in the previous two chapters I have encouraged you to think of tuples as either the coordinates of points, or as the components of a straight-line movement, now we are interpreting them as digital images. This is no big deal but perhaps we should remember that not every tuple can be interpreted in this way, for example, negative entries are meaningless as there is no such thing as "negative luminescence".

13.1 Generating an algebra of vectors

We already touched upon this topic in Section 11.6 but let us be more general now. If we are given a collection $\vec{v}_1, \ldots, \vec{v}_n$ of vectors, then we can look at the algebra of vectors generated by them. The generation is done by forming all **linear combinations** of the given vectors, that is, we compute

$$a_1 \cdot \vec{v}_1 + \ldots + a_n \cdot \vec{v}_n$$

for all choices of the scalars $a_1, \ldots, a_n$. We will have to compute a lot of these expressions in this chapter, so let's introduce a more compact notation for them:

$$\sum_{i=1}^{n} a_i \cdot \vec{v}_i \;\; = \;\; a_1 \cdot \vec{v}_1 + \ldots + a_n \cdot \vec{v}_n$$

To get some practice in this new notation, let's check again that the collection of all linear combinations (also known as the **span** of the vectors $\vec{v}_1, \ldots, \vec{v}_n$) is itself an algebra of vectors. That's because linear combinations can be added:

142

and the result is itself a linear combination. Likewise, a linear combination can be multiplied with a scalar:

143

and again we get a linear combination of the original n vectors $\vec{v}_1, \ldots, \vec{v}_n$. Combining addition and scalar multiplication, we can say that a linear combination of linear combinations is again a linear combination of the original vectors.

13.2 Linear independence and bases

Now imagine that we have some algebra of vectors in mind and we are seeking to generate it from a small number of "basic" vectors. That's exactly what we did in Section 11.3, when we represented all the movements along a straight line as multiples of a basic vector $\vec{v}$. It makes sense that we should like to employ as few vectors as possible for the generation process. Therefore, the following properties should hold about our collection $\vec{v}_1, \ldots, \vec{v}_n$:

- the null vector $\vec{0}$ does not occur in the list $\vec{v}_1, \ldots, \vec{v}_n$ (because it doesn't generate anything);
- none of the v_i is a linear combination of the remaining $n-1$ vectors (because that would be a redundancy).

If these two conditions are satisfied, then we say that the collection $\vec{v}_1, \ldots, \vec{v}_n$ is **linearly independent**. We can express the two conditions very neatly in a single criterion:

Theorem 8 *A collection $\vec{v}_1, \ldots, \vec{v}_n$ of vectors is linearly independent exactly if*

$$\sum_{i=1}^{n} a_i \cdot \vec{v}_i = \vec{0} \quad \textit{happens only in case} \quad a_1 = a_2 = \ldots = a_n = 0\,.$$

If every element of an algebra V of vectors can be written as a linear combination of the vectors $\vec{v}_1, \ldots, \vec{v}_n$ and if the v_i are linearly independent, then we call $\vec{v}_1, \ldots, \vec{v}_n$ a **basis** of V.

Time for an example. Consider the algebra of all four-tuples of (rational) numbers (interpreted, if you wish, as the movements of four-dimensional space, or as 2×2 digital images). The following is the **standard basis** for this algebra:

144

Linear independence is easy to check for these four vectors using the criterion of Theorem 8:

145

How about the following four vectors?

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}, \quad \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

This is where Gaussian elimination can help. The equation $\sum_{i=1}^{4} a_i \cdot \vec{v}_i = \vec{0}$ amounts to four linear equations in the four unknowns a_1, a_2, a_3 and a_4. Since the right-hand side of each equation equals zero, we always have $a_1 = a_2 = a_3 = a_4 = 0$ as one solution, so the system can't be contradictory. If there is also a non-zero solution, then Gaussian elimination will find it:

146

We get a perfect echelon form which shows that there is only one solution; that solution is $a_1 = a_2 = a_3 = a_4 = 0$. (In a moment we will have an even better criterion for the linear independence of these four vectors which will allow us to avoid Gaussian elimination.)

