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.