13.3 Coordinates

Whenever we have a basis $\vec{v}_1, \ldots, \vec{v}_n$ for an algebra of vectors, any given vector $\vec{w}$ can be written as a linear combination of the $\vec{v}_i$. That's because the basis generates every vector. So if

$$\vec{w} = \sum_{i=1}^{n} a_i \cdot \vec{v}_i$$

then we can call the numbers $a_1, a_2, \ldots, a_n$ the **coordinates** of $\vec{w}$ with respect to the basis $\vec{v}_1, \ldots, \vec{v}_n$. Now you might wonder whether the coordinates are uniquely determined, or whether it is possible that $\vec{w}$ has two different representations:

$$\vec{w} = \sum_{i=1}^{n} a_i \cdot \vec{v}_i = \sum_{i=1}^{n} b_i \cdot \vec{v}_i$$

The first option is correct and the second option cannot happen. Here is the argument:

147

So once we have a basis, the coordinates of each vector are uniquely determined, and in fact, we can then think of the algebra as an algebra of n-tuples. This is exactly what we have been doing in Section 11 when we discussed movements in the plane and in 3D space with respect to a fixed basis (which at the time we called a "coordinate system").

13.4 Dimension of an algebra of vectors

If we have found a basis $\vec{v}_1, \ldots, \vec{v}_n$ for an algebra of vectors, is it possible that there is another one with fewer elements? That is not the case. The number of elements in a basis is an intrinsic feature of the algebra; it is called its **dimension**. Let us state this formally:

Theorem 9 *Any two bases of an algebra of vectors have the same number of elements.*

This may be intuitive but a proof is not so easy to come by. We give one in the case that the dimension is finite. So assume that both $\vec{v}_1, \ldots, \vec{v}_n$ and $\vec{w}_1, \ldots, \vec{w}_m$ are a basis for an algebra and our aim is to show that $n = m$. The crucial proof idea is contained in the following:

Lemma 10 *Let both $\vec{v}_1, \ldots, \vec{v}_n$ and $\vec{w}_1, \ldots, \vec{w}_m$ be a basis for an algebra of vectors. Let $\vec{v}_k$ be a member of the first basis. Then there exists a member $\vec{w}_p$ of the second basis that can replace $\vec{v}_k$ in the sense that $\vec{v}_1, \ldots, \vec{v}_{k-1}, \vec{w}_p, \vec{v}_{k+1}, \ldots, \vec{v}_n$ is also a basis.*

With this powerful lemma we can now prove our theorem: Given two bases, we can one-by-one replace all the $\vec{v}$'s with a $\vec{w}$. Doing so, we will never use the same $\vec{w}$ twice (or the result would not be a basis) and it follows that we must have started with at least as many $\vec{w}$'s as there were $\vec{v}$'s. Doing the same in the opposite direction, we conclude that there must have been at least as many $\vec{v}$'s as there were $\vec{w}$'s. These two statements can only be true if $n = m$, that is, both bases had the same number of elements.

13.5 All bases are not equal

So now we know that any two bases for the same algebra have the same number of basis vectors. However, depending on what we are trying to achieve, one basis may be a lot more convenient than another one. This is exactly the message that `jpeg`-compression is telling us. Let's start with an example of a tiny 2×2 digital (grey-scale) image, written as a four-tuple. In the standard basis, we can read off the coordinates immediately:

$$\begin{pmatrix} 10 \\ 10 \\ 11 \\ 12 \end{pmatrix} = 10 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + 10 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + 11 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + 12 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

but let's look at the same "image" represented in the basis given in Box 145:

148

We see that the first coordinate is much bigger than the other three and in fact, if we "forgot" about the last two, we would still get a reasonable approximation to the original image:

149

This example already illustrates the main idea behind `jpeg`-compression: Represent the image vector via its coordinates with respect to an alternative basis, *better suited to images*, and suppress all coordinates that are smaller than some threshold. In `jpeg` the transformation is done for each block of 8×8 pixels separately, something that is very visible if we set a very high compression factor:

(This is the same image fragment as displayed at the beginning of the chapter, but now highly compressed.)

13.6 Orthogonal and orthonormal bases

Wait a moment, you will say, where do we get the coordinates of the image vector with respect to the alternative basis from? Especially, since `jpeg`-compression works with 64-dimensional tuples. The answer comes from orthogonality as defined by the inner product.

So assume that we have a basis $\vec{v}_1, \ldots, \vec{v}_n$ where any two vectors are orthogonal to each other, that is, we have

$$\langle \vec{v}_i, \vec{v}_j \rangle = 0 \quad \text{whenever} \quad i \neq j$$

Then I claim that the coordinates a_k of a given vector $\vec{w}$ with respect to the $\vec{v}$'s are simply computed as

$$a_k = \frac{\langle \vec{w}, \vec{v}_k \rangle}{\langle \vec{v}_k, \vec{v}_k \rangle},$$

which is the formula we saw a lot in the previous chapter already. Why is this the case? We assumed that the $\vec{v}$'s form a basis and hence we know that

$$\vec{w} = \sum_{i=1}^{n} a_i \cdot \vec{v}_i$$

for *some* scalars $a_1, \ldots, a_n$. To show that the scalars have the form given above, we simply evaluate an inner product:

150

so we get the formula we claimed by solving for a_k.

If the computations are done on a computer, then we can make sure that each basis vector has length 1, so the division by $\langle \vec{v}_k, \vec{v}_k \rangle$ is not necessary when computing the coordinates. In that case one speaks of an **orthonormal basis**. If computations are done on paper, then it's better to avoid the surds[15] and stay with the merely *orthogonal basis*. You can check that the basis given in Box 148 is orthogonal and that the coordinates given in Box 149 can be computed by the formula above.

However, we have been a bit nonchalant in our computations above. For the moment, there is nothing to tell us that the inner product of a vector with itself is different from zero, and so the term $\frac{\langle \vec{w}, \vec{v}_k \rangle}{\langle \vec{v}_k, \vec{v}_k \rangle}$ may not exist. For the algebras of vectors considered in the last two chapters (i.e., tuples of rational numbers), this problem does not exist because here the following is true:

Positive definiteness of the inner product
$\langle \vec{v}, \vec{v} \rangle \geq 0$ $\langle \vec{v}, \vec{v} \rangle = 0 \implies \vec{v} = \vec{0}$

Over GF(2) we still have the inner product defined as in the last chapter, but it is not positive definite as the following example shows:

151 $\left\langle \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\rangle =$

Indeed, there is no positive definite inner product for vectors over GF(2) and orthogonal bases may not exist.

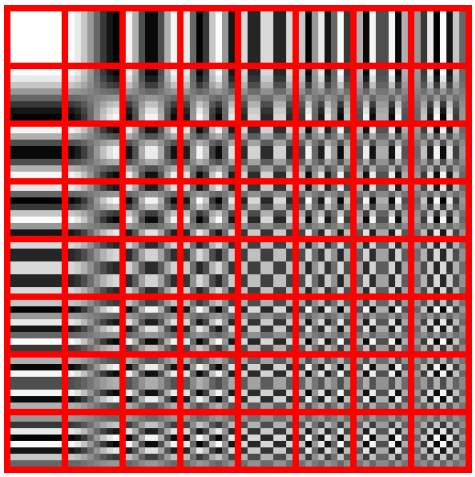
Creating an orthogonal basis. If we have a positive definite inner product then we can turn any basis into an orthogonal one. The idea is simple: we use the projection property (Section 12.4) to remove from each basis vector that part which is not orthogonal to the others. More formally, let $\vec{v}_1, \ldots, \vec{v}_n$ be a basis of our algebra. We define a new basis $\vec{w}_1, \ldots, \vec{w}_m$ whose members are orthogonal to each other:

$$\begin{array}{lcl} \vec{w}_1 & \stackrel{\text{def}}{=\!=} & \vec{v}_1 \\ \vec{w}_2 & \stackrel{\text{def}}{=\!=} & \vec{v}_2 - \frac{\langle \vec{v}_2, \vec{w}_1 \rangle}{\langle \vec{w}_1, \vec{w}_1 \rangle} \cdot \vec{w}_1 \\ & \vdots & \\ \vec{w}_n & \stackrel{\text{def}}{=\!=} & \vec{v}_n - \sum_{i=1}^{n-1} \frac{\langle \vec{v}_n, \vec{w}_i \rangle}{\langle \vec{w}_i, \vec{w}_i \rangle} \cdot \vec{w}_i \end{array}$$

It's a good exercise to show that each $\vec{w}_k$ is orthogonal to all the previous ones, i.e., to $\vec{w}_1, \ldots, \vec{w}_{k-1}$.

On a computer we would now divide each $\vec{w}_k$ by its length to obtain an orthonormal basis.

The 64 basis vectors used in `jpeg`-compression form an orthonormal basis. Here is a visual representation of them all (taken from Wikipedia):



[15] Expressions containing roots

To avoid any misunderstandings, the power of `jpeg`-compression does *not* come from the fact that these vectors form an orthonormal basis; this only helps in the calculations. After all, the standard basis is already orthonormal. It is rather that the pixels of real-life images are not random: If several neighbouring ones have similar values then it is likely that many more have similar values (e.g., a patch of blue sky), and if several neighbouring pixels show some oscillation then it is likely that the oscillation continues further (e.g., the leaves of a tree, or waves on a lake). The orthonormal basis of `jpeg`-compression picks up on these "oscillations" which is why it is also said that it represents the image in the "frequency domain". What we have here is an example of a **discrete Fourier transform**. There are many variations on this theme with multitude applications in computer science, but the underlying principle is always the same: representation in an alternate (orthonormal) basis.

Exercises

1. Show that the five vectors $\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$, $\begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix}$, $\begin{pmatrix} 0 \\ -1 \\ 3 \end{pmatrix}$, $\begin{pmatrix} 3 \\ -2 \\ 0 \end{pmatrix}$, and $\begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$ are linearly dependent.
2. Select three of them which form a basis for the algebra of three-tuples.
3. Turn it into an orthogonal basis using the procedure of Section 13.5 above.
4. Compute the coordinates of the vector $\vec{u} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ with respect to your orthogonal basis.

Practical advice

In the exam I expect you to be able to

- test whether a given set of vectors (over any field) is linearly independent;
- complete a given set of linearly independent vectors to a basis;
- transform a given basis into an orthogonal one.

13 Bases

142

$$\sum_{i=1}^{n} a_i \cdot \vec{v}_i + \sum_{i=1}^{n} b_i \cdot \vec{v}_i = \sum_{i=1}^{n} (a_i + b_i) \cdot \vec{v}_i$$

143

$$s \cdot (\sum_{i=1}^{n} a_i \cdot \vec{v}_i) = \sum_{i=1}^{n} (s \times a_i) \cdot \vec{v}_i$$

144

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

145

$$\vec{0} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = a_1 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + a_2 \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + a_3 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + a_4 \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \quad \text{implies} \quad a_1 = a_2 = a_3 = a_4 = 0$$

146

$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & -1 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -1 & 1 & -1 & 0 \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & -2 & -2 & 0 \\ 0 & -2 & -2 & 0 & 0 \\ 0 & -2 & 0 & -2 & 0 \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & -2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & -2 & -2 & 0 \end{array}\right) \rightarrow \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & -2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 \\ 0 & 0 & 0 & -4 & 0 \end{array}\right)$$

147

If $\sum_{i=1}^{n} a_i \cdot \vec{v}_i = \sum_{i=1}^{n} b_i \cdot \vec{v}_i$ then $\vec{0} = \sum_{i=1}^{n} (a_i - b_i) \cdot \vec{v}_i$ and by the independence criterion of Theorem 8 we get $a_i = b_i$ for all $i = 1, \ldots, n$.

148

$$\begin{pmatrix} 10 \\ 10 \\ 11 \\ 12 \end{pmatrix} = \frac{43}{4} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \frac{3}{4} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} + \frac{1}{4} \cdot \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} - \frac{1}{4} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

149

$$\frac{43}{4} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \frac{3}{4} \cdot \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 10 \\ 10 \\ 11.5 \\ 11.5 \end{pmatrix}$$

150

$$\begin{aligned}\langle \vec{w}, \vec{v}_k \rangle &= \langle \textstyle\sum_{i=1}^{n} a_i \cdot \vec{v}_i, \vec{v}_k \rangle && \text{(given)} \\ &= \textstyle\sum_{i=1}^{n} a_i \times \langle \vec{v}_i, \vec{v}_k \rangle && \text{(properties of the inner product)} \\ &= a_k \times \langle \vec{v}_k, \vec{v}_k \rangle && \text{(all other products are zero)}\end{aligned}$$

so we get the formula we claimed by solving for a_k.

151

$$\left\langle \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\rangle = 1 \times 1 + 1 \times 1 = 1 + 1 = 0$$

Answers to exercises

1. The algebra of three-tuples has dimension 3, so all bases consist of three vectors. In other words, a collection of five vectors must be linearly dependent.

2. We have two ways to find three vectors which form a basis. First, we use the original definition of linear dependence and check whether there is a null vector (there isn't) or whether some vectors are a linear combination of others. For this we can use Gaussian elimination:

$$\begin{pmatrix} 1 & -1 & 1 \\ 2 & -1 & -1 \\ 0 & -1 & 3 \\ 3 & -2 & 0 \\ 1 & -2 & 3 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & -3 \\ 0 & -1 & 3 \\ 0 & 1 & -3 \\ 0 & -1 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & -3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

   We see that the third and the fourth vector are obtainable as a linear combination of the first two, so they are redundant. The remaining three are linearly independent, as this elimination also shows.

   The other method is based on Theorem 8: To find out which three vectors form a basis, we use Gaussian elimination to solve the equation $a_1 \cdot \vec{v}_1 + \ldots + a_5 \cdot \vec{v}_5 = \vec{0}$:

$$\left(\begin{array}{ccccc|c} 1 & 2 & 0 & 3 & 1 & 0 \\ -1 & -1 & -1 & -2 & -2 & 0 \\ 1 & -1 & 3 & 0 & 3 & 0 \end{array}\right) \longrightarrow \left(\begin{array}{ccccc|c} 1 & 2 & 0 & 3 & 1 & 0 \\ 0 & 1 & -1 & 1 & -1 & 0 \\ 0 & -3 & 3 & -3 & 2 & 0 \end{array}\right) \longrightarrow \left(\begin{array}{ccccc|c} 1 & 2 & 0 & 3 & 1 & 0 \\ 0 & 1 & -1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{array}\right)$$

   We see that if we drop the third and the fourth vector then the remaining three lead to a regular echelon form without special cases, or in other words, then only the solution $a_1 = a_2 = a_5 = 0$ remains. (We could also have dropped the second and the third, or the second and the fourth.)

3. $$\vec{w}_1 = \vec{v}_1 = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

$$\vec{w}_2 = \vec{v}_2 - \frac{\langle \vec{v}_2, \vec{w}_1 \rangle}{\langle \vec{w}_1, \vec{w}_1 \rangle} \cdot \vec{w}_1 = \begin{pmatrix} 2 \\ -1 \\ -1 \end{pmatrix} - \frac{2}{3} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4/3 \\ -1/3 \\ -5/3 \end{pmatrix} \quad \text{Simplified:} \quad \begin{pmatrix} 4 \\ -1 \\ -5 \end{pmatrix}$$

$$\vec{w}_3 = \vec{v}_3 - \frac{\langle \vec{v}_3, \vec{w}_1 \rangle}{\langle \vec{w}_1, \vec{w}_1 \rangle} \cdot \vec{w}_1 - \frac{\langle \vec{v}_3, \vec{w}_2 \rangle}{\langle \vec{w}_2, \vec{w}_2 \rangle} \cdot \vec{w}_2 = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix} - \frac{6}{3} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} - \frac{-9}{42} \cdot \begin{pmatrix} 4 \\ -1 \\ -5 \end{pmatrix} = \begin{pmatrix} -2/14 \\ -3/14 \\ -1/14 \end{pmatrix} \text{ Simplified: } \begin{pmatrix} -2 \\ -3 \\ -1 \end{pmatrix}$$

4. We use the formula developed in Box 150 and the text before it:

$$a_1 = \frac{\langle \vec{u}, \vec{w}_1 \rangle}{\langle \vec{w}_1, \vec{w}_1 \rangle} = \frac{1}{3} \quad a_2 = \frac{\langle \vec{u}, \vec{w}_2 \rangle}{\langle \vec{w}_2, \vec{w}_2 \rangle} = \frac{1}{6} \quad a_3 = \frac{\langle \vec{u}, \vec{w}_3 \rangle}{\langle \vec{w}_3, \vec{w}_3 \rangle} = \frac{1}{2} \text{ so: } \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \frac{1}{3} \cdot \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} + \frac{1}{6} \cdot \begin{pmatrix} 1 \\ -1 \\ -2 \end{pmatrix} + \frac{1}{2} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$